DS INDEX

Chapter ⇒1 Data Structure : Introduction

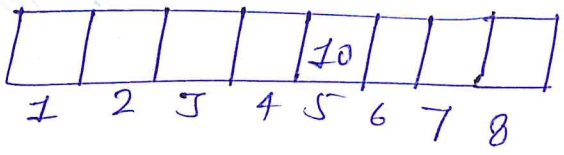
[18.1] What are data structures

⇒ Data structure is a data organization, management and storage format that enables efficient access and modification.

⇒ A data structure is collection of data values, the relationships among them, and the functions or operations that can be applied to the data.

What? data relationship, operations

arrays → A:



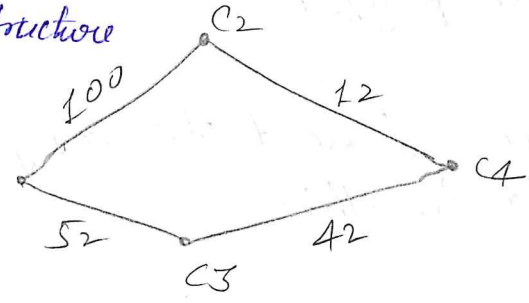
① arrays are the collection of same data types item.

② ordering of data element

③ operations:
set value. A[5] = 10
get value.
-traversal of array: simple for loop

by using index, I can directly to an element.

⇒ Graph data structure is used for storing these information & finding shortest path.



} Shortest path from C1 to C4

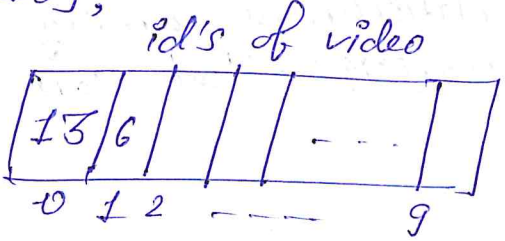
Why we need DS?

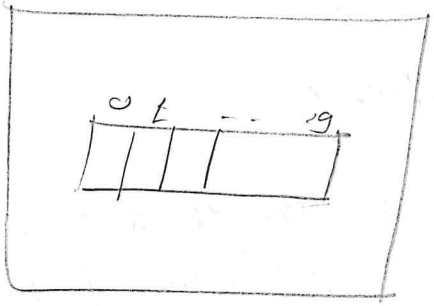
⇒ DS provides us efficient ways of storing information.

⇒ We come up with different data structures for solving more real world problems.

[18.2] Why do arrays fail?

Array → DS int P[10];



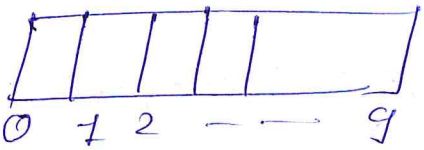


RAM

Can not grow size of array dynamically.

⇒ If I want to play video list in particular order, array gives us the order.

Problem: 10 videos int P[10]

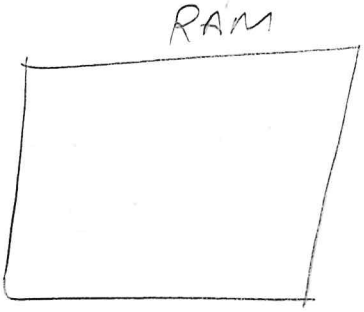


If I want to add 5 more video to the same playlist

int P1[15]; then I need to copy all previous data in this.

⇒ If I initialize array with large size

int P[250];



I have list of 10 video, which I will store in P.

Then I wasting space by allocating large memory.

Chapter ⇒ Singly Linked List

[19.1] Structure & memory organisation

⇒ As array can not grow dynamically. so we use linked-list

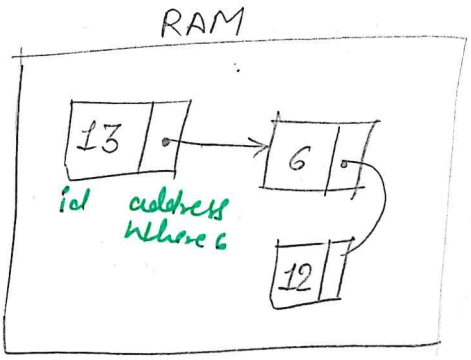
⇒ There are different types of linked lists:

- Singly Linked list.
- doubly linked list.
- Circular linked list.

⇒ Linked list have been used in computer from 1950's.

⇒ Dynamically grow and shrink of list what we use for storing data in data structure concept.

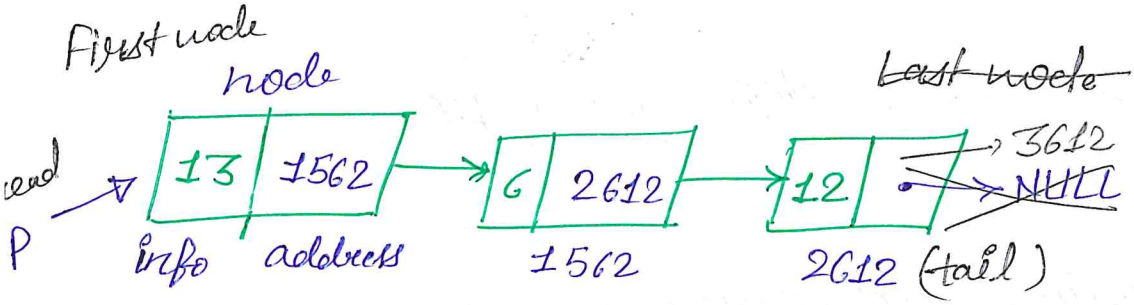
Create List dynamically



If there is only item in the song list with id 1

id 1 → 13
id 2 → 6 (added) with malloc (c)
id 3 → 12

⇒ In java, python pointers are known as references

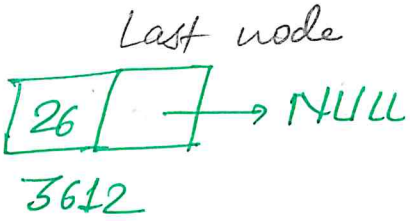


⇒ node basically consists of value and address

⇒ Array uses contiguous memory location but not linked list.

⇒ First node is refered as head node & last node is referred as tail.

⇒ If I want to add 26 one more node.



⇒ In C, we place both the (id, address) in the single structure.

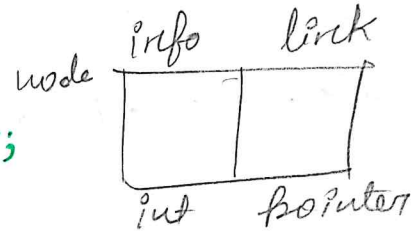
Struct → C
Class → C++, java, python

[19.2] Code for node

⇒ In C, We store a node using struct.

Program for single linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int info;
   struct node *link;
};
```



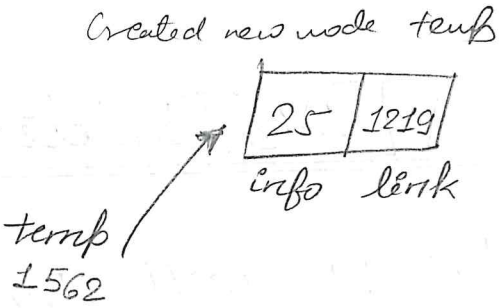

```
void display (struct node *head)
{
   struct node *P;
   if (head == NULL)
   {
      printf ("List is empty\n");
      return;
   }
   P = head;
   Printf ("List is :\n");
   While (P != NULL)
   { printf ("%d", P -> info);
      P = P -> link;
   }
   printf ("\n\n");
}
```

[19.3] Insertion

How to insert:

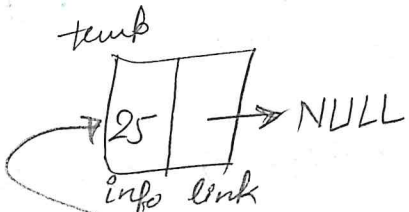
```
int main()
{
struct node *head = NULL;
int choice, data, item, pos;
while (1)
{
 Printf ("\n 1. Insert new node at begining \n");
 Printf ("\n 2. Insert new node at end \n");
 Printf ("\n 3. Insert new node after given \n");
 Printf ("\n 4. Insert given node at given position \n");
Printf ("\n 5. Quit \n\n");

 Printf ("\n enter your choice:");
 Scanf ("\n %d" &choice);
 Switch (choice)
 {
 Case 1:
```

```
Struct node * inset_at_beg (struct node *head, int data)
{
  struct node *temp;
  temp =(Struct node *) malloc (sizeof (struct node));
  temp -> info = data;
  temp -> link = head;
  head = temp; // head pointing to temp
  return head;
}
```



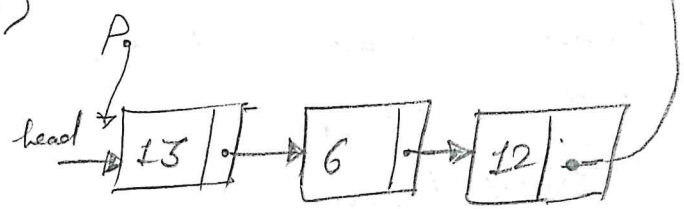
O(1)

```
Struct node * insert_at_end (struct node *head, int data
{
  Struct node *P, *temp;
temp = (struct node *) malloc (sizeof (struct node));
temp -> info = data;
temp -> link = NULL;
 P = head;
if (P)
{
while (P -> link != NULL)
      P = P -> link;
    P -> link = temp;
}
else (when P is NULL)
    head = temp;
    return head;
}
```



O(n)

P -> link = NULL



p -> link = temp

```
tail → link = temp
tail = temp
```

⇒ If I have tail pointer then time complexity for insertion_at_end will be O(1) only

[19.4] Delete

⇒ Here, we will try to know how to delete a node from singly linked list.

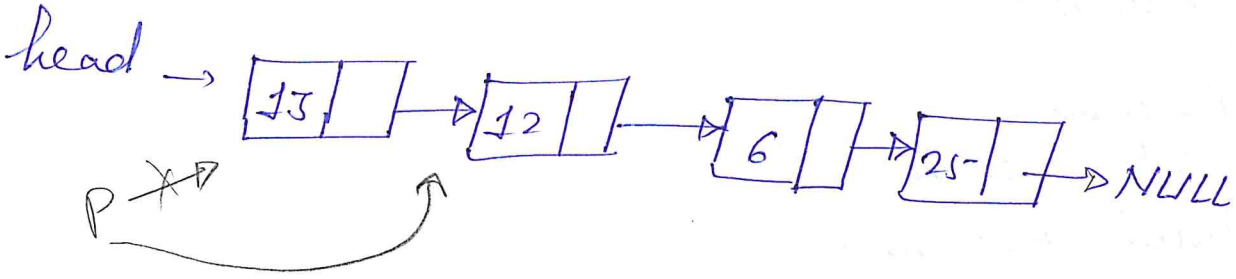
→ Deletion at beginning.
→ Deletion at last.

```
Struct node * del (Struct node *head, int data)
{
 Struct node *temp, *p;
 if (head == NULL)
 {
  printf ("List is empty \n");
   return head;
 }
 if (head → info == data)
 {
    temp = head;
    head = head → link;
    free (temp);
    return head;
 }
```

// Deletion in between or at the end.

```
P = head;
While (P -> link != NULL)   Start with head & goes to last element
{
  if (P -> link -> info == data)
  {
    temp = P -> link;
    P -> link = temp -> link;
    free (temp);
    return head;
  }
  P = P -> link;
}
Printf ("Element %d not found \n", data);
return head;
```

O(n)



=> If there are no duplicate elements, worst case time complexity will be O(n). because we are traversing list atleast half.

=> If duplicate elements. WC Time complexity O(n) because we have to check each and every element

[9.5] Traversal & Search

Traversal : by using head link going through each of node till end

Traverse the list or display every element in the list

```
void display (struct node *head)
{
  struct node *P;
  if (head == NULL)
  {
    Printf ("List is empty\n");
    return;
  }
  P = head;
  Printf ("List is: \n");
  While (P != NULL)
  {
    Printf ("%d", P->info);
    P = P->link;
  }
  Printf ("\n\n");
}
```

no o

accessing in order

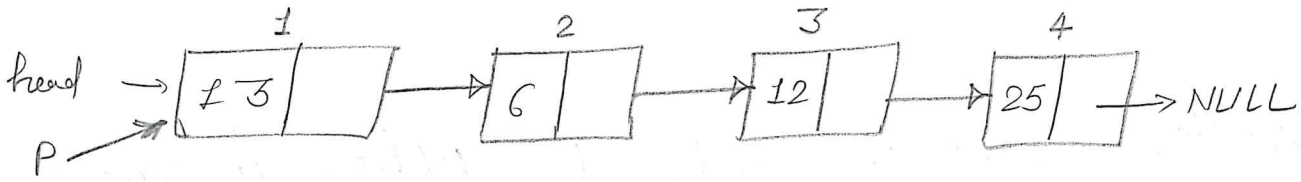
13 6 12 25

count no. of elements in the list.

```
void Count (struct node *head)
{
  struct node *P;
  int cnt = 0;
  P = head;
  While( P != NULL)
  {
    P = P->link; cnt++;
  }
  Printf ("No. of elements are %d\n", cnt);
```

Search : trying to get a particular item in list

```
Void search (struct node *head, int item)
{
  struct node *P = head;
  int pos = 1;
  while (P != NULL)
  {
    if (P->info == item)
    {
      Printf ("Item %d found at position %d\n", item, pos);
      return;
    }
    P = P->link;
    Pos++;
  }
  Printf ("Item %d not found in list\n", item);
}
```

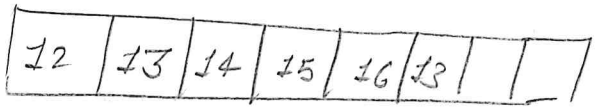


Time complexity = O(n)

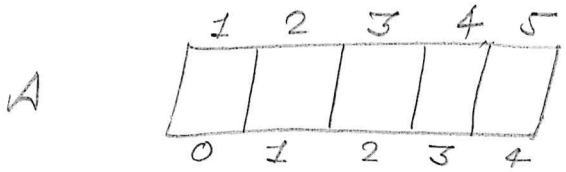
[9.6] Drawbacks

⇒ They use more memory than arrays because of the storage used by the pointers for address of next node.

⇒ Nodes are stored incontiguously, greatly increasing the time periods required to access individual elements with in the list, especially with a CPU cache.



⇒ Nodes in a linked-list must be read in order from beginning as linked lists are inherently sequential access.



A[3] can get element by using indexing

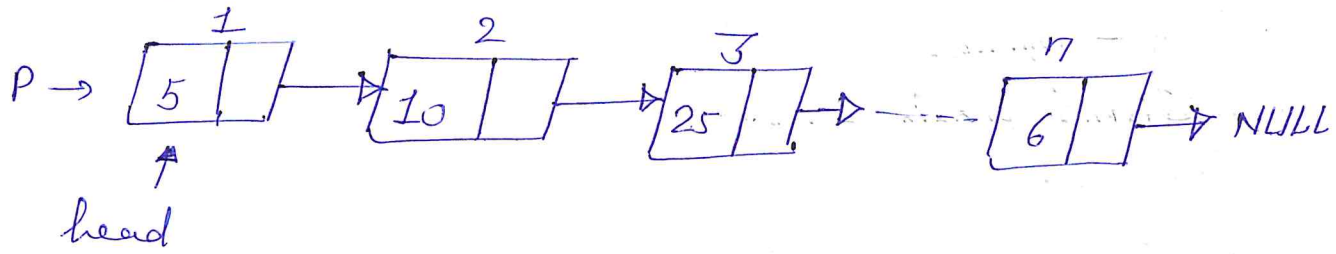
⇒ Difficulties arises in linked-list when it comes to reverse traversing

[9.7] Solved Problem GATE 2002

Ques: In the worst case, number of comparisons needed to search a singly linked list of length n for a given element is

(A) $\log n$ (B) $n/2$ (C) $\log n - 1$ (D) n

Solution:



If I need to search 25

1st compare to 1st node

compare to 2nd node

.

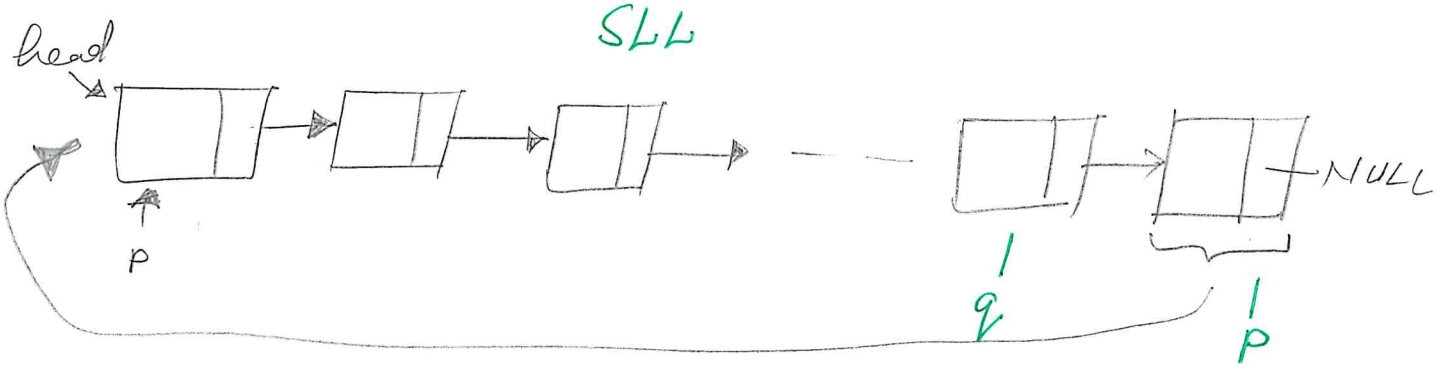
.

So in worst case, there will be n comparison.

[9.8] Solved Problem GATE 2010

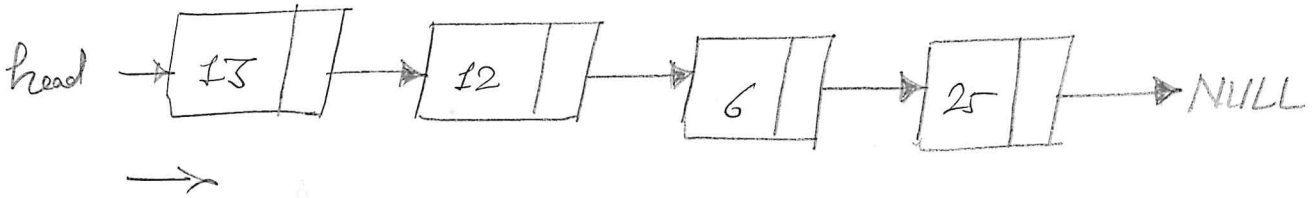
Ques: The following C function takes a simply linked list as input argument. It modifies the list by moving the last element to the front of the list and returns the modified list. Some part of the code is left blank.

```
typedef struct node
{
    int value;
    struct node *next;
} Node;

Node *move to front (Node *head)
{
    Node *p, *q;
    if (head == NULL : || (head -> next == NULL))
        return head;
    q = NULL;  p = head;
    while (p -> next != NULL)
    {
        q = p;
        p = p -> next;
    }
        q -> next = NULL;  p -> next = head;  head = p;
    return head;
}
```



Chapter ⇒ 5 Doubly Linked List

[20.1] Motivation of DLL

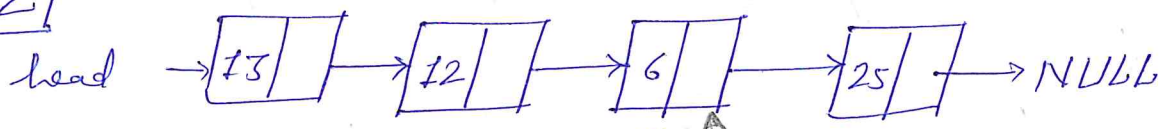


① Traverse this SLL in reverse order.

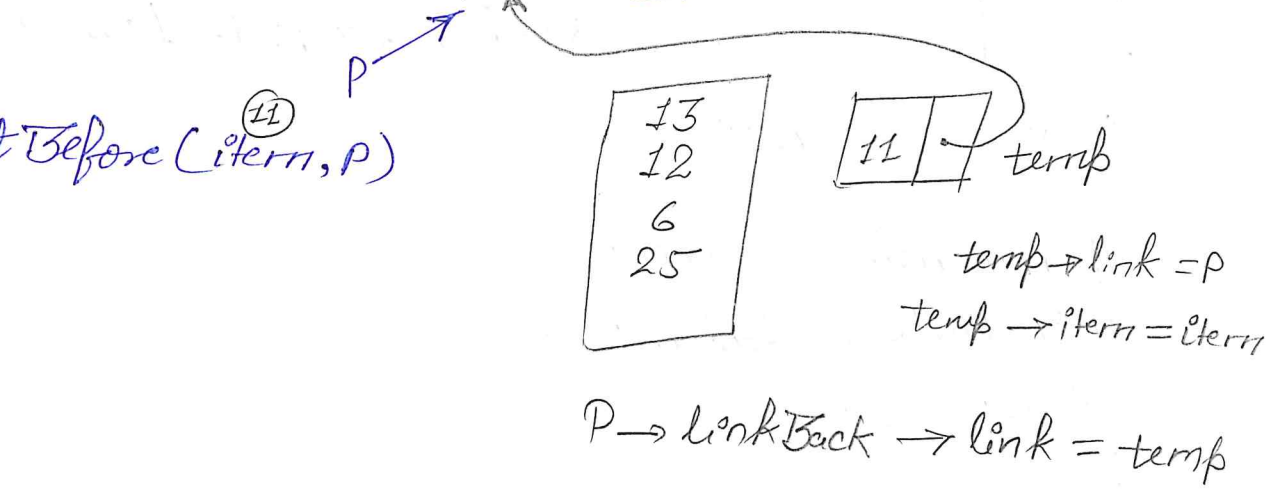
④, ③, ②, ①

② ⇒ Didn't come back

2



O(n)

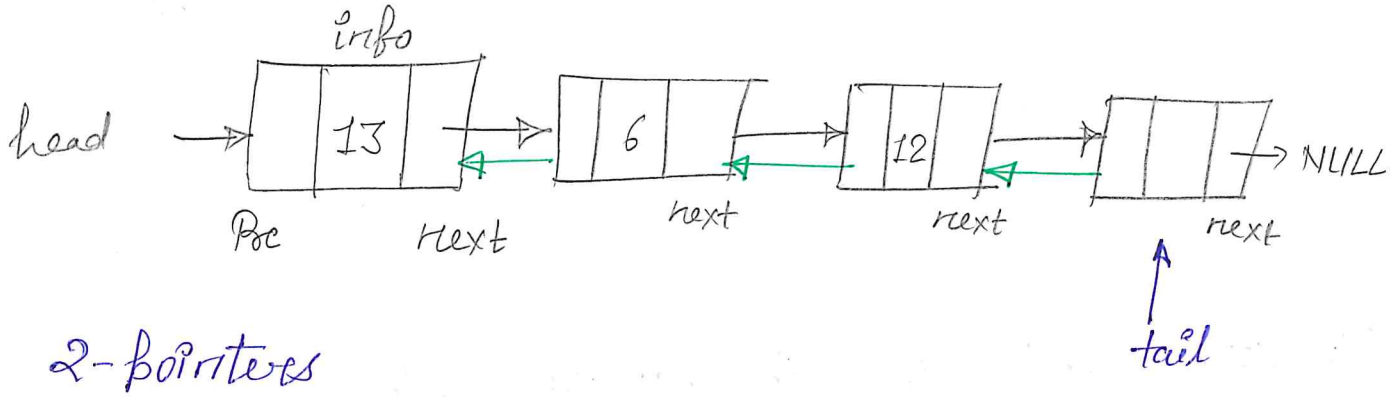


{

}

20.2 Structure and memory organization

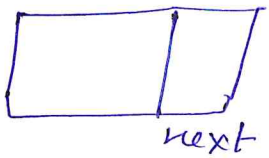
Doubly linked list :



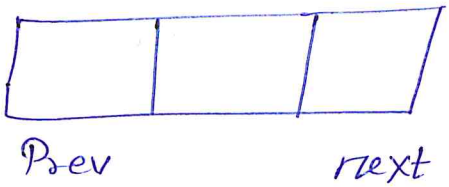
2-pointers

P= head tail

⇒ This takes more memory



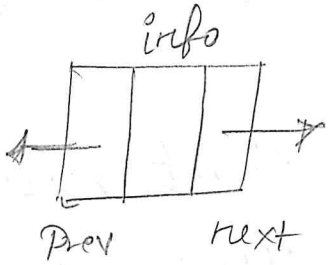
: singly linked-list (one pointer)



: Doubly linked-list (2-pointer)

```
#include <stdio.h>
#include <stdlib.h>
Struct node
{
 struct node *Prev;
  int info;
  Struct node *next;
};
void display (struct node *head)
{
  Struct node *P;
```

```
if(head == NULL)
{
  Printf("List is empty\n");
  return;
}
P = head;
Printf("List is:\t");
while(P != NULL)
{
  Printf("%d", P->info);
  P = P->next;
}
Printf("\n\n");
}
```

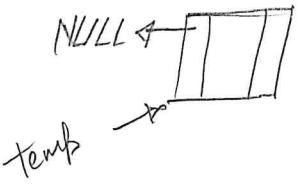


[20.3] Insert

O(1)

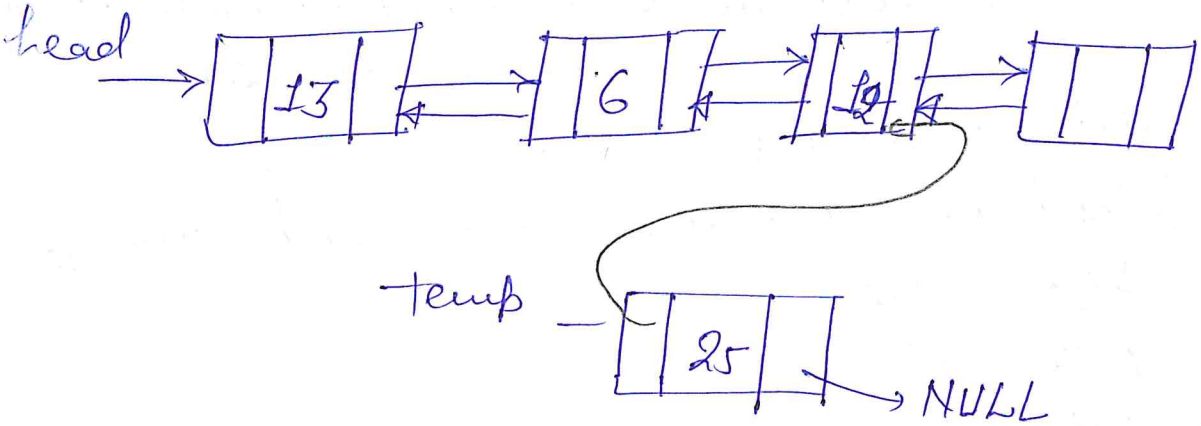
Insertion at beginning:

```
Struct node *insert_at_beginning(Struct node *head,
                                 int data)
{
 Struct node *temp;
temp = (Struct node *) malloc (sizeof (Struct node));
temp -> info = data;
temp -> prev = NULL;
temp -> next = head;   // Started pointing head of list
if (head)
    head -> prev = temp;
    head = temp;
    return head;
}
```



```
Struct node * insert_at_end (Struct node * head,
                             int data)
{
    Struct node *temp, *P;
    temp=(Struct node *)malloc (sizeof (Struct node));
    temp -> info = data;
    P= head;
    if (P)
    {
        while (P -> next != NULL)
            P = P -> next;
            P -> next = temp;
            temp -> prev = P;
    }
    else
        head = temp;
        return head;
}
```

O(n)



Insert after a given position: O(n)

```
Struct node *insert_after_given_node(Struct node *head,
                                     int data, int item)
{
 Struct node *temp, *P;
 temp = (Struct node *) malloc (sizeof (Struct node));
 temp -> info = data;
 P = head;
 While (P != NULL)
 {
  if (P -> info == item)
  {
   temp -> prev = P;
   temp -> next = P -> next;
   if (P -> next != NULL)
      P -> next -> prev = temp;
      P -> next = temp;
      return head;
  }
   P = P -> next;
 }
 Printf ("%d not present in the list \n\n", item);
 return head;
}
```

P = 1000

1000 != NULL

15 == 6 ✗ false 1002 ✓ 6 ✓

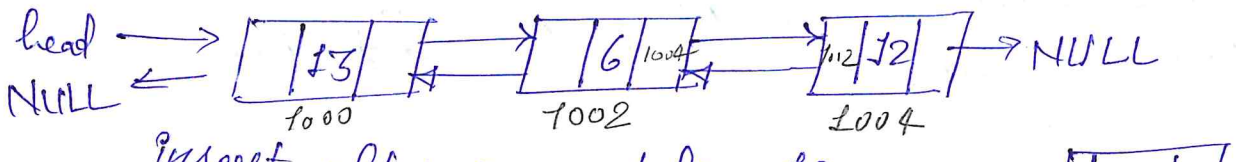
temp -> prev = ~~2000~~ 1002

temp -> next = 1004 ✓ 1004

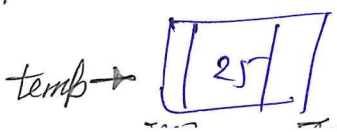
P -> next -> prev = ~~1002~~ 2000

P

P = ~~1000~~ 1002



Insert after 6 or before 12



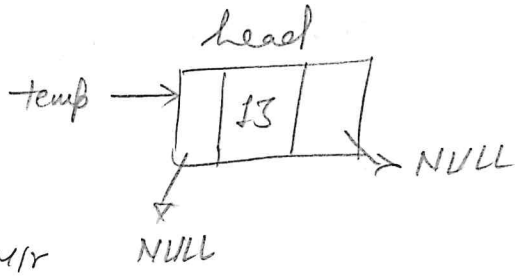
[20.4] Delete

⇒ Deleting node from start
from end
from in-between.

Deleting node from the list.

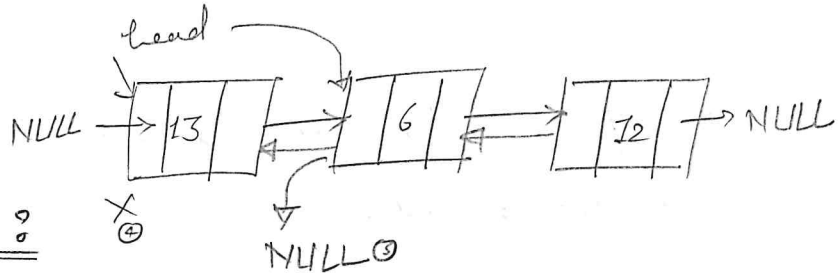
```
Struct node *del (Struct node *head, int data)
{
 Struct node * temp ; // creating temporary pointer
 if(head == NULL)
 {
  Printf ("List is empty \n");
  return head ;
 }
 if(head -> next == NULL)  // only one node in the list
 {
  if (head -> info == data)
  {
     temp = head ;
     head = NULL;
     free (temp); // free u/r
     return head;
  }
  else // If data is not there
  {
  Printf ("Element %d not found \n", data);
  return head;
  }
}
```

Deleting a node which has some data



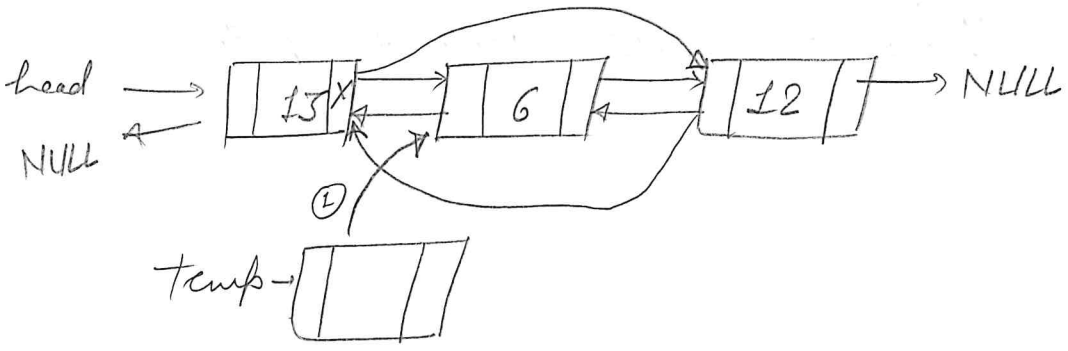
Deletion of first node

```
if (head -> info == data)
{
    temp = head;
    head = head -> next;
    head -> prev = NULL;
    free (temp);
    return head;
}
```



Deletion in between :

```
① temp = head -> next;
while (temp -> next != NULL)
{
    if (temp -> info == data)
    {
②       temp -> prev -> next = temp -> next;
        temp -> next -> prev = temp -> prev;
        free (temp);    // Deleting node of Data 6
        return head;
    }
    temp = temp -> next;
}
```



Deletion of last node:

```
If (temp -> info == data)
{
 temp -> prev -> next = NULL;
  free temp;
  return head;
}
Printf ("Element %d not found\n", data);
return head;
}
```

[20.5] Drawbacks

Doubly linked list has two major drawbacks:

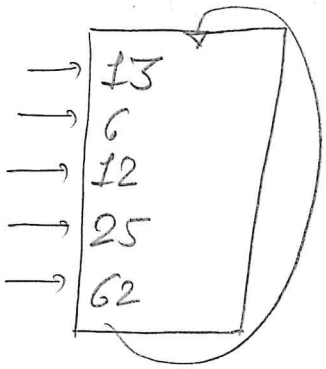
(1) It occupies more space or more memory. That's because here we have two pointers.

DLL → 2 ptr , SLL → 1 ptr

(2) You have to adjust, modify or edit more no. of pointers, when you insert / Delete a node in the list. So there are more no. of pointer operations.

Chapter ⇒ 4 Circular Linked List

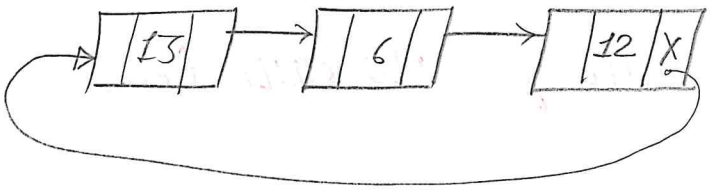
[21.1] Motivation



Playlist → Music → Linked_list

After last song, I want to come back to first song

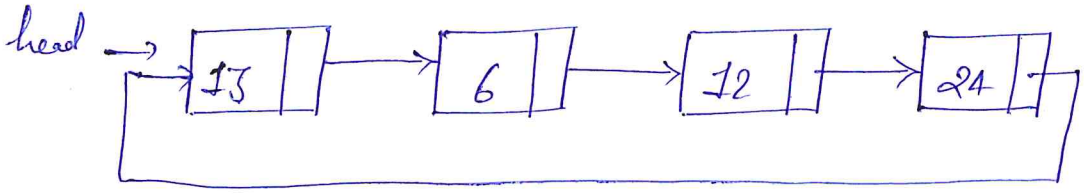
For that, circular linked list is best



Instead of pointing to NULL what if last node next pointer points to first node.

⇒ Circular linked-lists are used in the kernal of operating system (Mac, win, Linux) Android)

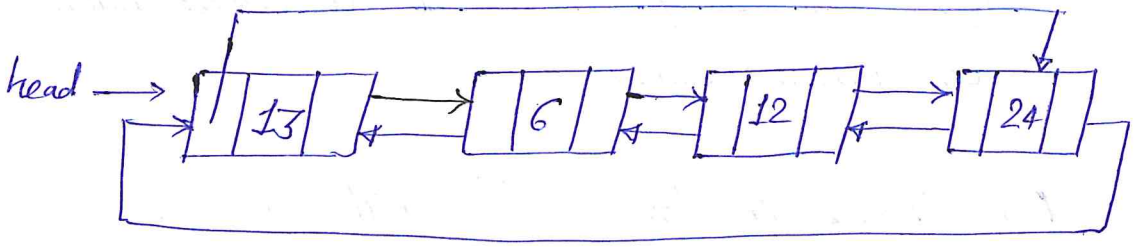
[20.2] Singly and Doubly Circular Linked List



① When you remove or add at first place of list, you not only adjust first node next, you need to take care of next of last node.

(2) When you are traversing each of element as (P==head) you again reach to the start node because there is no NULL at last next.

C-DLL



(P==head) P=NULL ✗ NO NULL here

[20.3] code

SLL & DLL

CLL

code for circular linked list is attached in the description. Please refer that link.

[20.4] Solved Problem GATE 2016

Ques: N items are stored in a sorted doubly linked list. For a delete operation, a pointer is provided to the record to be deleted. For a decrease-key operation, a pointer is provided to the record on which the operation is to be performed. An algorithm performs the following operations on the list in this order:

$O(N)$ delete, $O(\log N)$ insert, $O(\log N)$ find, and $O(N)$ decrease-key

What is the time complexity of all these operations put together.

Solution:

DLL → forward, backward

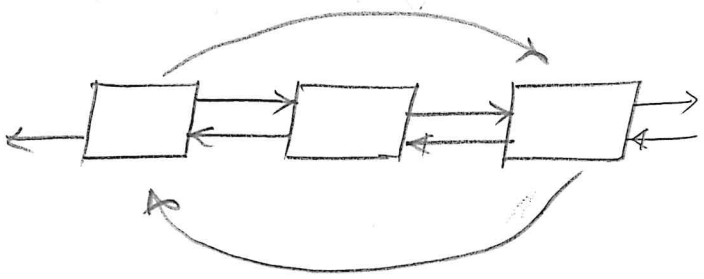
sorted →

delete-op → $O(1) * O(N) = O(N)$

decrease-op → $O(N) * O(N) = O(N^2)$

insert-op → $O(N) * O(\log N) = O(N \log N)$

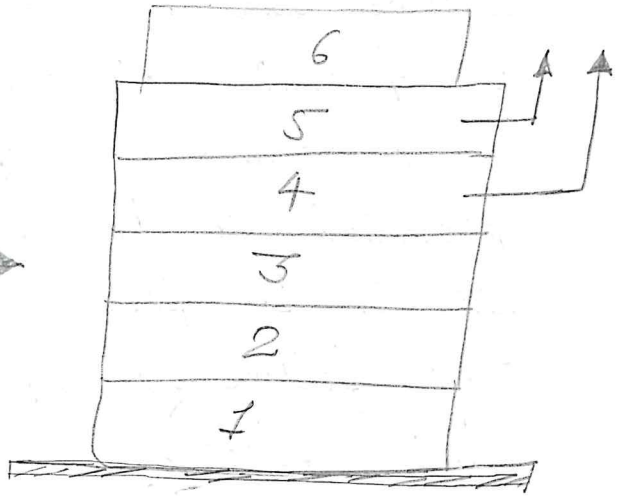
find-op → $O(N) * O(\log N) = O(N \log N)$



$O(N^2)$

Chapter ⇒ 5 Stacks

⇒ Application of books



Stack of books →

If we remove then sth book is removed

Last inserted book = 6
First deleted book = 6

LIFO
Last in first out

Why ?
Where all stacks are used ?
how

expression evaluation :

$x = 23 + 2 * 6/4 - 2$

divide by 4 (higher precedence)

BODMAS

↓
Stack

Parenthesis check : [for every open parenthis, there is closing parenthis]

$(21+(2+3/2)+(4/6)-(2+(3/2)))$

⇒ Imagine you are writing a recursive function factorial, fiboracci, rmerge sort, quick sort.

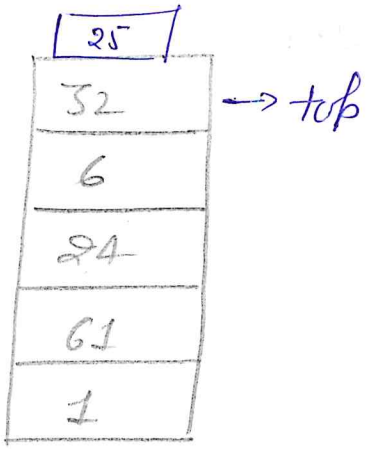
how does compiler work for recursion

Compiler, execution } using stack

[22.2] operations : Push & Pop

⇒ ① collection of items on which operations are done.

DS ② operations
③ relationships



collection of books on relation as one book is lying on top of other

2- Mejors operations in stack : Push, Pop.

Push (S, 25) = It will be stored on top of 32
POP(S) : item at top is removed.

x = POP(S)
x = 25

now top = 32

Last item which went into stack, will be first item which is removed from stack.

Why we have two items (push, Pop) only

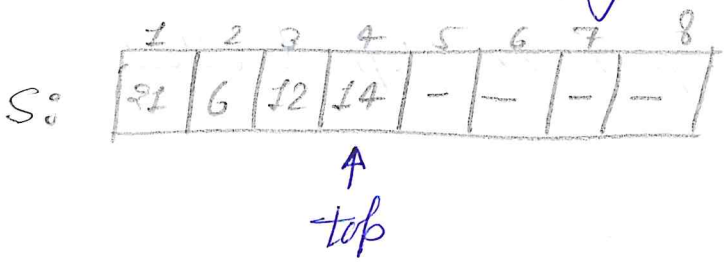
Because by using these two items, we can perform various major operation.

→ Parenthesis check
→ Expression evaluation

[22.3] How to implemented a stack

There are two ways of using stack

→ Array
→ using linked-list



Push (S, top, a) a=10 Push = O(1)

1: increment top = top = top+1

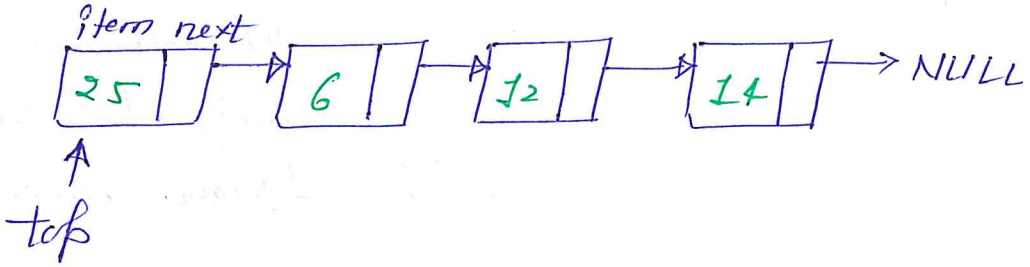
2: S[top] = a // inserted top

If array is full & we are inserting.

Pop (S)

1. temp = S[top];
2. top = top - 1
3. return temp;

using singly linked-list:



Push (top, 8) ⇓ O(1)

1: Create a temp node with value 8
2: temp → next = top
3: top = temp

There is no limit, you can add as long memory space is not full.

POP (top) ⇓

- If stack is empty, top should point NULL.

1. temp = top
2. top = top → next
3. x = temp → info
4. free (temp);
5. return x;

top → NULL

O(1)

[22.4] Application : Parenthesis check

Stack : Push & Pop

{ (2+3) * {4+6} / 2+3 }

Three types of parenthesis we use

→ ( ) Simple bracket
→ { } curly braces
→ [ ] Square brackets

for array indexing A[10] = 25

for if condition if(---)

curly braces : fun () {

}

int f( ){

}

} // Wrong

} error don't have opening brace

Balanced : for every open brace, there is a close brace.

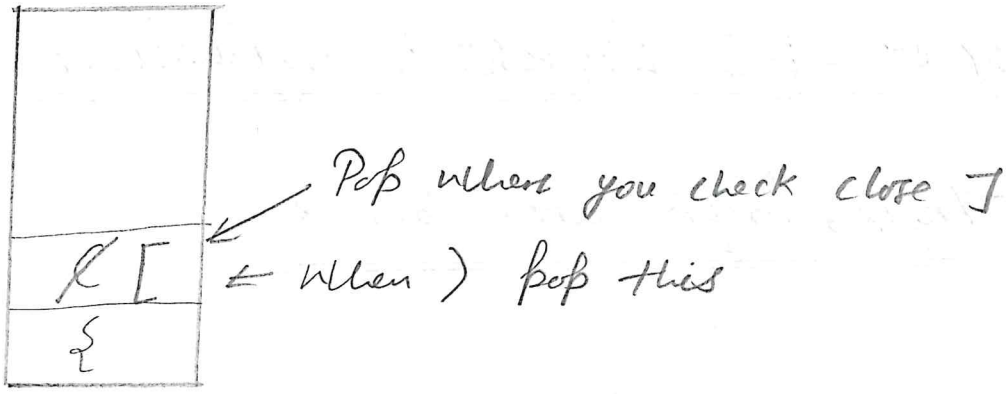
{ ( ) [ ] }

{ ( ) [} ✗ imbalanced parenthesis

use of stack for balancing parenthesis check

① { 23 + 4 * (4+6) + [4+6] }

Parenthesis_check (str) // will return 1 if balanced else error



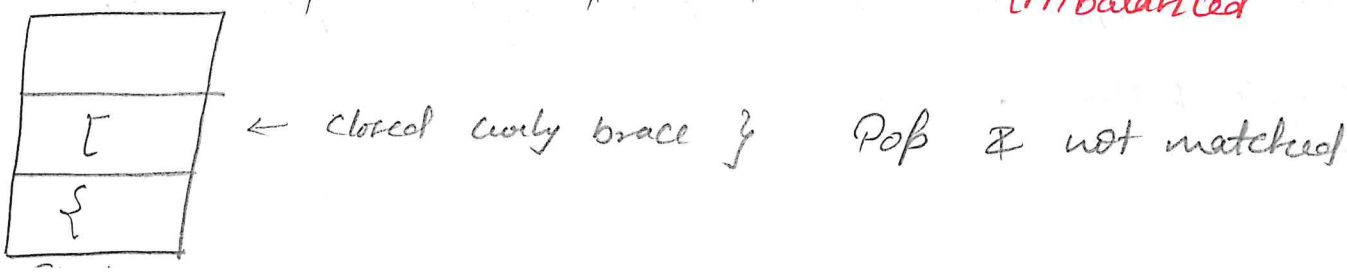
① Push start brace into the stack

② When you see close brace, just pop its open brace from stack

③ no braces then ignore other things.

Ex:2 str = { 21+3 * [4 * 2 }]

imbalanced



{ C [ ] ✗

{ C [ ) ✗

⇒ Parenthesis check is used not only for arithmetic expression, for executing codes by compiler.

Chapter ⇒ 6 Expression Evaluation

[23.1] Infix, prefix and postfix

A + B * C operators: +, * 2 parameter

operands: A, B, C

Infix: in this, operator is placed between operands. sin(x)

A + B

Prefix: operators is placed before operands.

Polish notation + A B

Postfix: operators are placed after operands.

reverse polish notations A B +

⇒ While executing expression, we need to take care of precedence of operators

BODMAS rule

A + B * C

↙ ↘

A + (B * C) ((A + B) * C)

} Parenthesis are helpful for getting order of evaluation.

Prefix:

+ A * B C

⇓

(A + (B * C))

We don't need parenthesis for 2-parameter prefix expression.

* + A B C ⇒ (A + B) * C

Precedence is inherent.

Postfix notation:

A B C * + ⇒ ((B * C) + A)

Infix —BODMAS→ Parenthesize infix —Stack→ evaluation

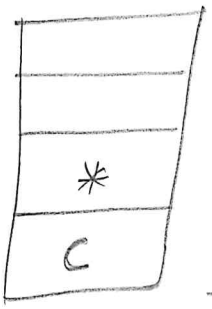
Prefix → evaluation

Postfix → evaluation

[23.2] Infix to postfix

```
#include <stdio.h>
 Char stack[20];
 int top = -1;

void push (Char x)
{
  stack[++top] = x;
}
Char pop ( )
{
 if (top == -1)
   return -1;
 else
   return stack[top--];
}
int priority (Char x)
{
  if (x == '(')
     return 0;
 if (x == '+' || x == '-')
     return 1;

if (x == '*' || x == '/')
     return 2;
}
```

(2* 3+4 * (5-6))



stack

2 x 3 + 4 * (5 - 6))

```
int main()
{
 char exp[20];
 char *e, x;
 Printf("enter the expression::");
 Scanf("%s", exp);
   e = exp;
while(*e != '\0')
 {
   if(isalnum(*e))
     Printf("%c", *e);
 elseif (*e == '(',
         Push(*e);
elseif (*e == ')'){
   while((x = Pop()) != '(')
     Printf("%c", x );
 }
 else {
   while (priority (stack[top]) >= priority (*e))
   Printf("%c", Pop());
    Push(*e);
 }
  e++;
 }
   while (top != -1)
   {
    printf("%c", Pop());
 }
```

[20.3] Infix to prefix

→ Infix to postfix O(n)

① reverse the expression = rev_exp
O(n) (→)) → (

Ex:1 (2 + (3 * 4) * 6)

↓ reverse

open → close
close → open

(6 * (4 * 3) + 2)

② reverse expression —stack→ postfix = Post exp
O(n)

(6 * (4 * 3) + 2)

↓

4 3 *

6 4 3 * * 2 +

③ reverse post_exp ⟶ prefix to exp
O(n)

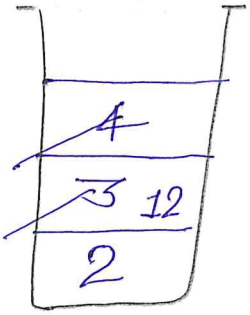
6 4 3 * * 2 + ⟶ + 2 * * 3 4 6

So total time complexity = O(n)

[23.4] Evaluation of postfix

2 3 4 * 6 * +

1. read each element (L to R)
2. if it is an operand
   Push it into stack

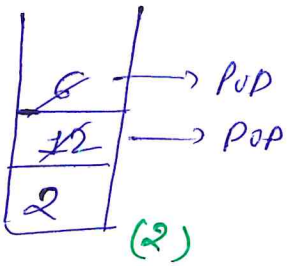


(1)

3. if it is an operat
   → pop the top 2 element
   → apply the operator on popped elements
   → push the result on to the stack

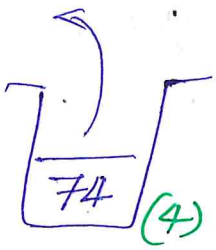
get * 4*3 = 12

4. Pop = result

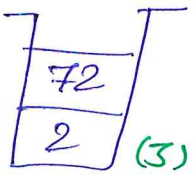


(2)

get * = 6*12 = 72



(4)



(3)

get +
= 72+2
= 74

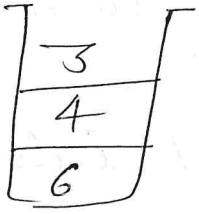
2 3 4 * 6 * +

(2+((3*4)*6)) = 2+12×6 = 74

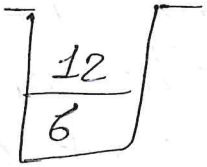
⇒ In postfix expression, we always have operator after operands. 3 4 *

[23.5] Evaluation of prefix

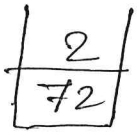
(1) read the expression from Right to Left +2**346



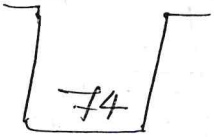
when get * = 3*4=12



get * = 12*6 = 72



get + = 2+72 = 74



Chapter ⇒ 7 More Application : call stack

24.1 Call-Stack — runtime Stack or machine Stack

```
factorial (n)
  if n=1
     return 1
  else
     return n * factorial (n-1)
```

$T(n) = c + T(n-1)$

use tree method

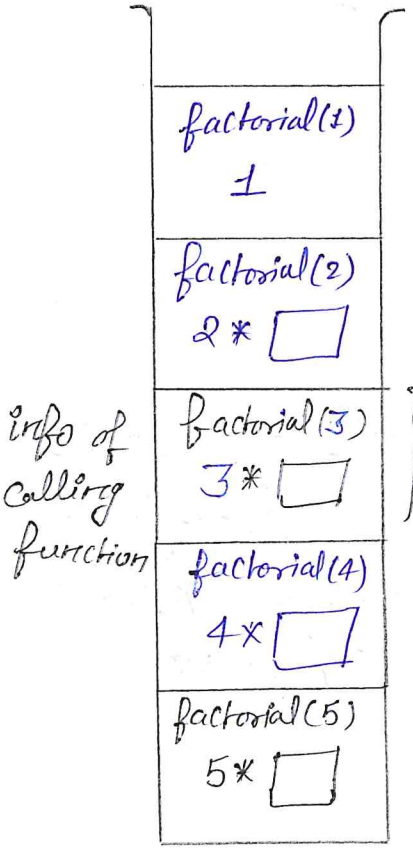
n
n-1
(n-1)-1
.
.
1

$TC = O(n)$
$SC = O(n)$

Ex:1

factorial (5)
⇓ 120
5 x factorial (4) = 5x24 = 120
⇓ 24
4 x factorial (3) = 4x6 = 24
⇓ 6
3 x factorial (2) = 3x2 = 6
⇓ 2
2 x factorial (1) = 2x1 = 2
⇓
1

Runtime Stack



⇒ Call stack is used everywhere, when we design compiler.

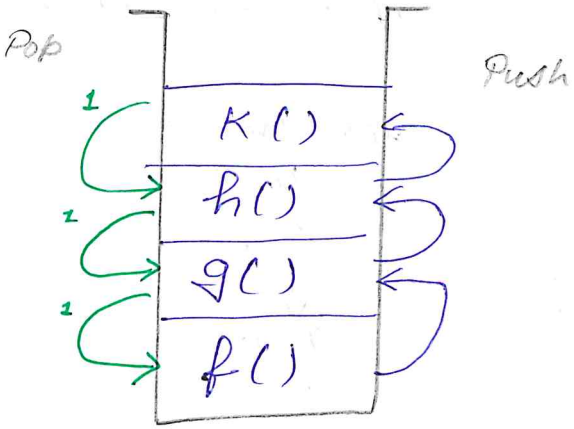
← n-frames in Call Stack
$S(n) = O(n)$

f()
g()

g()
h()

h()
k()

k()
{ return 1 }



Chapter ⇒ 8 Solved problems of Stacks

26.1 GATE 2004-1

The best data structure to check whether an arithmetic expression has balanced parenthesis is a (A) queue (B) stack (C) tree (D) list

Solution: Stack is best data structure

use Push - Pop

26.2 GATE 2004-2

Assume that the operators +, −, × are left associative and ^ is right associative. The order of precedence (from highest to lower) is ^, ×, +, −. The postfix expression corresponding to the infix expression a+b×c−d^e^f is

(A) abc x + def ^ ^ -
(B) abc x + de^f ^ -
(C) ab + c x d - e^f^
(d) - + a x bc^^def

Solution:

infix

a+bxC-d^e^f

(a + (b*c)) - (d^(e^f))

abc* + def^^ -

abc* + def^^ -

left:

2+3+4 = ((2+3)+4)

Right:

2+3+4 = (2+(3+4))

[26.3] GATE 2007

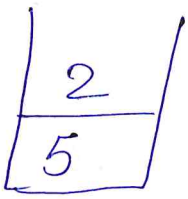
```
#include
# define EOF -1
void push (int);
int POP (void);
void flagError();
Int main()
{
   int c, m, n, r;
while ((c=getchar()) != EOF)
{ if (isdigit (c))
        Push(c)
else if ((c == '+') || (c == '*')
  {
      m = Pop();
      n = Pop();
      r = (c == '+') ? n+m : n*m;
      Push(r);
  }
else if (c != ' ')
    flagError();
}
Printf ("%c", Pop());
}
```

What is output of the program for the following input.?

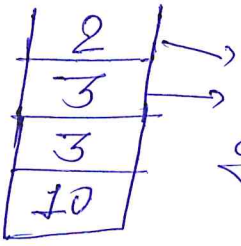
5 2 * 3 3 2 + * +

(A) 15 (B) 25 (C) 30 (D) 150

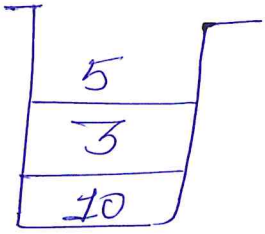
Solution:



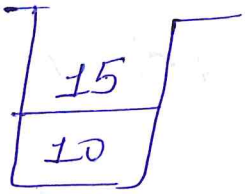
get *



get + = 2+3
= 5



get *
3x5=15



get + = 10+15
= 25

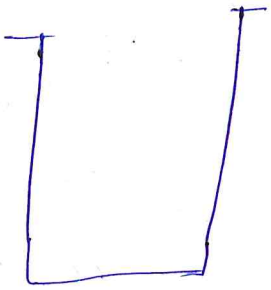
[26.4] GATE 1997

Which of the following is essential for converting an infix expression to the postfix form efficiently.

(a) An operator stack (b) An operand stack
(c) An operand stack and an operator stack
(d) A parse tree.

Solution:

Infix to postfix expression



Stack

[26.5] GATE 2005

A function f is defined on stacks of integers satisfies the following property

$f(\phi) = 0$ and $f(Push(S,i)) = \max(f(S),0)+i$

for all stacks S & integers i.

If a Stack S contains the integers 2, -3, 2, -1, 2 in order from bottom to top. What is f(S).

(A) 6 (B) 4 (C) 3 (D) 2

Solution: 2, -3, 2, -1, 2

f(S)	i
0	2
2	-3
-1	2
2	-1
1	2

$f(Push(S,i)) = \max(f(S),0)+i$

$\max(0,0)+0$

$= 0$

3

[26.6] GATE 1991

Following sequence of operations is performed on Stack

Push(10) Push(20) Pop Push(10) Push(20) Pop, Pop, Pop Push(20), Pop

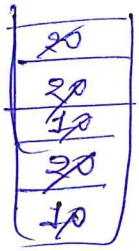
Sequence of popped elements.

(a) 20, 10, 20, 10, 20

(b) 20, 20, 10, 10, 20

(c) 10, 20, 20, 10, 20

(d) 20, 20, 10, 20, 10



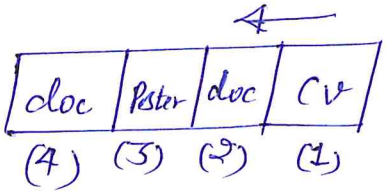
→ 20, 20, 10, 10, 20

Chapter ⇒ 9 Queue

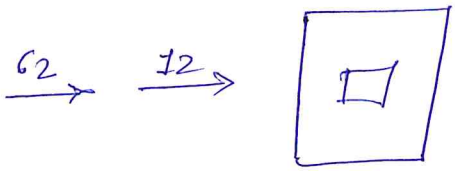
[28.1] Motivation: Why we need them?

Queue → first-in-first-out

Why?

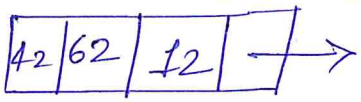


Reservation bus / train / flight



first person who requested will get seat first

Queue (FIFO)



OS

Processes / Programs

2- primary operations on Queue:

operations → enqueue
operations → Dequeue

FIFO organization

Collection

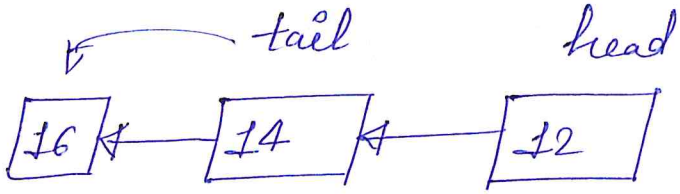
[28.2] operations : Enqueue and Dequeue

Queue (FIFO) : enqueue , Dequeue

Array Linked-List

head and tail

When queue is empty head = NULL
tail = NULL

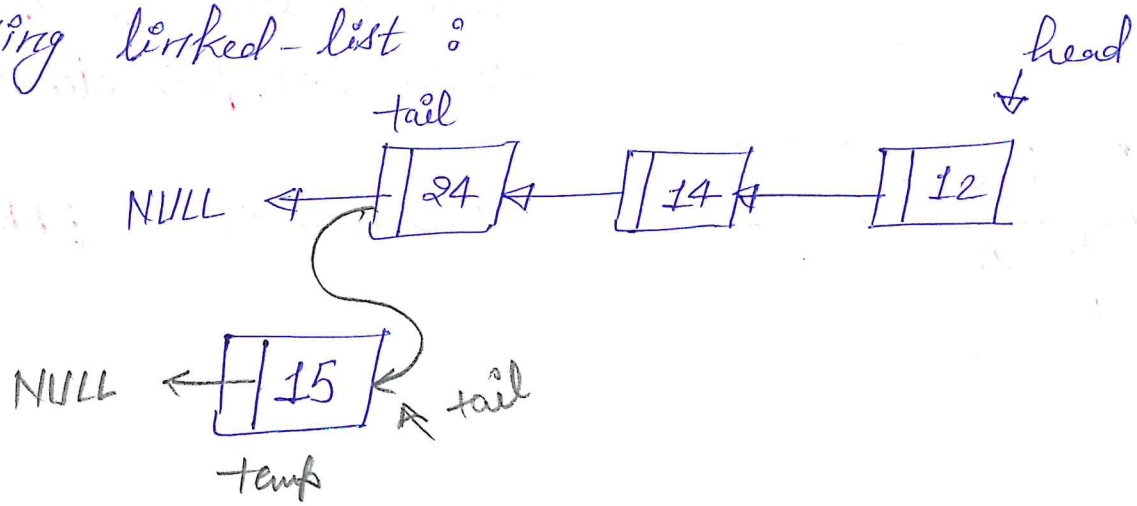


When you are adding elements in queue, only tail pointer will move not head

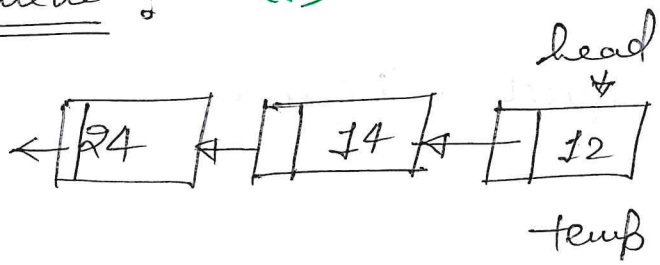
When enqueuing → tail (insertion from here)
When dequeuing → head (Deletion from here)

[28.3] how to implement them

using linked-list :



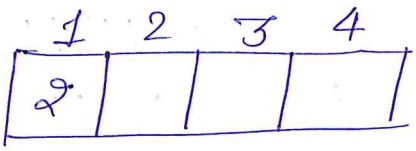
Dequeue : O(1)



first temp starts pointing head

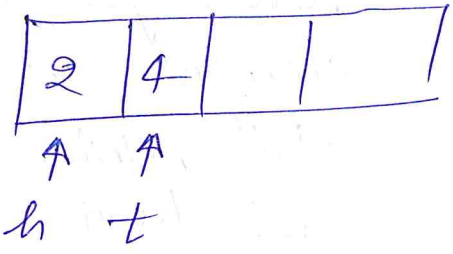
Array implementation

O(1) O(1)



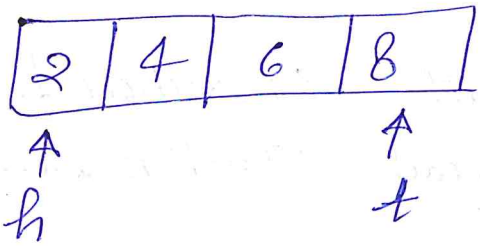
Can not hold more than 4-elements.

head → tail →



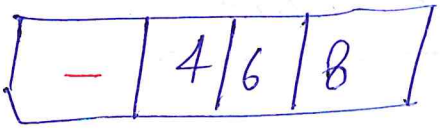
4 inserted

enqueue (2)
enqueue (4)
enqueue (6)
enqueue (8)

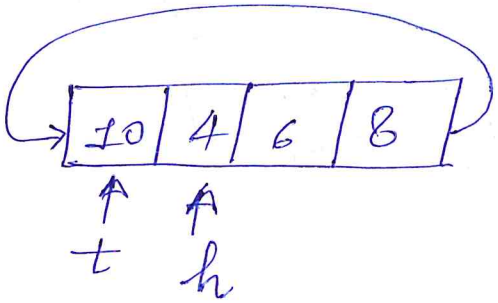


Dequeue → 2

enqueue (10)

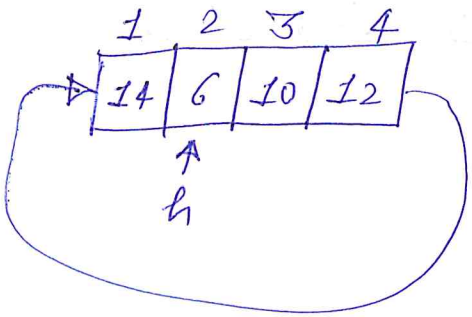


even if space is there We can not put element



enqueue (12) — ERROR

[28.4] Linear vs Circular Queue Implementation



enqueue (14)

Circular array implementation

↓

upto n element

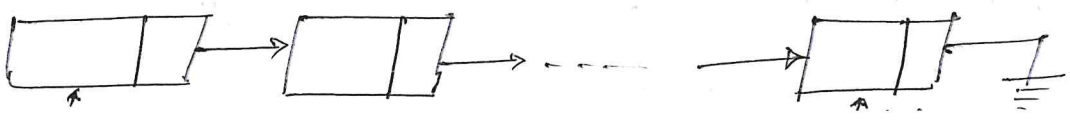
Linear array → Worse

Can not store n-elements enqueue (14) ✗

⇒ We can not elements fwether in Linear array queue Cause head pointer moved forward after deletion.

[28.5] Solved Problem GATE 2004

A queue is implemented using a non-circular singly linked list. The queue has a head pointer and a tail pointer, as shown in the figure. Let n denote the no. of nodes in the queue. Let enqueue be implemented by inserting a new node at the head, and dequeue be implemented by deletio of a node from the tail.



Which of the following is the time complexity of the most efficient implementation of 'enqueue' and 'dequeue', respectively for this data structure.

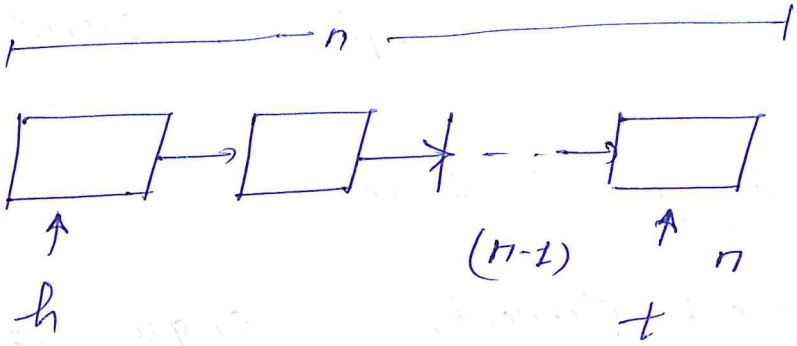
(A) $O(1)$, $O(1)$

(B) $O(1)$, $O(n)$

(C) $O(n)$, $O(1)$

(D) $O(n)$, $O(n)$

Solution :

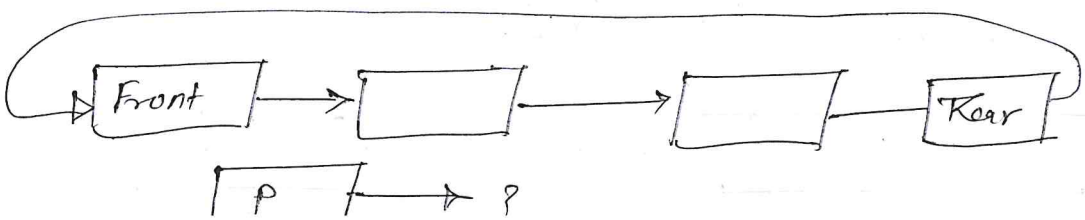


for enqueue $O(1)$

dequeue $O(n)$ We want next of second last so we can not get previous of it from tail point in SLL.

[28.6] Solved Problem GATE 2016

A circularly linked list is used to represent a queue. A single variable P is used to access the queue. To which node should P point such that both the operations enqueue and dequeue can be performed in constant time.



(A) rear node
(B) front node
(C) not possible with a single pointer
(D) node next to front

Solution:

When pointer at FRONT

add some at REAR : if n-element list

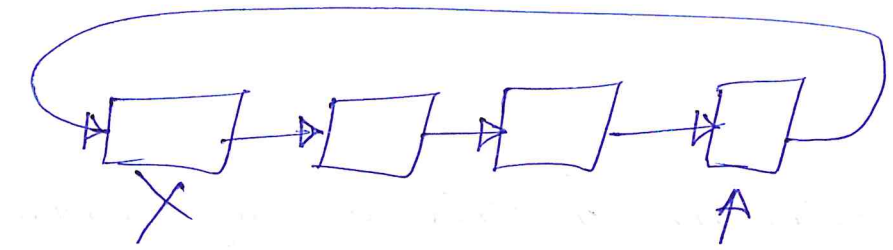
→ It will take O(n) time

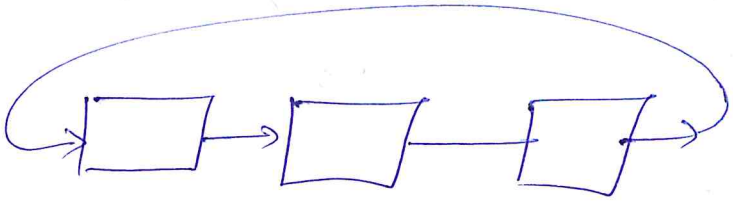
When pointer points at REAR

P → next = FRONT of queue

enqueue = O(1)

dequeue = O(1)





[28.7] Solved problem GATE 2013

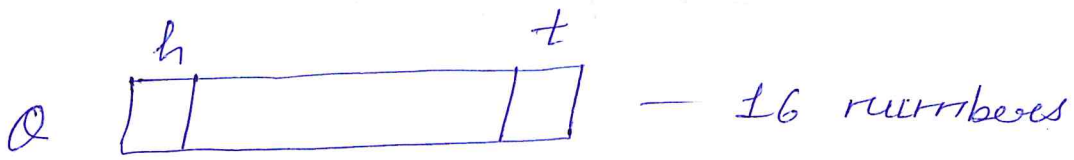
Let Q denote a queue containing 16 numbers and S be an empty stack. Head(Q) returns the element at the end of the queue Q without removing it from Q. Similarly Top(S) returns the element at the top of S without removing it from S. Consider the algorithm

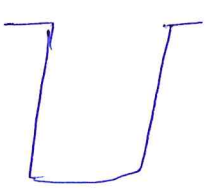
```
while Q is not empty do
if S is empty OR Top(S) ≤ Head(Q) then
        x = Dequeue(Q)
        Push(S, x);
else
        x = Pop(S);
        enqueue(Q, x);
end
end
```

The maximum possible no. of iteration of the while loop in the algorithm is

(A) 16 (B) 32 (C) 256 (D) 64

Solution:





Dequeue C

Push into Stack

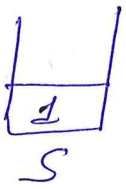
16, 15, ---- 1

15, 14 - - - - 1

(2) { 15, 14, --- 1, 16
14, 13 --- 1, 16, 15

1, 16, 15 ---- 2

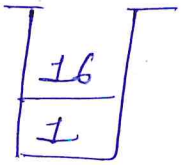
16, 15 --- 2



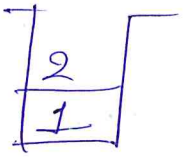
top(S) ≤ head(Q) always

30 + 1 Push(16) } 2 iteration
Pop(16)

15, 14 ---- 2



31, 29, 27 -



[28.8] Solved problem GATE 2016-2

Ques: Consider the following operation along with Enqueue and Dequeue operations on queues, where k is a global parameter.

```
MultiDequeue(Q)
{
    m = k
while ((Q is not empty) and (m > 0))
 {
    Dequeue(Q)
     m = m - 1
 }
}
```

What is the worst case time complexity of a sequence of n queue operations on an initially empty queue.

(A) $O(n)$ (B) $O(n+k)$ (C) $O(n*k)$ (D) $O(n^2)$

Solution:

Multiqueue is applying dequeue till the queue is not empty.

$O(n)$

less than equal to n elements

$\leq n$ operations

[28.8] Solved Problem GATE 2014

A queue is implemented using an array such that ENQUEUE and DEQUEUE operations are performed efficiently. Which one of the following is CORRECT (n refers to the no. of items in the queue)

(A) Both operations can be performed in O(1) time

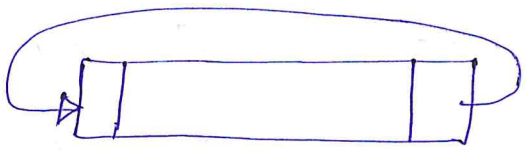
(B) At most one operation can be performed in O(1) time but the worst case time for the other operation will be $\Omega(n)$

(C) The worst case time complexity for both operation will be $\Omega(n)$

(D) Worst case time complexity for both operations will be $\Omega(\log n)$

Solution :

Circular implementation



[28.10] Solved Problem GATE 2018

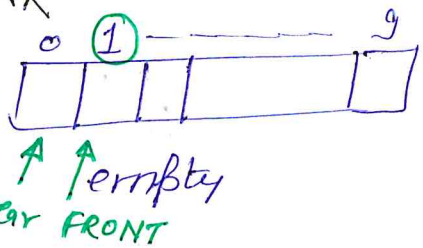
Suppose a circular queue of capacity $(n-1)$ elements is implemented with an array of n-elements. Assume that the insertion and deletion operations are carried out using REAR and FRONT as array index variables, respectively. Initially, REAR = FRONT = 0. The conditions to detect queue full and queue empty are:

(A) FULL: $(REAR+1) \bmod n == FRONT$
empty: $REAR == FRONT$

(B) FULL: $(REAR+1) \bmod n = FRONT$
empty: $(FRONT+1) \bmod n == REAR$

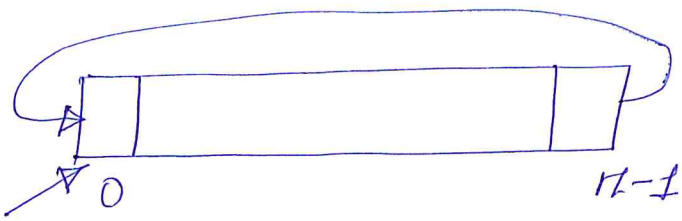
(C) FULL: $REAR == FRONT$
empty: $(REAR+1) \bmod n == FRONT$

(D) FULL: $(FRONT+1) \bmod n == REAR$
empty: $REAR == FRONT$



Full:

Solution:



rear = front = 0

mod - modulo

(rear + 1) mod 10

10 mod 10 = 0

[28.11] Solved Problem GATE 1996

Consider the following statements:

(i) FIFO types of computations are efficiently supported by Stack. LIFO

(ii) Implementing LISTS on linked-lists is more efficient than implementing LISTS on an array for almost all the basic LIST operations.

(iii) implementing Queue on a circular array is more efficient than implementing queues on a linear with two indices.

(iv) LIFO type of computations are efficiently supported by Queue. FIFO

Which are true: (A) (ii) & (iii) (B) (i) & (ii)
(C) (iii) & (iv) (D) (ii) & (iv)

[28.12] Solved Problem GATE 2007

Suppose you are given an implementation of a queue of integers. The operations that can be performed on the queue are:

(i) isEmpty(Q) — return true, if queue is empty else false

(ii) delete(Q) — deletes the elements at front of queue and returns its value.

(iii) insert(Q, i) — inserts the integer i at rear of queue.

Consider the following functions:

```
void f(queue Q) {
 int i;
if (! isEmpty (Q)) {
      i = delete (Q);
       f(Q);
      insert (Q, i);
    }
}
```

What operations is performed by the above function f?

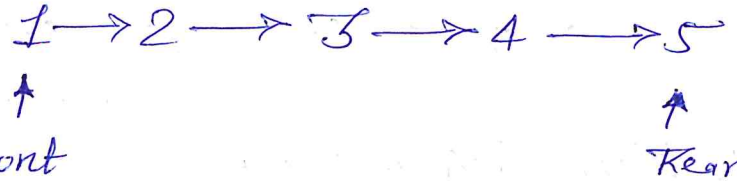
(A) Leaves the queue Q unchanged

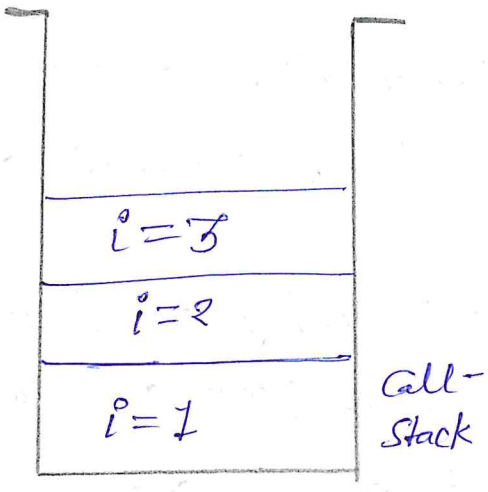
(B) Reverses the order of elements in the Q.

(C) Deletes the element at the front of queue Q and insert it at the rear keeping the other elements in the same order

(D) Empties the queue Q.

Solution:





Chapter ⇒ 10 Arrays as a data Structure

[29.1] one-dimensional array

Array: collection of items of similar type or data type

int a[10]; all 10 - elements of integer type

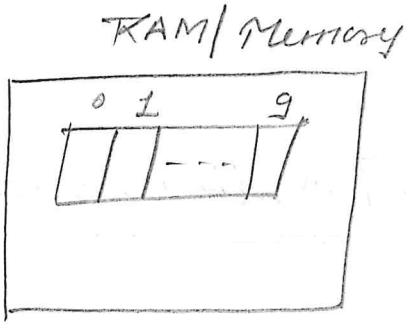
⇒ ordering among elements.

⇒ Modify a[5] = 2

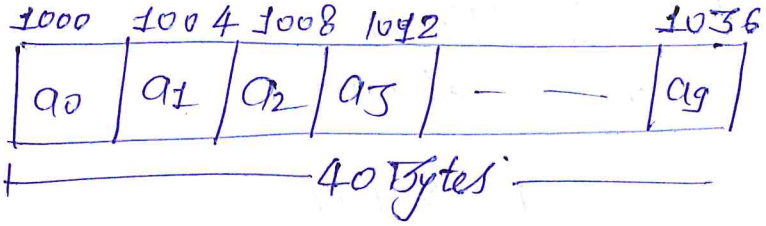
x = a[0]

⇒ In case of array we can directly access a particular element by its index.

a[0] - 0 indexed array
a[1] - 1 - indexed array



a[10] : integer takes 4 bytes in c programming



1000 - 1003 = a0
1004 - 1007 = a1

a | 1000 |

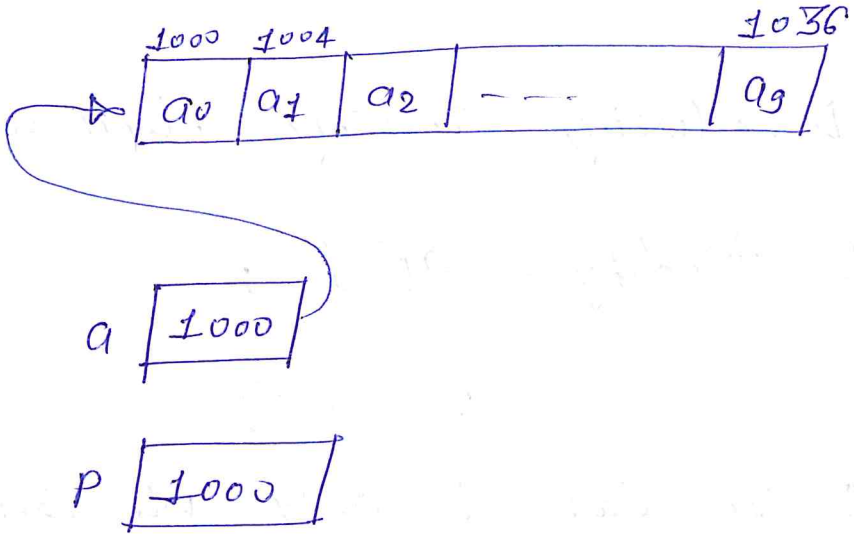
$a_i = 1000 + (i * 4)$
$= 1000 + 9 \times 4$
$= 1036$

Contiguous memory

2D Array : looks like matrix. It has length and width

Pointer arithmetic

int a[10]
int *P = a;



P+1 ≠ 1001
P+1 = 1004 P is the pointer to integer type data

[29.2] Multidimensional Array

2D Array : int a[10][20]
rows columns

m × n
↓ ↓
rows column

a_{00}	a_{01}	-	-	-	$a_{0,19}$
					$a_{1,19}$
					$a_{2,19}$
a_{190}					→ $a_{9,19}$

Memory is organized in two types:

- Row major order (RMO)
- Column major order (CMO)

↓

FOTRAN – (mathematical)

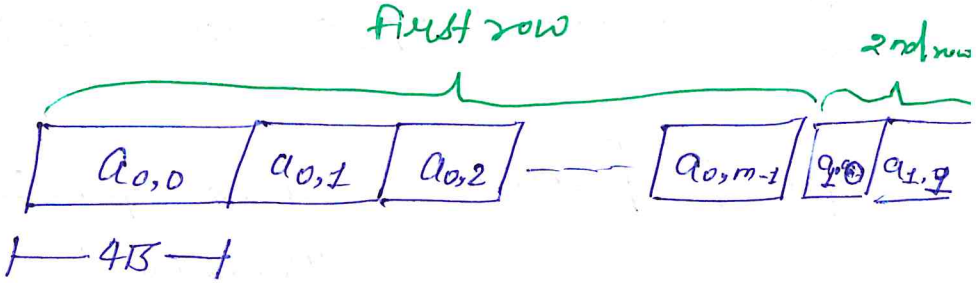
RMO

a_{00} a_{01} a_{02} ---- $a_0 a_{m-1}$

a_{10} a_{11} a_{12} ---- $a_1 a_{m-1}$

$a_{n-1,0}$ $a_{n-1,1}$ ---- $a_{n-1} a_{m-1}$

⇒ RMO is a format in which data is organized into memory

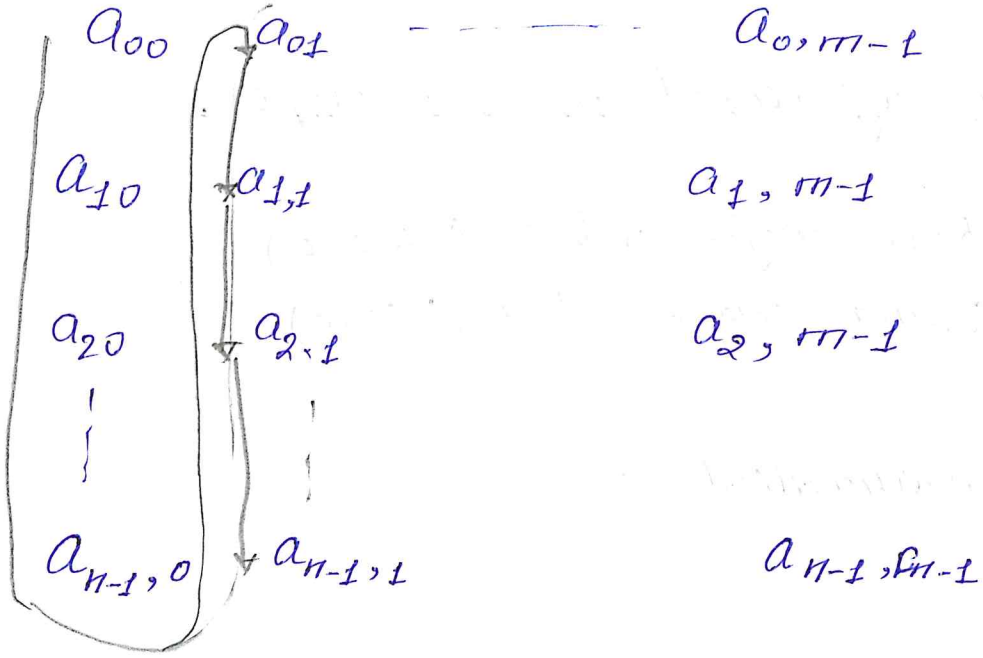


after completing first row, we start filling elements of 2nd row in the memory

$$\text{mem\_loc}(a_{i,j}) = 1000 + (m * i) * 4 + (j * 4)$$

B.A.

CMO



⇒ In column major order, After storing first column, we will start inserting from 1st element of 2nd row.

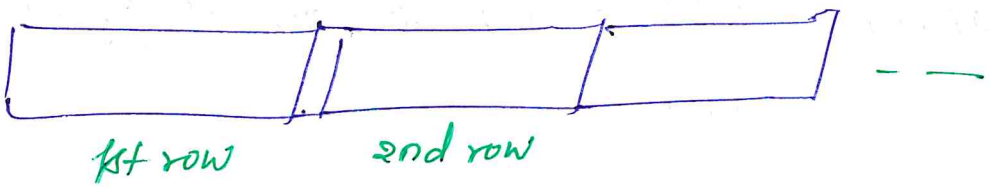
$$\text{mem\_loc}(a_{i,j}) = 1000 + (j * n) * 4 + i \times 1$$

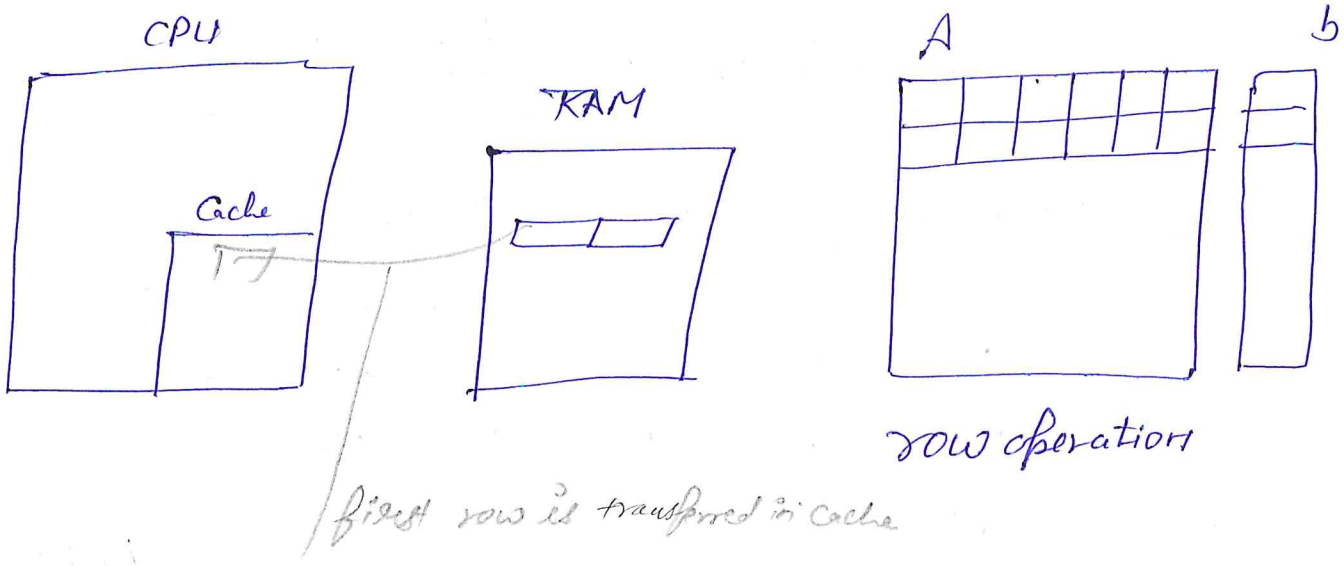
Why C uses RMO and FOTRAN uses CMO

for performing matrix operation, we need

Multiplication of 2 m
addition
Substraction
Inverse

row major





Matrix operation → Column operations

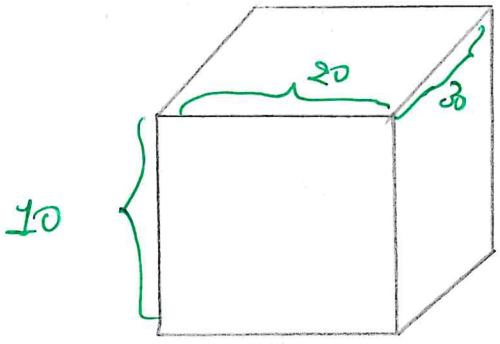
↓

CMO

Multi dimension array

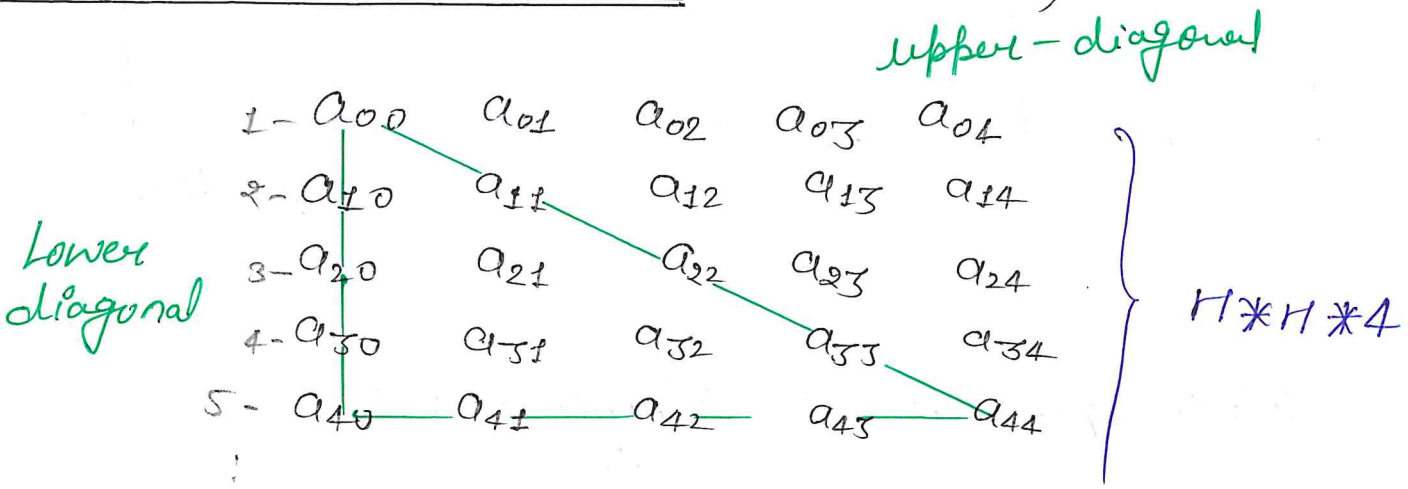
3D

int a[10][20][30]



Chapter ⇒11 Special types of 2D arrays

[30.1] Symmetric matrix (2D Array)



① Square matrix : No. of rows = No. of column

n x n

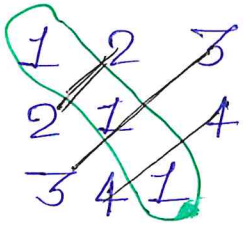
② Diagonal elements : a_{ij}

diagonal of the $a_{00}, a_{11}, a_{22}, a_{33}, a_{44}$

③ Symmetric matrix : If $a_{ij} = a_{ji}$ $\forall i, j$

$a_{01} = a_{10}$, $a_{20} = a_{02}$

Ex:



3x3 Symmetric matrix

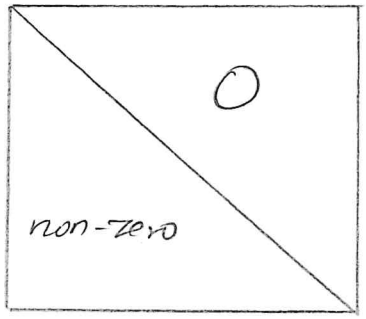
Lower triangle = 1+2+3 ---- n

$= \left[\frac{n(n+1)}{2}\right] * 4$

↑ each of 4B

[30.2] Lower triangular matrix & Diagonal matrix

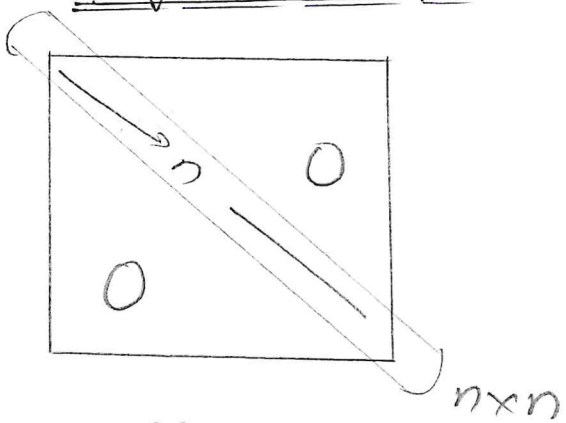
- All the elements, that are not part of lower diagonal matrix are zero. Then such matrix is called lower triangular matrix.



n^2

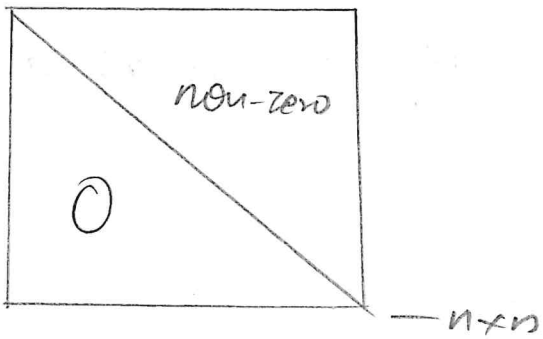
Instead of storing n^2 element just $n(n+1)/2$ elements will be stored

Diagonal matrix



$\Rightarrow 4 * n$

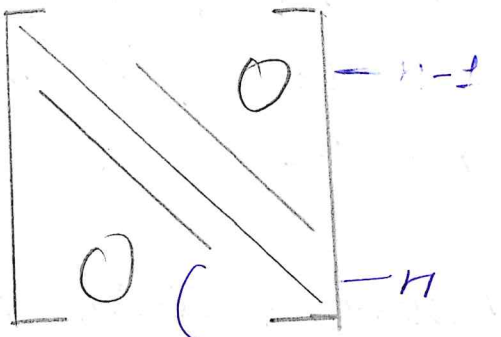
upper triangular



$\Rightarrow n(n+1)/2$

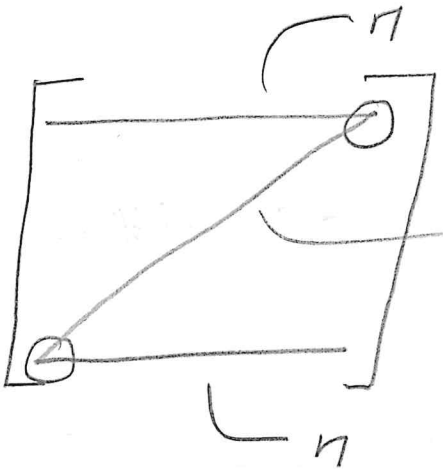
$\Rightarrow$ In all these matrix, we use RMO / CMO

[30.5] Tridiagonal matrix, Z-matrix, Teoplitz matrix



$3n-2$

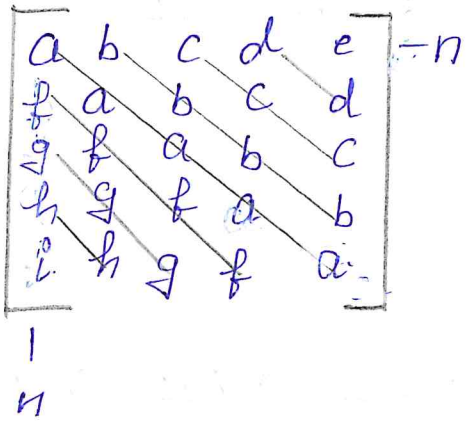
Z-matrix



$= 3n-2$

Teoplitz matrix:

all diagonal elements are same.



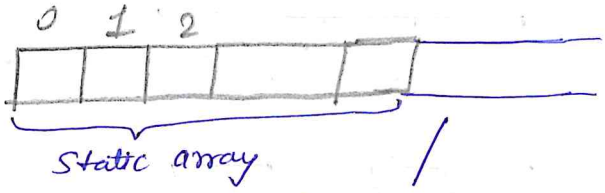
1
n

Chapter ⇒ 12

[31.1] Dynamic Arrays & amortized time

Dynamic Array: Size can be increased.

int a[10]

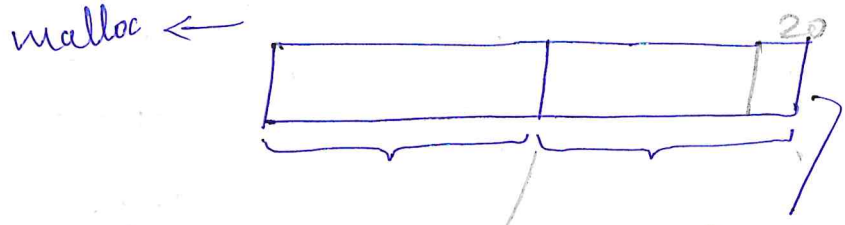


10 elements
insert (11)

Could already be in use

⇒ In C++, vector
In java, ArrayList

insert 11th element



insert 12th — 20th
insert 21st

allocating new memory

Regular array: O(1)

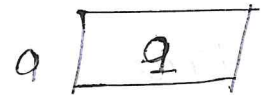
1 — 10 → 1
11 → 10+1
12 →

Dynamic:

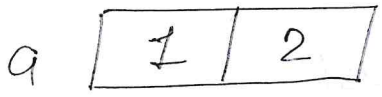
20 → 1
21 → 20+1

time it takes to insert an element =

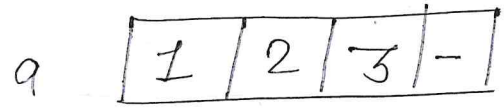
amortized analysis: average cost of insert operation averaged over many such operations



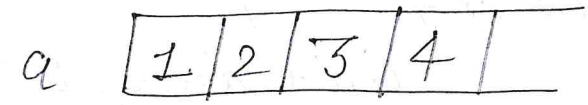
1 : 1



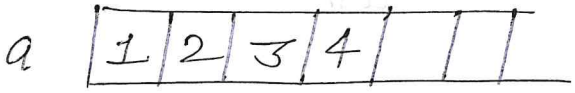
2 : 1 + 1 2^0



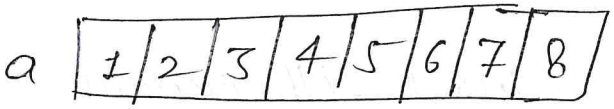
3 : 2 + 1 2^1



4 : 1



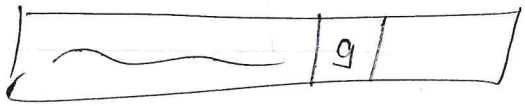
5 : 4 + 1 2^2



6 : 1

7 : 1

8 : 1



9 : 8 + 1 2^3

10 : 1

16 : 1

17 : 16 + 1 2^4

$n = 17$

$17 * 1 + \{2^0 + 2^1 + 2^3 + 2^4\}$

$\log_2 17$

4.

for any n:

$= (n * 1) + \{2^0 + 2^1 + \dots + 2^k\}$ s.t $k = \lfloor \log n \rfloor$

$= 2^{k+1} - 1$

$= 2 \cdot 2^k - 1 \quad = O(n)$

$n + O(n)$

$O(n)$ ⟶ n insertion

$O(1)$ ⟶ 1 insertion

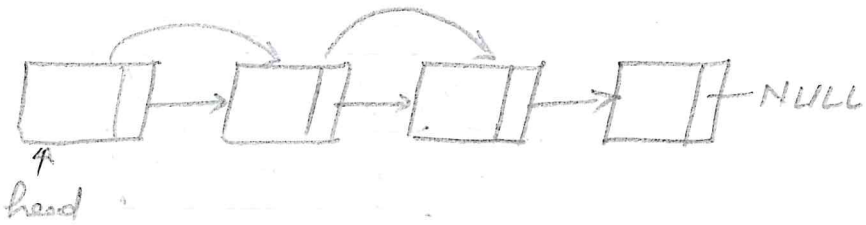
↓

amortized

Chapter ⇒ 13 Array vs linked list [32.1]

(1) Array is indexed ⟶ access ⟶ $O(1)$

Pointers, access ⟶ $O(n)$ ⟵ Linked list



(2) A : static (dynamic)

LL : dynamic

Chapter ⇒ 14 Solved Problems

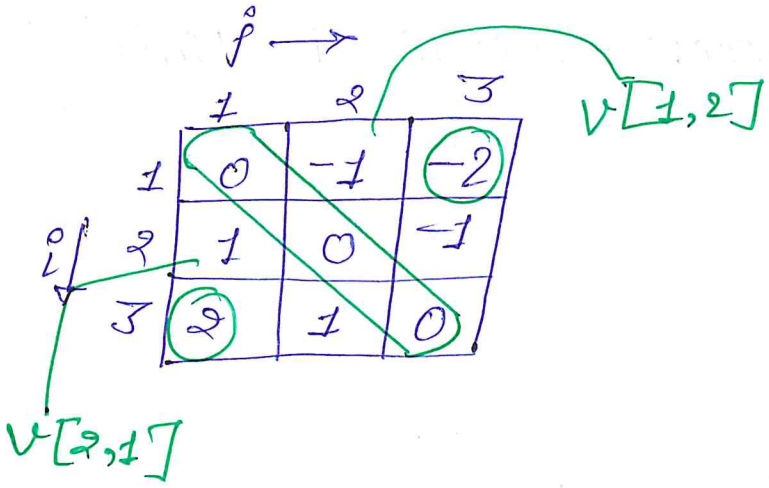
[33.1] GATE 2000

An $n \times n$ array v is defined as follows:
$v[i,j] = i-j$ for all i,j ; $i < n$, $1 \le j \le n$

The sum of the elements of the array v is:

1. 0
2. $n-1$
3. $n^2 - 3n+2$
4. $n^2 (n+1)/2$

Solution :



∴ Sum of elements = 0

[33.2] GATE 2002-2

Suppose you are given an array $S[1--n]$ and a procedure reverse (S, i, j) which reverse the order of elements in a between positions i and j (both inclusive). What does the following sequence do, where $1 <= k <= n$.

```
reverse (S, 1, k);
reverse (S, k+1, n);
reverse (S, l, n)
```

(A) Rotates S left by k-positions
(B) Leaves S unchanged
(C) Reverse all elements of S
(D) None of the above.

Solution:

$S[1 -- n]$

1, 2, 3 --- k-1, k, k+1 --- n-1, n

(1) k, k-1, --- 3, 2, 1, k+1 --- n-1, n

(2) k, k-1 ---- 2, 1, --- n, n-1 --- k+1

(3) k+1, k+2, --- --- n-1, n, 1, 2, 3 --- k-1

rotates S left by k positions

[3.3.3] GATE 2004

A single Array A[1 - MAXSIZE] is used to implement two stacks. The two stacks grow from opposite ends of the array. Variables top1 and top2 (top1 < top2) point to the location of the topmost element in each of the stacks. If the stack is used efficiently, the condition for stack full is

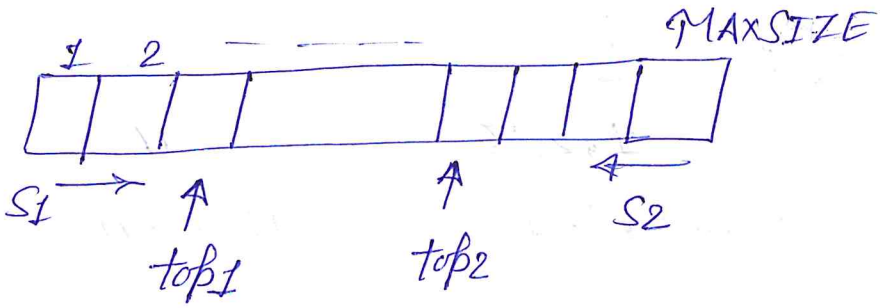
(A) (top1 = MAXSIZE/2) and top2 = (MAXSIZE/2+1)

(B) top1 + top2 = MAXSIZE

(C) top1 = (MAXSIZE/2) or (top2 = MAXSIZE)

(D) top1 = top2 - 1

Solution:



top1 = top2 - 1 Stacks are full when top1 and top2 are adjacent to each other.

[33.4] GATE 2005

A program P reads in 500 integers in the range [0--100] representing the score of 500 students. It then prints the frequency of each score above 50. What would be the best way for P to store the frequencies.

(A) An Array of 50 numbers
(B) An Array of 100 numbers
(C) An Array of 500 numbers
(D) A dynamically allocated array of 550 nums

Solution:

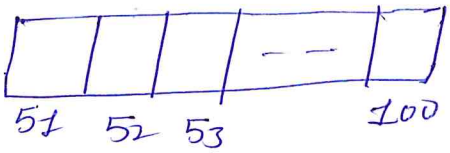
freq = No. of times an integer occurs among 500 integers:

55 - occurs 4 times ⇒ freq(55) = 4

freq(51)
freq(52)
freq(53)
:

freq(100)

freq(50+i)



[33.5] GATE 1994

In a comment single dimension array of lower triangular matrices (i.e all the elements above diagonal are zero) of size $n \times n$, non-zero elements (i.e: elements of lower triangle) of each row are stored one after the another, starting from the first row, the index of the (i,j)th element of the lower triangular matrix in this new representation is

(A) $i+j$ (B) $i+j-1$ (C) $(j-1)+i(i-1)/2$ (D) $i+j(j-1)/2$

Solution:

	j			
	1	0	0	0
	2	3	0	0
i	4	5	6	0
	7	8	9	10

row major

4X4

$\frac{i(i+1)}{2} + (j-1)$

1 : 1
2 : 2
3 : 3
⋮
$i-1 : i-1$

$1+2+3+\cdots i-1$

$\frac{i(i-1)}{2}$

$(j-1) + \frac{i(i-1)}{2}$

[33.6] GATE 1998

Let A be a 2D array declared as follows

A : array [1--10][1--15] of integers

Assuming that each integer takes one memory location the array is stored in row-major order and the first element of the array is stored at location 100. What is the address of the element A[i][j].

(A) $15i + j + 84$

(B) $15j + i + 84$

(C) $10i + j + 89$

(D) $10j + i + 89$

row = 10 − 1 + 1 = 10

Column = 15 − 1 + 1 = 15

Solution BA = 100

row major order:

$$A[i][j] = BA + (i-1) \times 15 + (j-1)$$

$$= 100 + 15i - 15 + j - 1$$

$$= 15i + j + 100 - 16$$

$$= 15i + j + 84$$

1	100	101	102	—	—	—	114
2	115	116	—	—	—	—	129
⋮							
10							

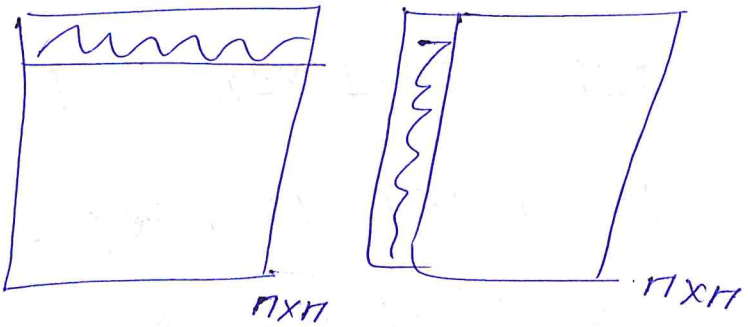
[33.7] GATE 2004

Two matrices M1 & M2 are to be stored in arrays A & B respectly. Each array can be stored either in row major or column major order in contiguous memory locations. The time complexity of an algorithm to compute M1 × M2 will be

(A) best if A is RM, and B is CM
(B) best if both are RMO
(C) best, if both are CMO
(D) ~~-~~ independent of storage scheme

Solution:

$A[i][j] \Rightarrow C = A * B$



M1 row major

M2 column

$C_{ij} = O(n)$

C is $n \times n = n^2$ elements

$O(n^3)$

Efficient Method

(A)

[33.8] GATE 2015

A Young tableau is a 2D array of integers increasing from left to right and from top to bottom. Any unfilled entries are marked with ∞. and hence there can not be any entry to the right o or below a ∞. The following young tableau conse of unique entries.

1	2	5	14
3	4	6	23
10	12	18	25
31	∞	∞	∞

When an element is removed from a Young tableau other elements should be moved into its place so that the resulting table is still a Young tableau (unfill entries may be filled with a ∞). The min. no of entries (other than 1) to be shifted, to removed 1 from the given Young tableau is ____

(A) 2 (B) 5 (C) 6 (D) 18

Solution

~~1~~ ← 2	4	5	14
3	4 ↑ 6	← ~~6~~	23
10	12	18	25
31	∞	∞	∞

(Can not move 5 as it must be in increasing order)

2	4	5	14
3	6	18	23
10	12	25	∞ (New)
~~31~~	∞	∞	∞

[33] Gate 2002

Consider the following declaration of a 2D array in C

```
Char a[100][100];
```

Assuming that the main memory is byte-addressable and that the array is stored starting from memory address 0, the address of a[40][50] is

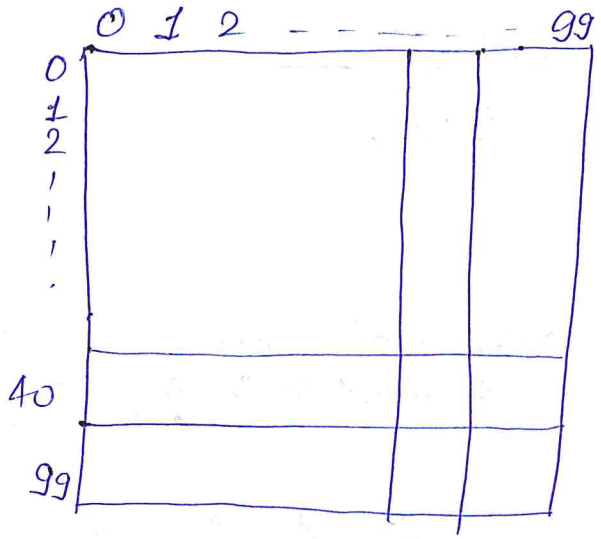
(A) 4040 (B) 4050 (C) 5040 (D) 5050

Solution:

$$A[40][50]$$

$$= BA + (40-0) \times 100 + (50-0)$$

$$= 0 + 4000 + 50 = 4050$$



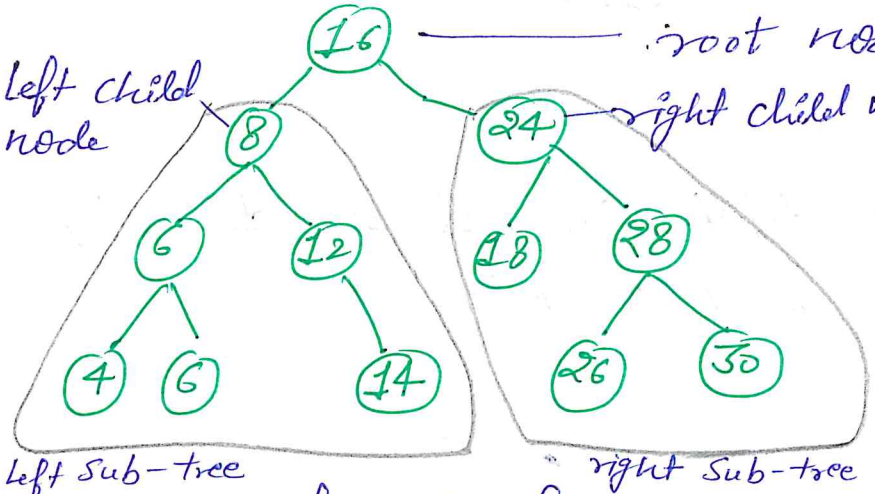
Chapter ⇒ Binary Search tree

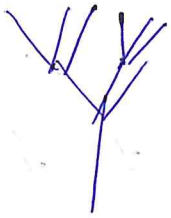
[38.1] Binary Search Tree : Intuition & Terminology

1960's

Collection of similar items
Structure (Tree)
operations ↓
Search → O(log n)

(Looks like tree)





looking like inverted-tree

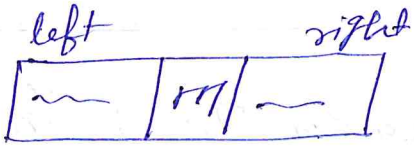
8 is parent of 6 & 12 where 6 is left child 12 is right child

Leaf nodes : Nodes which don't have children are known as leaf nodes.

Internal Nodes : All nodes except leaf nodes are called internal nodes.

⇒ For every node, you have either 2-children or no children. That's why it is known as the binary tree.

m-ary tree



Depth of a tree : depth is max depth of any node.

root is considered at 0

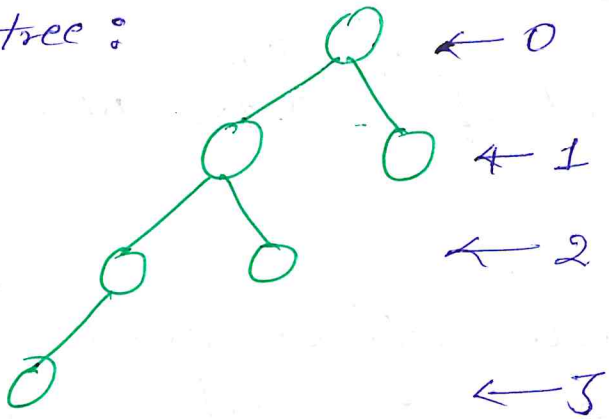
8, 24 ⟶ at depth 1

6, 12, 18, 24 ⟶ depth 2 (2 hop from root)

4, 6, 14, 26, 30 ⟶ depth 3

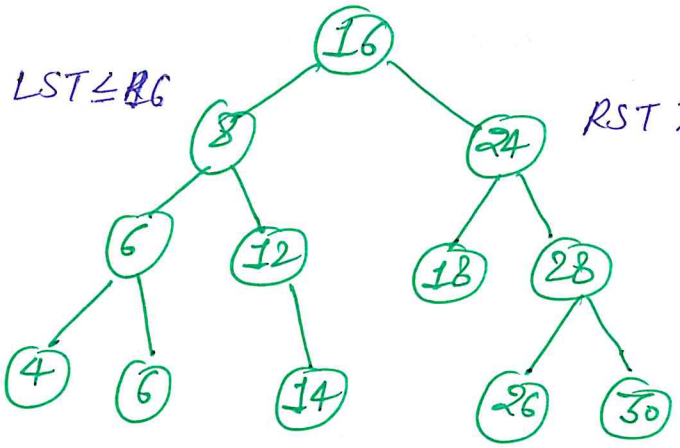
In Balanced tree depth can not be differ by more than 1 for LST & RST.

Unbalanced tree :



3-1 = 2

So this is not balanced tree.



⇒ BST is a Special tree

⇒ all nodes in LST are less than root.

⇒ All nodes in RST are greater than root.

Structure of a BST :

(1). Tree

(2). Binary Tree

(3). LST ≤ root & RST ≥ root

Implementation using Pointers / References [39-1]

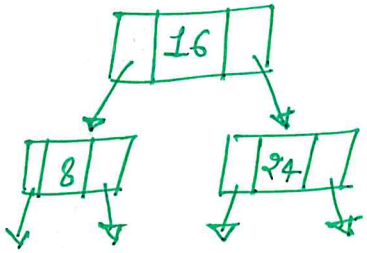
↳ C/C++ └ Java

root node has data & 2 pointers

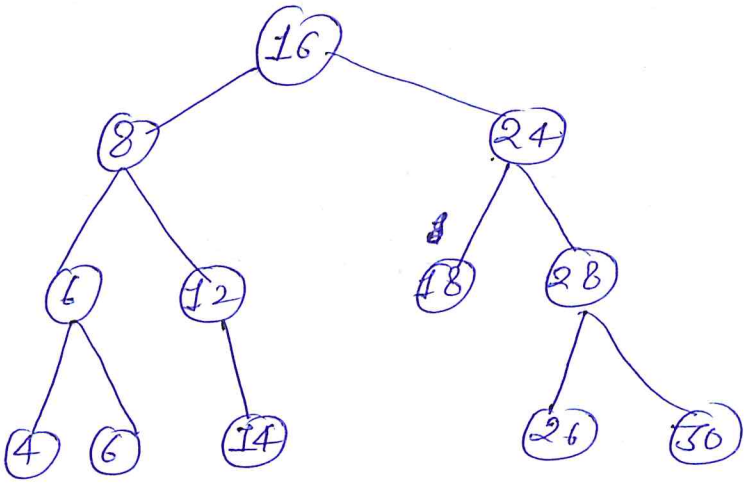
left pointer — LST

right pointer — RST

```
Struct node {
        int data ;
        Struct node * left ;
        Struct node * right ;
}
```



[39.1] BST : implementation using Array



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	8	24	6	12	18	28	4	6	-	14	-	-	26	30

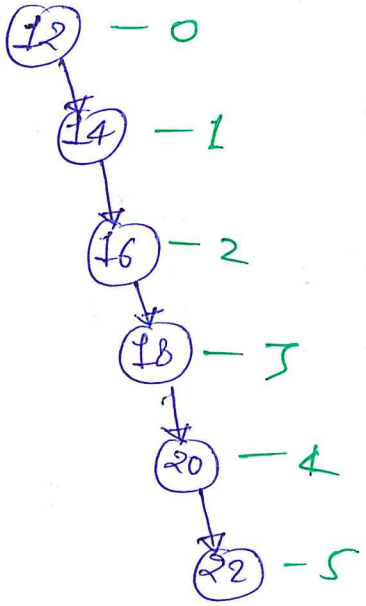
(1) root → 1

(2) node → i

left child → 2i

right child → 2i+1

⇒ In this implementation space is wasting becaues we are leaving spaces for nodes even if they are not in tree.



⇒ lot of space is wasting here.

⇒ I need an array of space 63.

Skewed trees

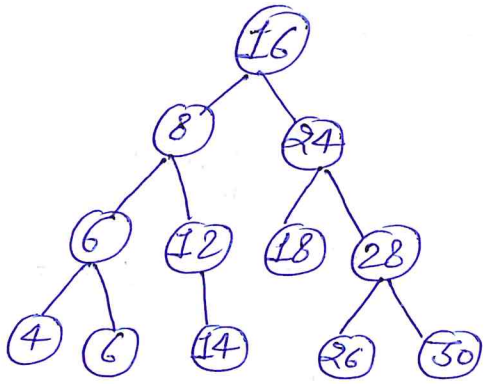
Depth of 5

Operations

[40.1] Build a tree

how to build a binary tree from the given list of numbers.

16, 8, 24, 6, 12, 18, 28, 4, 6, 14, 26, 30

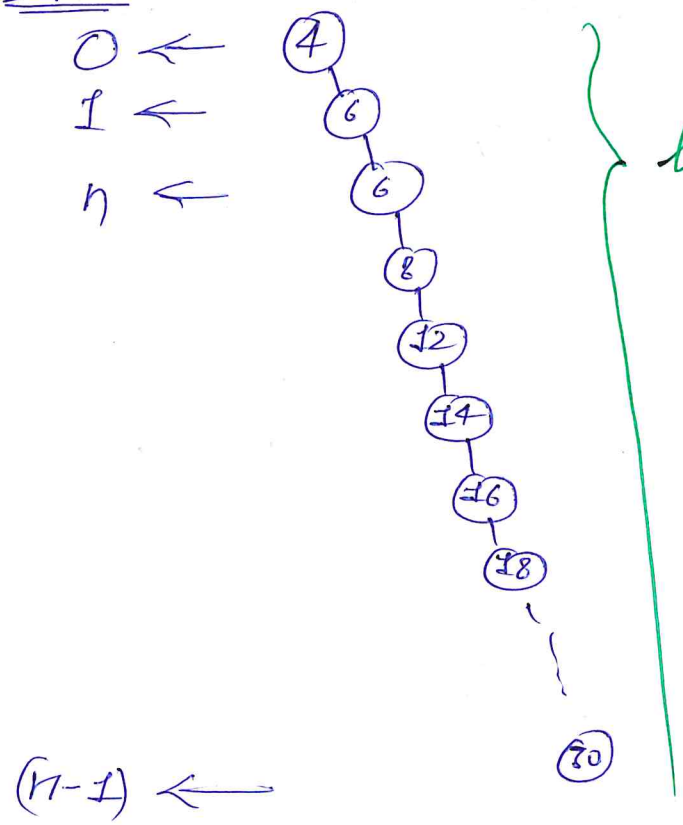


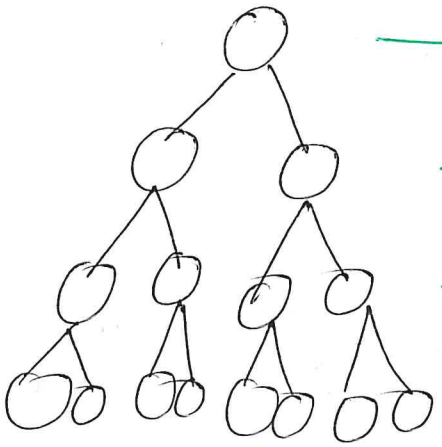
Put node < 16 @ left
node > 16 @ right

comp(6, 16) 6 < 16
comp(6, 8) 6 < 8

order is 4, 6, 6, 6, 8, 12, 14, 16, 18, 24, 26, 28, 30

Depth





→ 0 $(1) = 2^0$

→ 1 $(2) = 2^1$

→ 2 $(4) = 2^2$

— 3 $(8) = 2^3$

$\lfloor \log_2 15 \rfloor =$

→ If n-element binary tree

depth $= \lfloor \log_2 n \rfloor = O(\log_2 n)$ } Best case

$O(n)$ } Worst case

⇒ What is the time complexity to build a BST. with n-elements

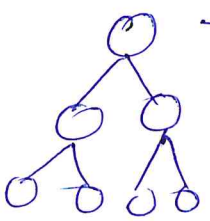
Worst case:

to insert ith element, I need to make (i-1) comp

$0 + 1 + 2 + \cdots + n = O(n^2)$

Best case:

Comp



1 — 0 } 2^0 $\log_2(1)$

2 — 1 } 2^0 $\log 2$

3 — 1

4 — 2 } 2^1 $\log$

5 — 2

6 — 2

7 — 2

8 — 3

⋮

15 — 3

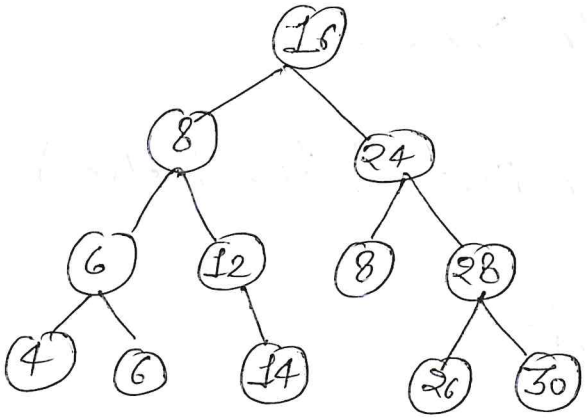
ith element → $O(\log i)$

ith element → $\lfloor \log i \rfloor$

$\sum_{i=1}^{n} \log i$

time complexity for comparing $= O(n \log n)$

[40.2] operations : Search, insert, Min and Max



3.31

⇒ search for (14)

comp (14, 16) - 14 < 16 goto LST
comp (8, 14) — 8 < 14 goto RST of LST
comp (12, 14) — 12 < 14 go right
— Found

Ex: search for x = 13

comp (16, 13) — 13 < 16 goto LST
comp (8, 13) — 13 > 8 goto right of LST
comp (12, 13) — 13 > 12 go down
comp (14, 13) — 13 < 14 there is no element as 13

```
1. def search_recursively (key, node):
2.     if node is None node.key == key:
3.         return node
4.     if key < node.key :
5.         return search_recursively (key, node.left)
6.     # key > node.key
7.     return search_recursively (key, node.right)
```

1) What is time complexity of search.

Time complexity of search is equal to depth of Binary Search Tree.

$O(n)$ – in worst case (Skewed tree)

$O(\log n)$ – BC & AC

2) how to find Min & Maximal element in BST

Minimal :

Smaller than Root (Root) greater than root

By recursively following left sub-tree till we reach at dead-end or a null pointer

Maximal :

By recursively following right sub-tree till we reach at dead-end or a null-pointer.

Time complexity = depth of the tree

$O(n)$ Worst

$O(\log n)$ best & avg.

(3) insert an element

insert 13 into BST (1) search for $x = 13$

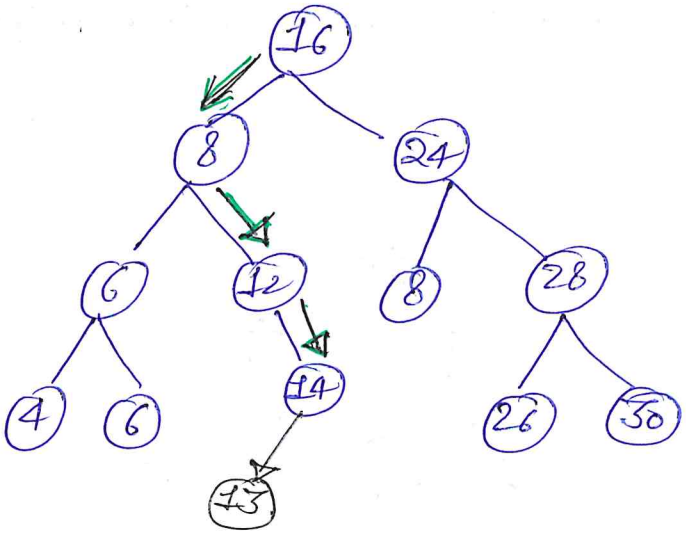
follow this path

$13 < 16$

$13 > 8$

$13 > 12$

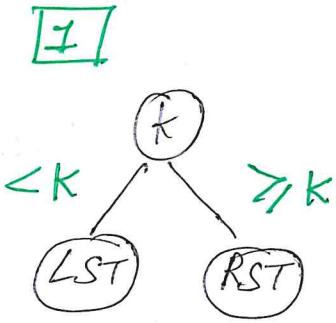
$13 < 14$

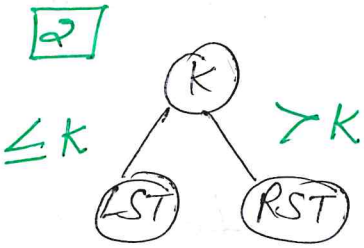


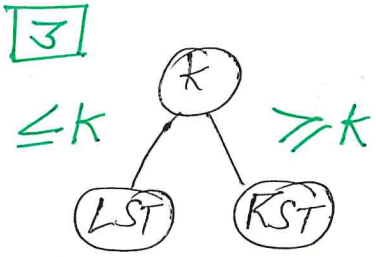
(2) just insert node at the end of dead end

Case 2: insert $x = 12$

(1) search for 12 in BST







[40.3] Traversal: In-order / Sort, Preorder, Post-order

Traversal means accessing each of the element of BST. There are three ways of performing traversal in BST.

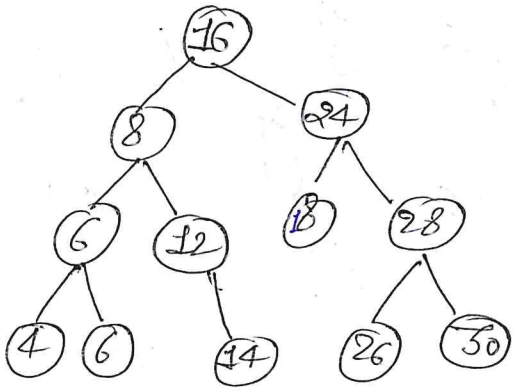
(1) Inorder (sort)

(2) Preorder

(3) Post-order

Inorder Traversal : (Left node) (Root node) (right node)

do it recursively



4, 6, 6, 8, 12, 14, 16, 18, 24, 26, 28, 30 (Sorted list)

14: inorder predecessor of 16

18: inorder successor of 16

Preorder traversal :

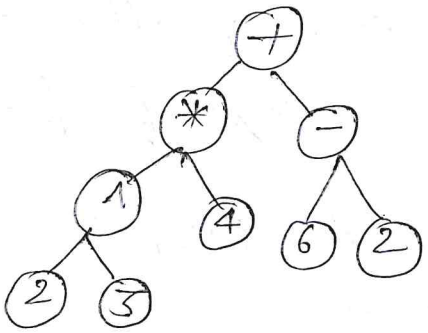
Root node, left node, right node

16, 8, 6, 4, 6, 12, 14, 24, 18, 28, 26, 30

Post-order traversal: Left, Right, Root (recursively)

Other approaches :

Expression trees are binary trees not BST.



leaf nodes — operands.

non-leaf nodes — operators.

(Infix, Prefix, Postfix)

$((2 \wedge 3) * 4) + (6-2)$

① in-order traversal → infix expression
② Pre-order traversal → Prefix expression

Rt, L, R

+ * ^ 2 3 4 − 6 2

③ Post-order traversal → Post-fix expression

L, R, Rt 2 3 ^ 4 * 6 2 − +

Time complexity of total Traversal :

$T.C. = \theta(n)$

⇒ Pre-order, Post-order for binary search tree do not have such deep applications but for binary trees like prefix, postfix expressions these are used.

[40.4] Delete : operations

Case 1: a pointer to the node, that I want to delete.

If node has no child → leaf node
Simple deallocate the memory and adjust/add the NULL pointer to its parent.

Case 2: node has one child.

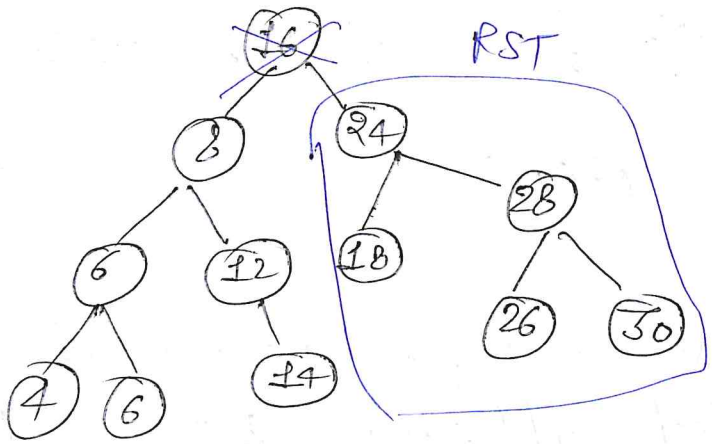
nodes Parent → nodes Only parent

Case 3: node has two children

16 replace node with its inorder successor

Smallest element in the Right sub-tree

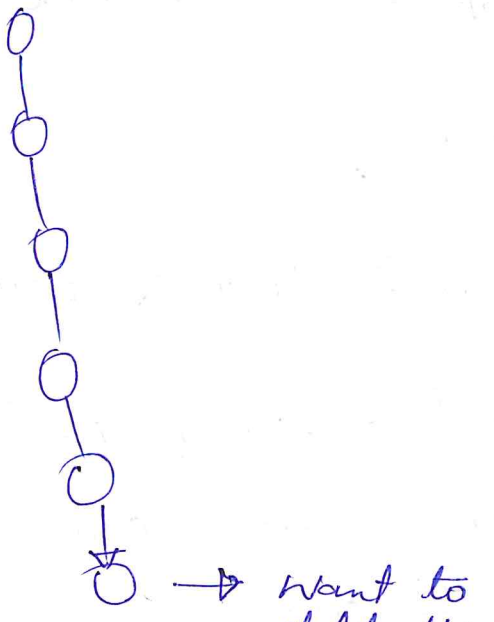
here 16 will be replaced by 24.



Delete 16

↓

replace by 18



→ want to delete this

Worst case to delete an element is $\theta(n)$

$\theta(\log n)$

$\theta(\log n)$

Find smallest in RST:

WC = $\theta(n)$

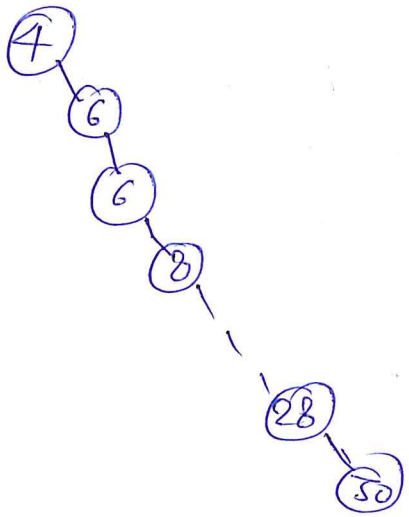
TC = $\theta(\log n)$

[41.1] Randomized BST

List : 4, 6, 6, 8, 12, 14, 16, 18, 24, 26, 28, 30 (Sorted order)

BST (constructing)

(Worst Case)



Skewed tree

Operations: $O(h)$: height of the tree

$h = O(n)$ worst case (Skewed tree)

$h = O(\log(n))$ → balanced tree

Ques: even if we are given a sorted list as input, can we avoid the extremely skewed tree.

Randomization → Quick Sort (to pick pivot randomly)

Chapter ⇒ Solved Problems

[42.1] Gate 2003

The usual $\Theta(n^2)$ implementation of insertion sort to sort an array uses linear search to identify the position where an element is to be inserted into the already sorted part of the array. If instead, we use binary search to identify the position Worst case running time will

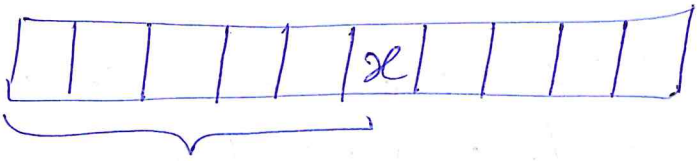
A. remain $\Theta(n^2)$

B. become $\Theta(n(\log n)^2)$

C. become $\Theta(n \log n)$

D. become $\Theta(n)$

Solution:



Sorted

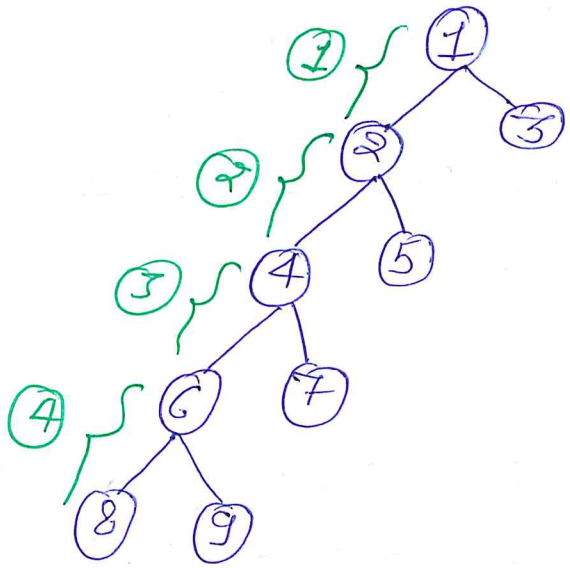
GATE 2018 [42.2]

The Post-order traversal of a binary tree is 8, 9, 6, 7, 4, 5, 2, 3, 1. The in-order traversal of the same tree is 8, 6, 9, 4, 7, 2, 5, 1, 3. The height of a tree is the length of longest path from root to any node (leaf). The height of the binary tree above is.

Solution:

Post — 8, 9, 6, 7, 4, 5, 2, 3, (1) (L, R, Rt)

in-order — 8, 6, 9, 4, 7, 2, 5, (1), 3 (L, Rt, R)



$h = 4$

It is just Binary tree not BST

[42.3] GATE 2017

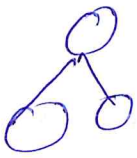
Let T be a binary search tree with 15 nodes. The Min. & Max. possible heights of T are.

Note: height of tree with single node is 0.

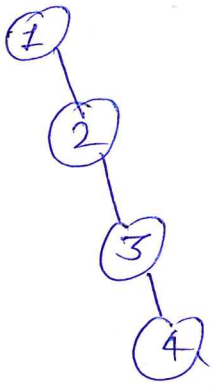
(A) 4 & 15 (B) 3 & 14 (C) 4 & 14 (D) 3 & 15

Solution:

→ 0



→ 1



nodes

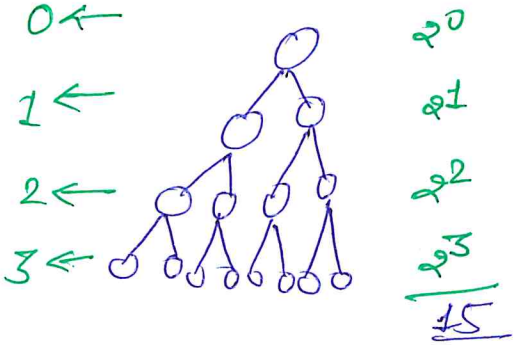
→ 0 2^0

→ 1 2^1

→ 2 2^2

→ 3 2^3

15 → 14 15



$0 \rightarrow 1 = 2^0 = 2^1 - 1$

$1 \rightarrow 1+2 = 3 = 2^2 - 1$

$2 \rightarrow 1+2+4 = 7 = 2^3 - 1$

$3 \rightarrow 1+2+4+8 = 15 = 2^4 - 1$

$2^{k+1} - 1$

[42.4] GATE 2007

Let A be an array of 31 numbers consisting of a sequence of 0's followed by a sequence of 1's. The problem is to find the smallest index i such that A[i] is 1 by probing the min. no. of locations in A. The worst case number of probes performed by an optimal algorithm is. (A) 2 (B) 3 (C) 4 (D) 5

Solution: 1, 2, 3, 4 ---- 31

0 0 0 ---- 1 1 1 — 1 ← Sorted

Binary search:

1, 2, 3, 4 ---------- 51

0, 1, 1, 1 ---------- 1

$m = \left\lfloor \frac{l+r}{2} \right\rfloor = 16$ $\quad A[16] = 1$

$l = 1, \quad r = 16$

$\left\lfloor \frac{1+16}{2} \right\rfloor = A[8] = 1$

$l = 1, \quad r = 8$

$\left\lfloor \frac{1+8}{2} \right\rfloor = A[4] =$

④ $l = 1, \quad r = 4$

$m = 2 \qquad A[2] = 1$

Smallest index where $a[i] = 1$ should be 2

⑤ $l = 1, \quad r = 2$

$m = 1 \qquad A[1] = 1$

[425] GATE 2003

Let $T(n)$ be the no. of different binary search trees on n distinct elements.

$$T(n) = \sum_{k=1}^{n} T(k-1)\, T(x)$$ where x is

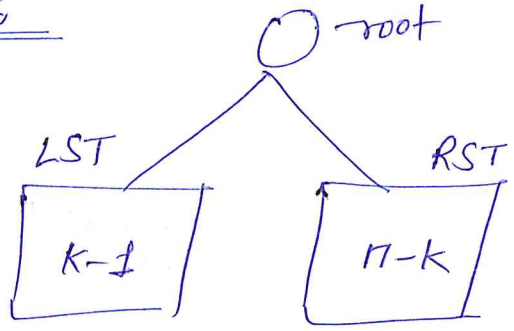
(A) $n-k+1$

(B) $n-k$

(C) $n-k+1$

(D) $n-k-2$

Solution :

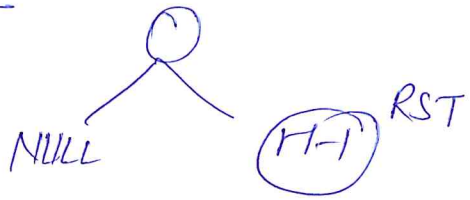


$n = 1 + k-1 + RST$

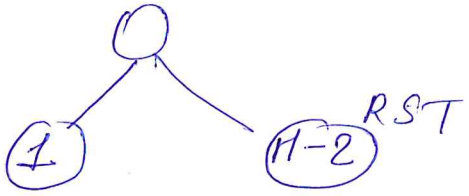
$RST = n-k$

$k = 1, 2, 3 - - - - - n$

When $k=1$



for $k=2$



[42.6] Gate 2005

Suppose the numbers 7, 5, 1, 8, 3, 6, 0, 9, 4, 2 are inserted in that order into an initially empty binary search tree. The BST uses the usual ordering on natural numbers. What is in-order traversal sequence of resultant tree.

(A) 7 5 1 0 3 2 4 6 8 9
(B) 0 2 4 3 1 6 5 9, 8, 7
(C) 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
(D) 9, 8, 6, 4, 2, 3, 0, 1, 5, 7

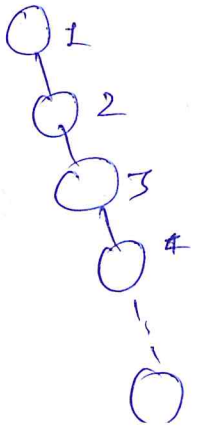
Solution: inorder traversal - automatically generates sorted array.

So 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

[42.7] GATE 2013

What is the tightest upper bound that represents the time complexity of inserting an object into a BST of n-nodes. (A) O(1) (B) O(log n) (C) O(n) (D) O(n log n)

Solution:



n+1 = BST becomes like

[42.8] GATE 2015

What are the worst case complexities of insertion and deletion of a key in a BST.

(A) $\Theta(\log(n))$ for both insertion and delete

(B) $\Theta(n)$ for both "

(C) $\Theta(n)$ for insertion, $\log n$ for deletion

(D) $\Theta(\log n)$ for insertion and $\Theta(n)$ for deletion

Solution:

In worst case BST behaves like LL

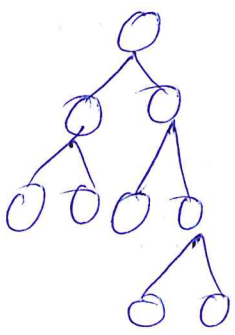
so $\Theta(n)$ for insertion & deletion.

[42.9] GATE 2011

Why are given a set of n-distinct elements and an unbalanced binary tree with n-nodes In how many ways can we populate the tree with the given set so that it becomes a BST

(A) 0 (B) 1 (C) $n!$ (D) $\left(\frac{1}{n+1}\right)^{2n}C_n$

Solution:



unlabelled

structure of the tree is fixed.

$n=9$, 1,2,3,4,5,6,7,8,9

there is only one single way

[42.10] GATE 1994

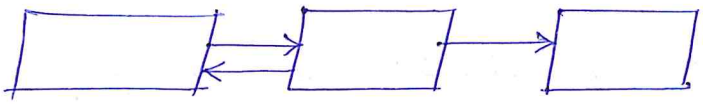
Linked-Lists are not suitable data structures for which one of the following problems.

(A) Insertion Sort

(B) Binary search

(C) Radix sort

(D) Polynomial Manipulation



m

l r

array

a[m] → θ(1)

LL → θ(n)

[42.11] GATE 2014

Consider the C function given below. Assume that the array listA contains n (>0) n (>0) elements sorted in ascending order.

```
int ProcessArray (int * listA, int x, int n)
{
 int i, j, k;
 i=0; j= n-1;
 do{
     k= (i+j)/2;
  if (x<= listA[k]) j= k-1;
  if ( listA[k]<= x) i= k+1;
  }
  while (i<=j)
  if (listA[k]== x) return(k);
  else
      return -1
}
```

Middle

change if x<

change if x>

Binary search

Which one of the following statements about the function ProcessArray is correct

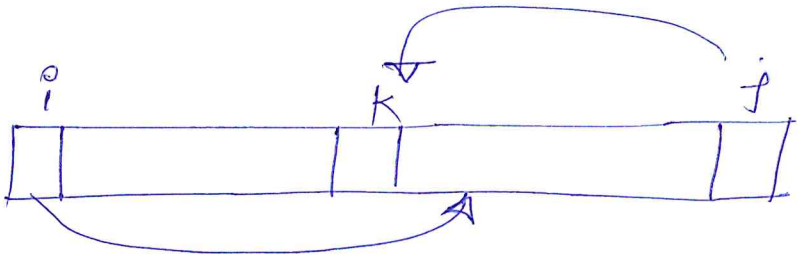
(A) It will run into an infinite loop when x is not in listA

(B) It is an implementation of binary search

(C) It will always find the max element in listA.

(D) It will return -1 even when x is present in listA.

Solution:



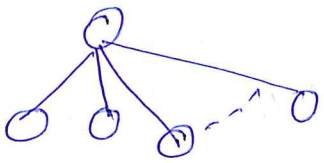
Trees

43.1 Logical structure & implementation

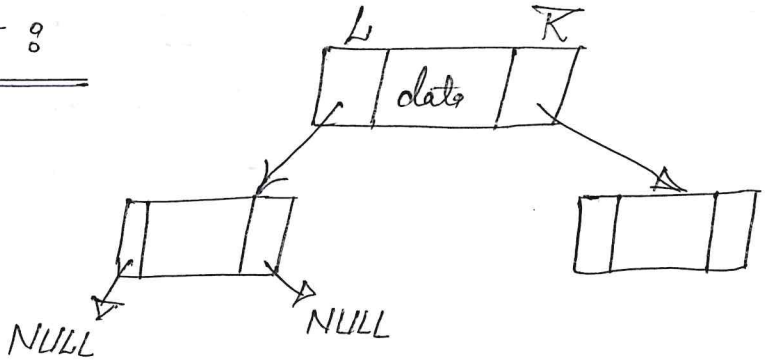
⇒ general purpose trees & Binary trees.

Binary Tree : Every node in a binary tree has a maximum of 2-children.
(2-ary Tree)

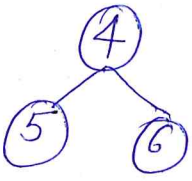
K-Ary Tree : Every node in a k-ary tree has a maximum of n-children.

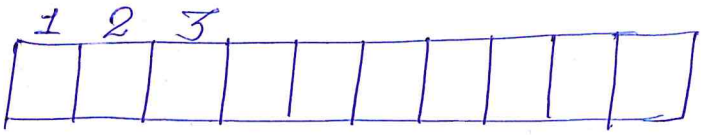


Represent :



ith index {1, 2, 3 - - - - n}

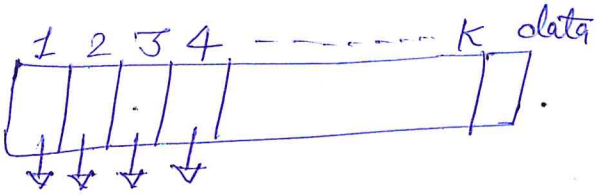




Left child = 2 * i

Right child = 2 * i + 1

K-ary tree :



arrays pointers / references

ith index → node
1st child → k*i
2nd child → k*i +1
3rd child → k*i +2

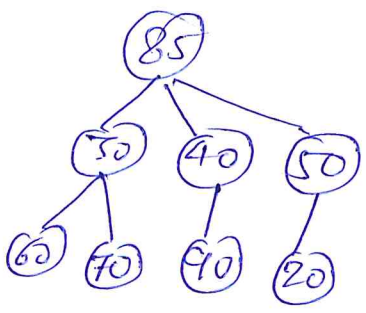
kth child → (k*i) +(k-1)

If we are using array with index 0 for storing k-ary tree then for getting index of a particular node we can use the formula

$k * i + c$ where k is arity
i is index
c is child

for example:

ternary tree is given and we try to find its first child of root



0	1	2	3	4	5	6	7	8	9	10	11	12
85	30	40	50	60	70	-	90	-	-	20	-	-

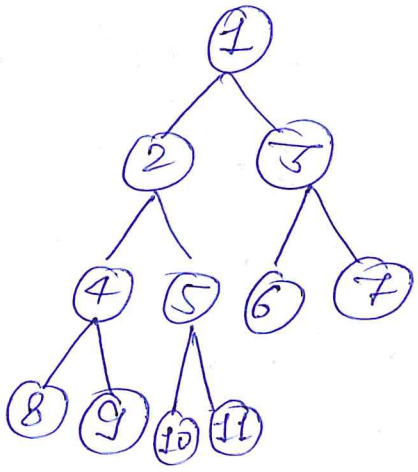
getting first child of 85

= k*i +c

= 3*0 +1 = 1 (location in array)

index of 85

[43.2] Terminology & Traversal



Root : 1

Child : 2, 3 are child of 1

Parent : 1 is Parent of 2, 3

Sibling : 2, 3 are siblings (same parent node)

Descendent : ↓

Ancestor : ↓ all grandchildren & all are descendent ~~of~~

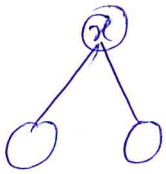
4 is descendent of 1

common ancestors of 7, 6 → 1

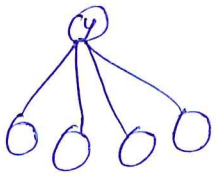
Leaf node {External nodes}

Internal nodes (non-leaf node)

Degree of node : No. of children, a node has



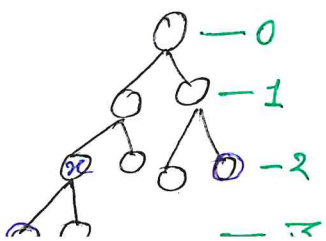
$deg(x) = 2$



$deg(y) = 4$

$deg(z) = 0$

Depth :

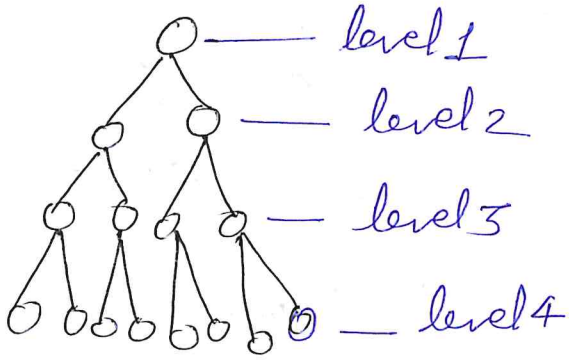


No. of edges are between nodes & root.

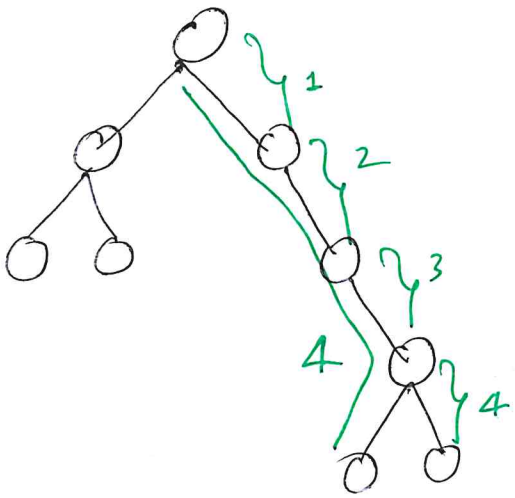
$depth(x) = 2$

level : 1 + No. of edges between node & root:

1 + depth of the node

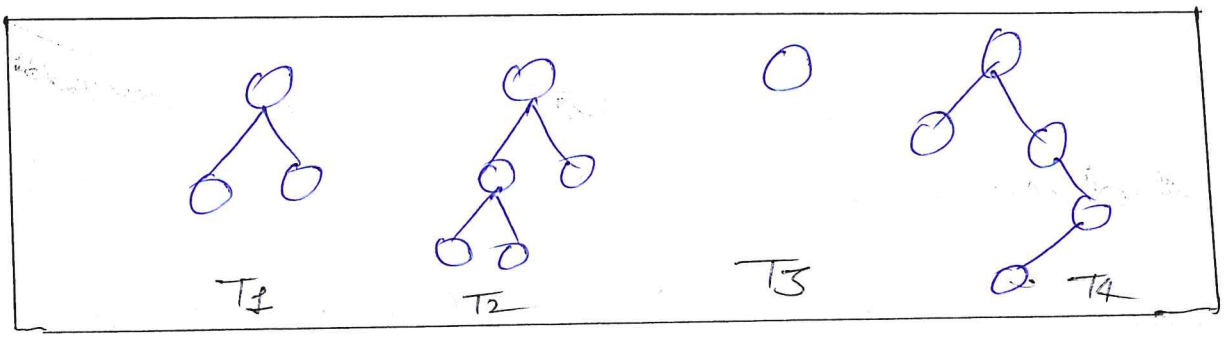


height of a tree : Number of edges on the longest path from any node to root.



∴ height = 4

Forest Set of trees that are disjoints

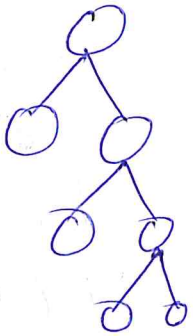


Traversal: — Inorder | Pre-order | Post-order

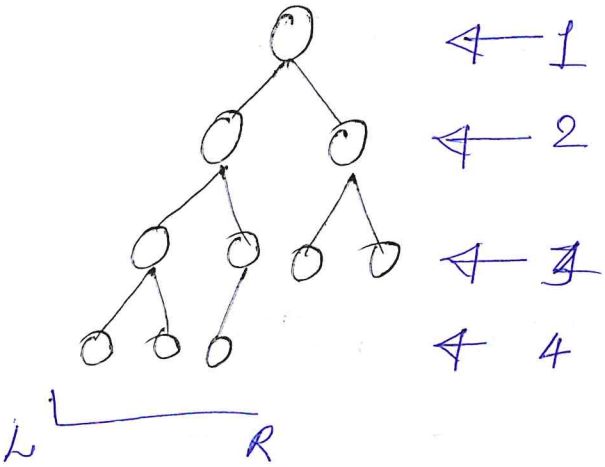
L, Rt, R | Rt, L, R | L, R, Rt

[435] Types of Binary Trees

① Full Binary Tree: Every node has 0 or 2 children.

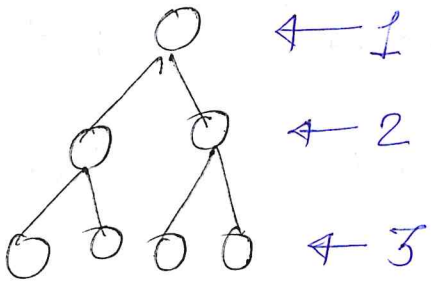


② Complete Binary Tree: Except last level, tree is completely filled and in the last level leaves are presented L to R.

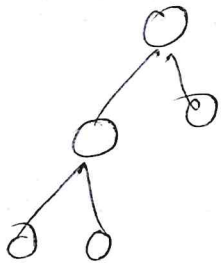


③ Perfect Binary Tree: ① all internal nodes have 2-children

② all leaf are at same level.



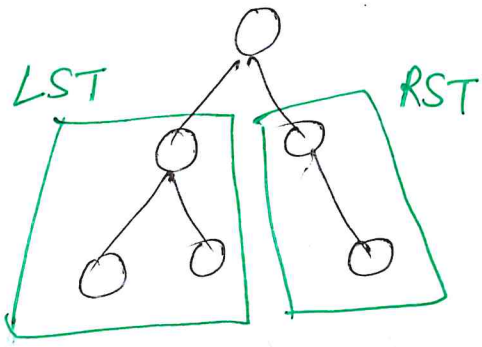
Interior = non-leaf



Complete BT but not Perfect B.T.

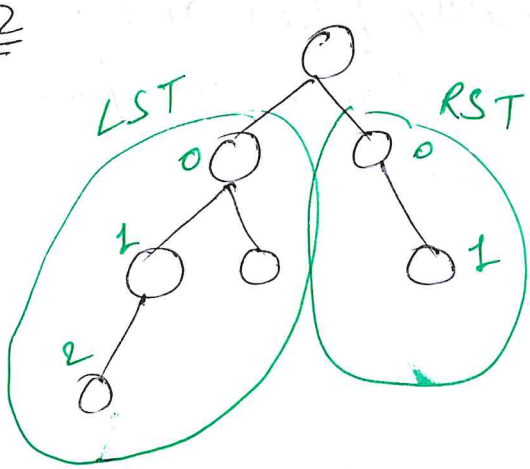
Balanced Binary Tree : At any node the height of LST & RST do not differ by more than 1

Ex:1



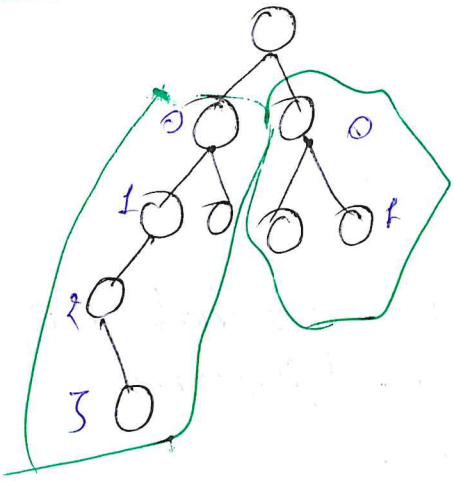
$h(LST) \rightarrow$
$h(RST)$

Ex:2



$= h(LST) - h(RST)$
$= 2 - 1 = 1$

Ex:3

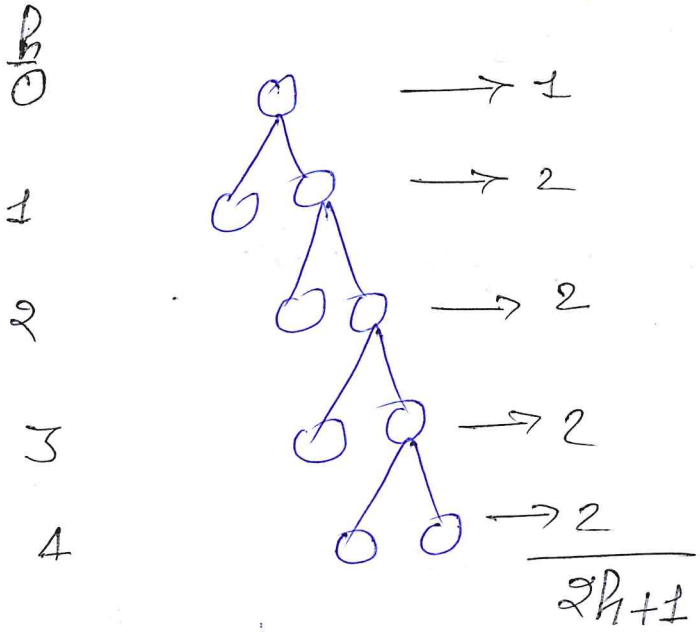


$= h(LST) - h(RST)$
$= 3 - 1 = 2$

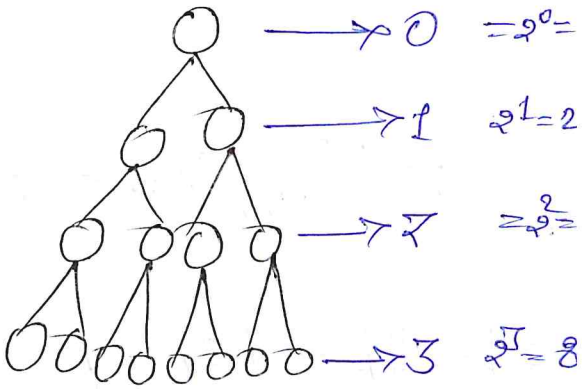
[43.4] Properties of a Tree : Depth, Nodes, Leafs

Full BT : n = no. of nodes, h = height
l = No. of leaves

Minimum & maximum no. of nodes



FBT & Perfect BT



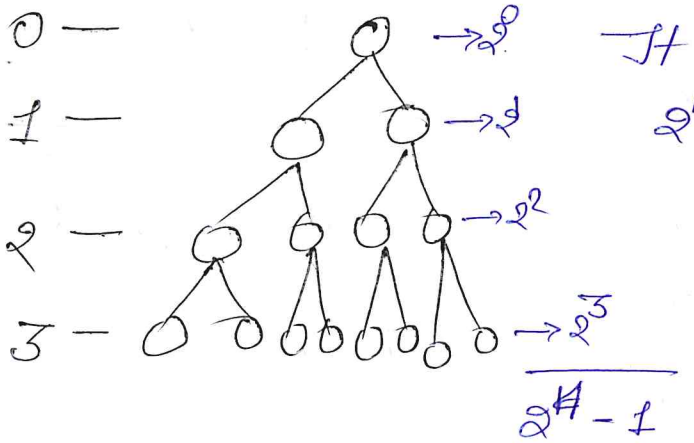
$2^0 + 2^1 + 2^3 \ldots 2^h = 2^{h+1} - 1$

Min. no. of nodes in FBT = $2h+1$

Total no. of

Max no. of nodes in FBT = $2^{h+1} -$

Perfect BT of height h

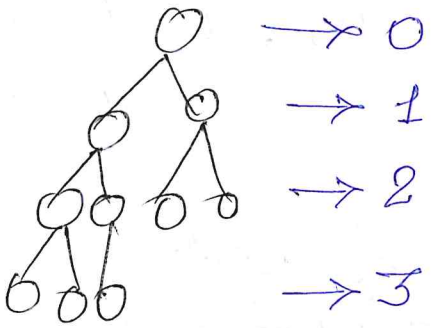


It has to contain exactly $2^{h+1} - 1$ nodes.

(b) l = No. of leaves = 2^h

No. of non-leaf nodes = $2^h - 1$

Complete Binary Tree



n = No. of nodes = 10

No. of internal nodes = $\lfloor 7/2 \rfloor$

= 5

$\lfloor \frac{11}{2} \rfloor = 5$

[43.5] Application: Backtracking for Sudoku

⇒ Application of trees

⇒ Simplest algorithm to solve Sudoku
(NOT the Best But Much more advanced)

⇒ Simplest algo to solve

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

9×9 Sudoku

n×n grid

Task

(1) Fill all the cells such that constraints:

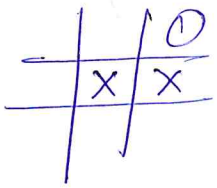
(a) each row should contain 1 to n without repeatation.

(b) each column should contain 1 to n without repeatation.

(c) each subgrid should contain all the no. from 1 to n w/o repeatation

Tic Tac Toe

Constrained Assignment problem



Ques: Given a sudoku problem how do you solve it computationally

⇒ some of cells are not filled. so our job is to fill all the cells

Ex:

2		3
1		
		1

3×3 Grid

Brute force strategy:

e_1, e_2, e_3, e_4, e_5

$e_i = \{1, 2, 3\}$

2	e_1	3
1	e_2	e_3
e_4	e_5	1

Place 1 at loc^n e_1

$e_1 \to 1$, $e_2 \to 1$, $e_3 \to 1$, $e_4 \to 1$

$e_5 \to 1$

(2) $e_1 \to 2$, $e_1 \to 1$ ...

3×3×3×3×3×3

(3) $e_1 \to 3$, e_1

(4) $e_1 \to 4$,

(5) $e_1 \to 5$.

n×n grid, k empty cells

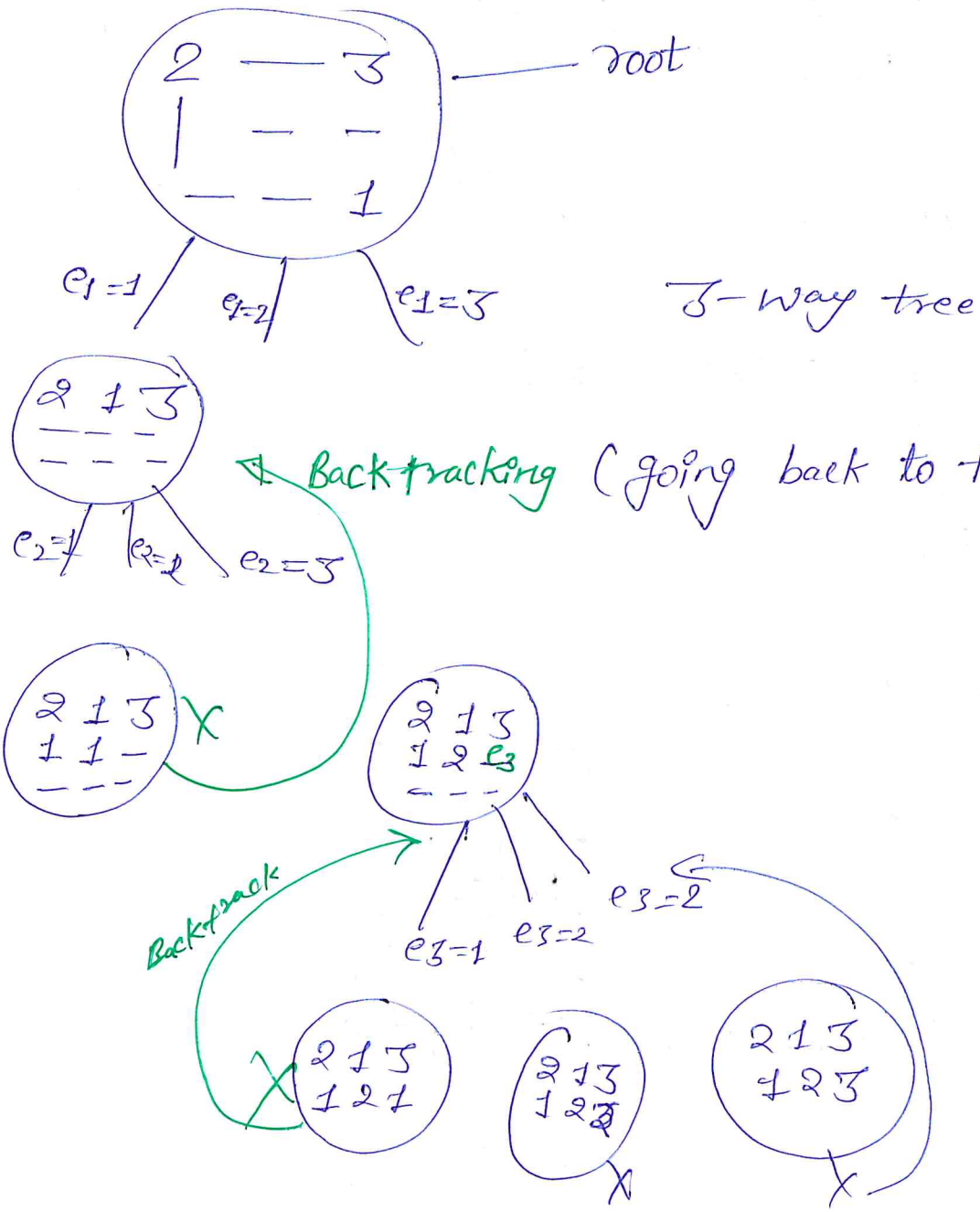
possible assignments = n^k

$k \to \theta(n)$

extremely bad — $\theta(n^n)$ → asymptotic

⇒ In Brute-force method, we force all possible cases.

Back-tracking : (Tree)



① Make a list of all empty cells
② Select an empty cell & place a possible value from 1 to n one after another

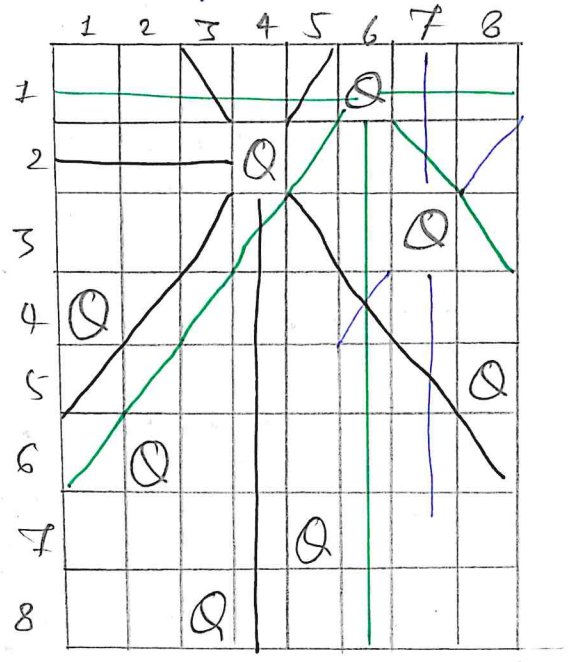
③ Recursively expand as long as you don't hit a dead-end.
   ↳ node is invalid
   ↳ Parent node, where all children are invalid

④ Backtrack wherever you hit a dead end & go back to the parent node & continue.

[43.6] Application: Backtracking for Eight Queens

N-Queens problem: Backtracking

Queen can attack in
→ Same row
→ Same column
→ Same diagonals

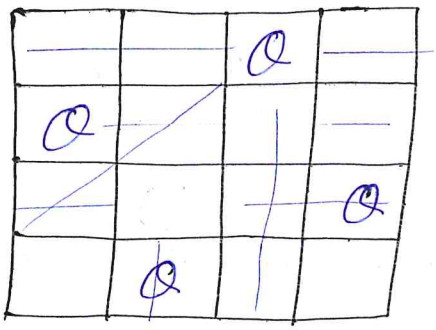


8×8

Puzzle: given an n×n chessboard can be place 8-queens such that no-queen is attacking another.

That is one arrangement of placing queens There are some other arrangements.

Ex:



4×4 Grid

4 Queens

Brute force:

cells = n×n = 4×4
= 16 cells

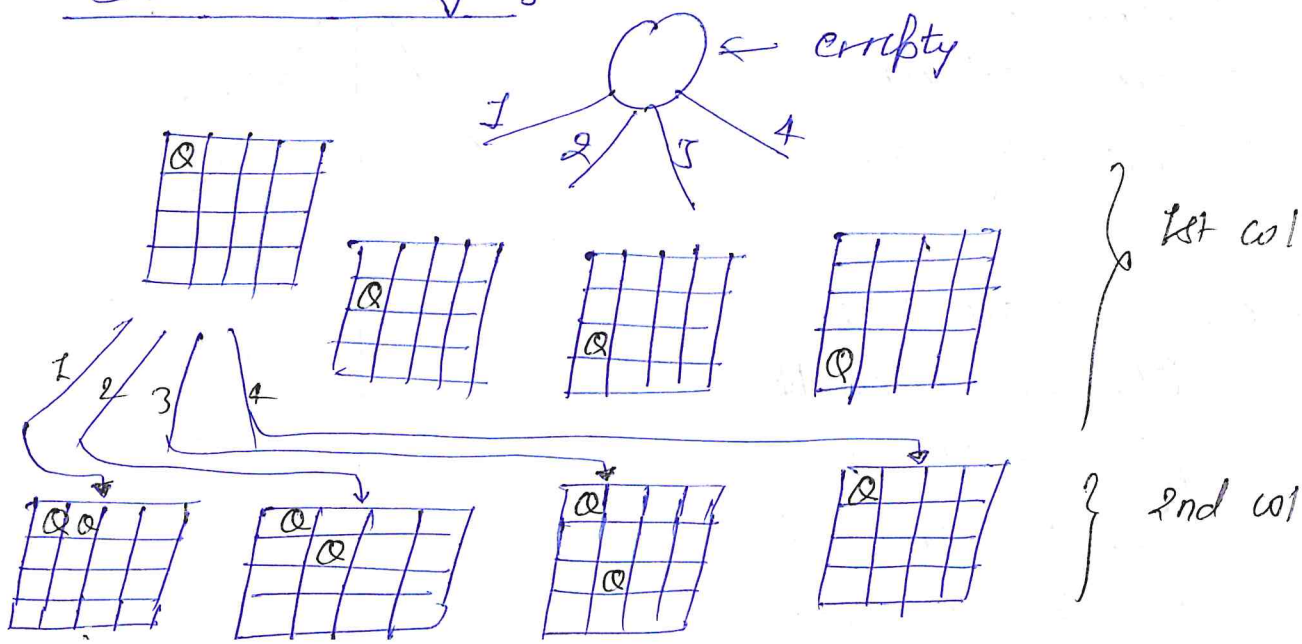
Queens = 4

$^{16}C_4$

No. of ways you can pick 4-cells out of 16 cells.

$^{n\times n}C_n = {}^{n^2}C_n$ combinations / possibilities

Back-tracking:



hit a dead-end → invalid configuration

↓ backtrack

↳ node whose all children are invalid.

① Go column by column from left to right.

② if n-queens are placed (or) all columns return

③ Try to place a queen in each row of the current column (recursively)

↳ if you hit a dead end → invalid node

→ Parent node with all children invalid

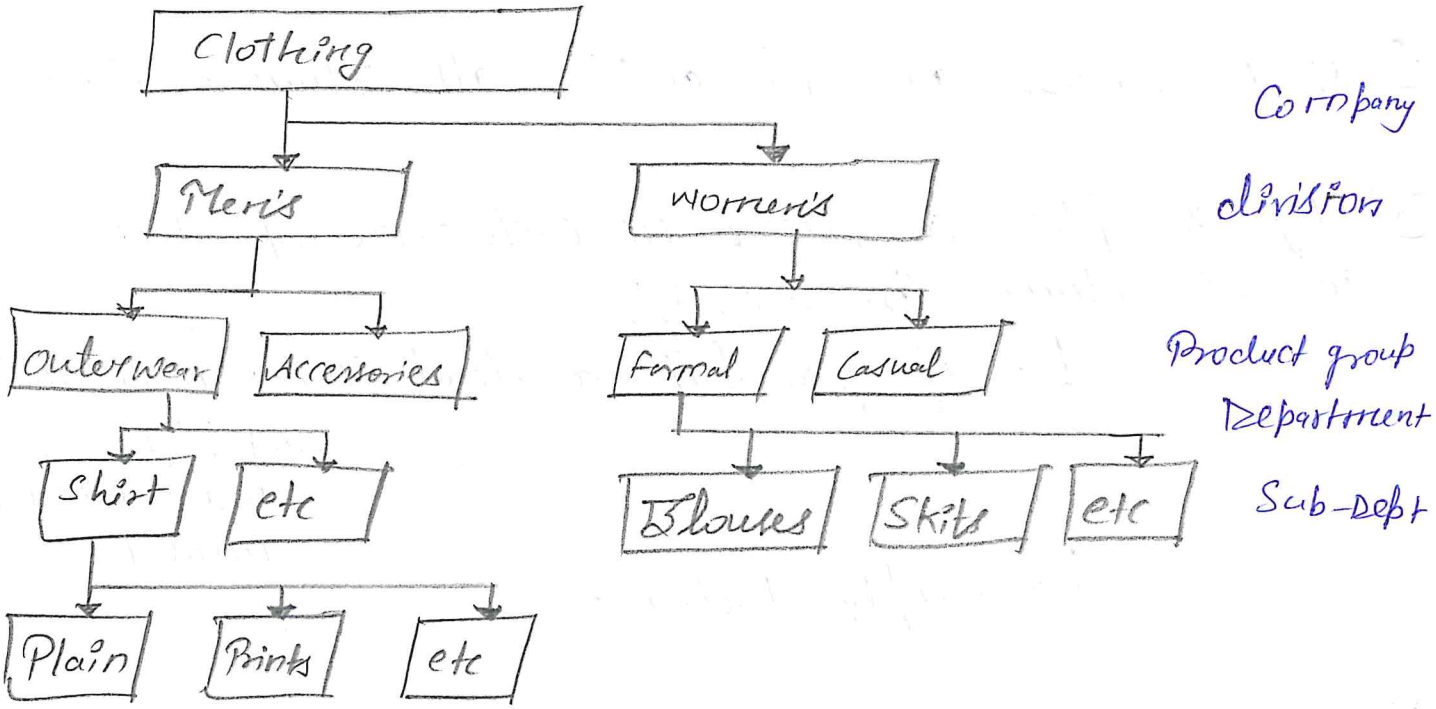
↳ simply backtrack

④ incase you are not able to place n queens in an n x n grid. after trying out all possibilities return impossible.

[43.7] Application of Trees: Hierarchical information websites (DOM)

=> Trees are used widely in tons of applications
ex: expression evalu, backtracking

Retail / E-commerce (Product Hierarchy)

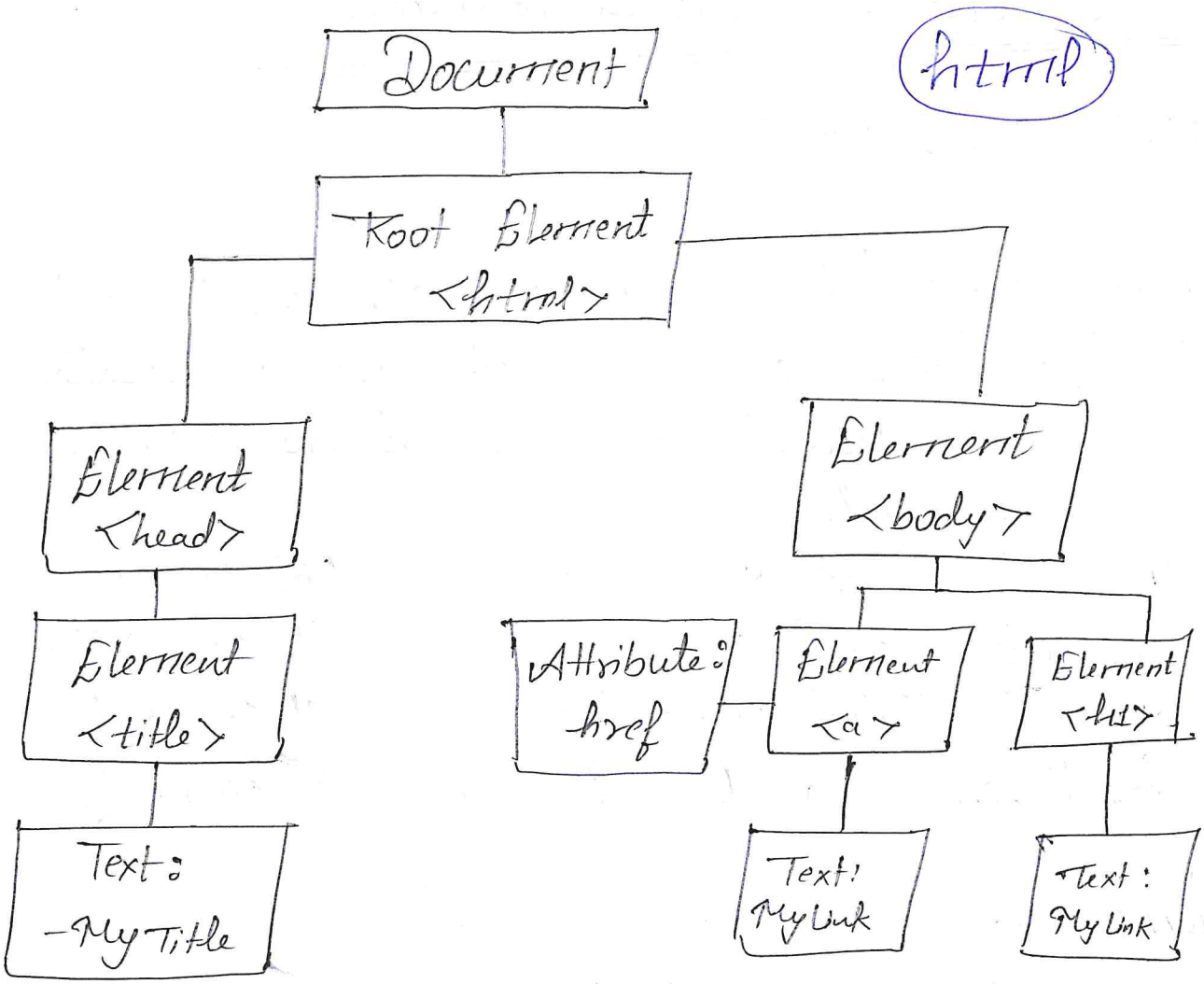


Document object Model (DOM tree)

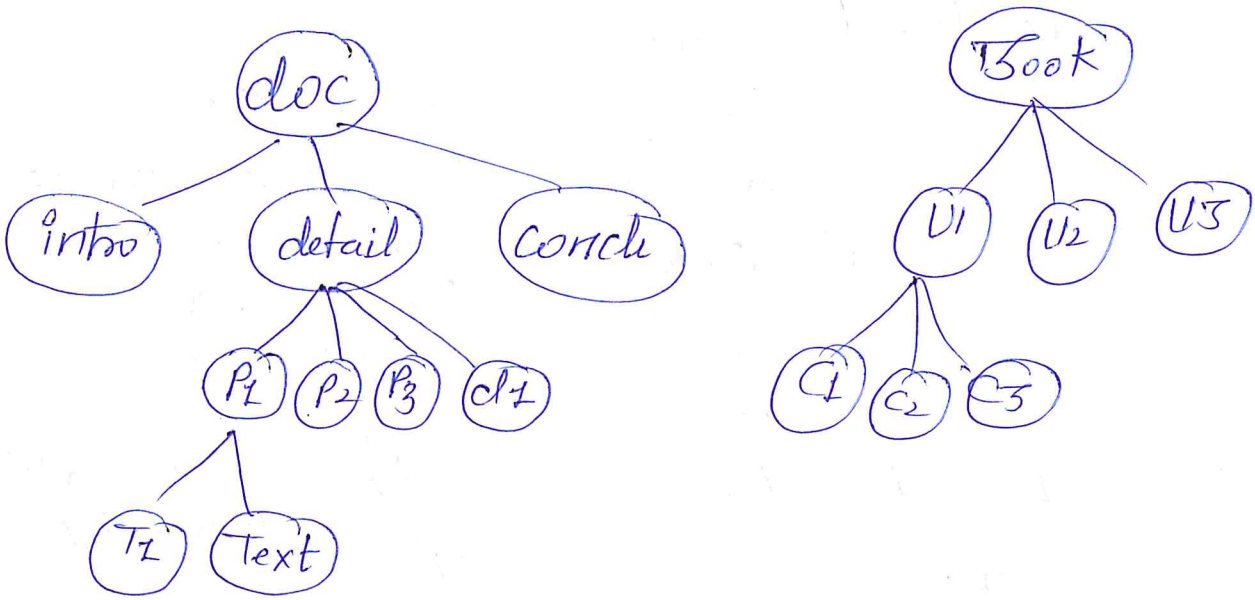
When you visit websites, you actually visit DOM tree.

W3School.com
Wikipedia

Webpages contain HTML
Ajaax
javascript



DOM Tree (Hierarchical Structure)



Chapter ⇒ Solved Problems

[44.1] GATE 2004

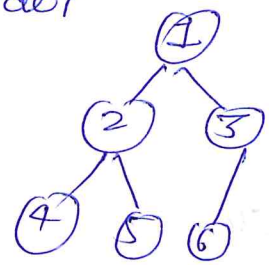
Consider the label sequences obtained by the following pairs of traversals on a labelled binary tree. Which of these pairs identify a tree uniquely.

(i) Pre-order and Post-order
(ii) in-order and Post-order
(iii) Pre-order and in-order
(iv) level order and post-order

(A) (i) only
(B) (ii), (iii)
(C) (iii) only
(D) (iv) only

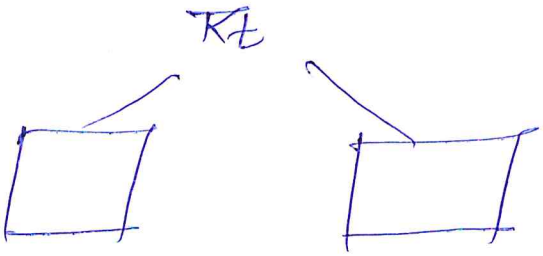
Solution :

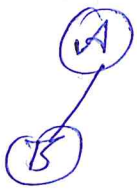
Pre-order → Rt, L, R
Post-order → L, R, Rt
In-order → L, Rt, R
level-order



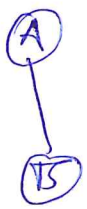
1, 2, 3, 4, 5
first find first level elements and then 2nd level, R then 3rd & so on

Pre	Rt	L	R
Post	L	R	Rt





Pre - AB
Post - BA



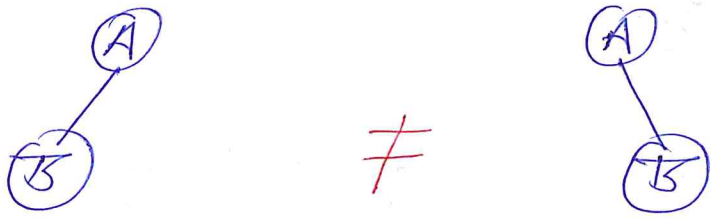
Pre - AB
Post - BA

We can not get unique tree using Pre-Post.

We can get using In+Pre, In+Post.

So option B is correct (i) (iii)

level order & Post order:



level-order: AB = level-order = AB
Post-order: BA Post-order = BA

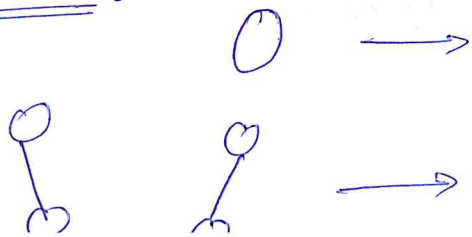
Both the trees are not same but their traversals are same. So this is also not correct.

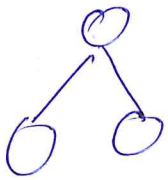
[44.2] GATE 2010

In a Binary tree with n-nodes, every node has an odd no. descendants. Every node is considered to be its own descendants. What is the no. of nodes in the tree that have exactly one child

(A) 0 (B) 1 (C) (n-1)/2 (D) n-1

Solution:



2-children  → 3-decendants

Binary tree → Every node has 0 or 2 children

So ans is 0

GATE 2017 [43]

The pre-order traversal of a binary search tree is given by 12, 8, 6, 2, 7, 9, 10, 16, 15, 19, 17, 20
Then the post order traversal of this tree is

(A) 2, 6, 7, 8, 9, 10, 12, 15, 16, 17, 19, 20

(B) 2, 7, 6, 10, 9, 8, 15, 17, 20, 19, 16, 12

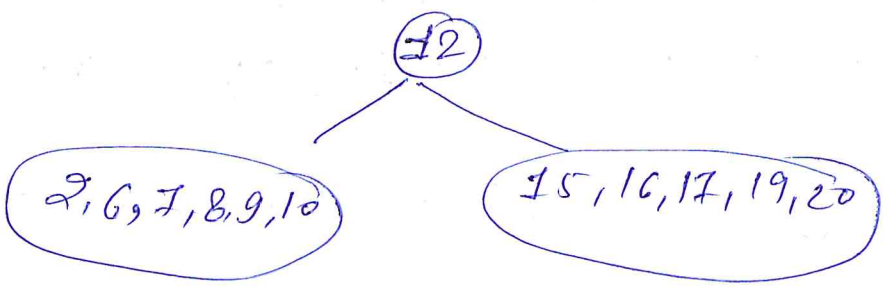
(C) 7, 2, 6, 8, 9, 10, 20, 17, 19, 15, 16, 12

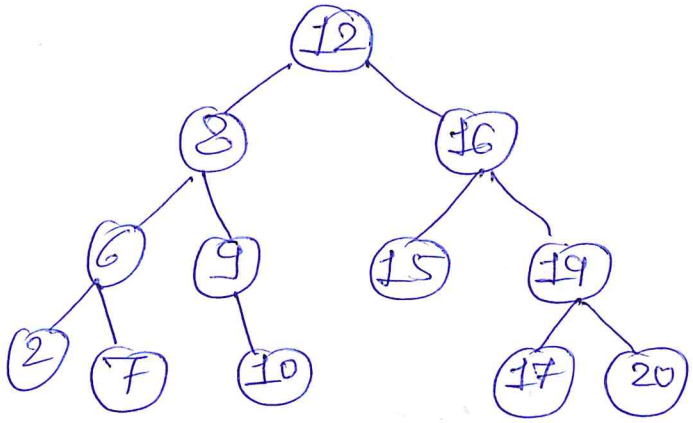
(D) 7, 6, 2, 10, 9, 8, 15, 16, 17, 20, 19, 20

Solution: Pre : 12, 8, 6, 2, 7, 9, 10, 16, 15, 19, 17, 20

In : 2, 6, 7, 8, 9, 10, 12, 15, 16, 17, 19, 20

Start from first in Pre-order traversal
for insertion location check it in In-order





Post order : L, R, Rt

2, 7, 6, 10, 9, 8, 15, 17, 20, 19, 16, 12

[44.4] GATE 2013

The Pre-order traversal sequence of a binary search tree is 30, 20, 10, 15, 25, 23, 39, 35, 42 Which one of the following is the post order traversal sequence of the same tree.

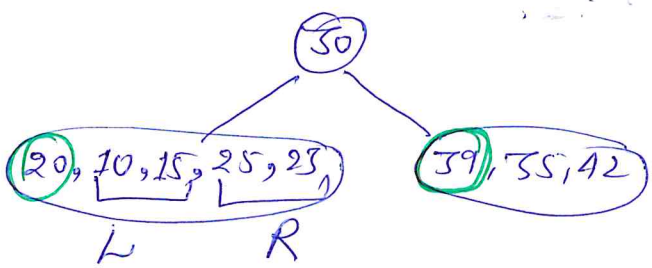
(A) 10, 20, 15, 23, 25, 35, 42, 39, 30

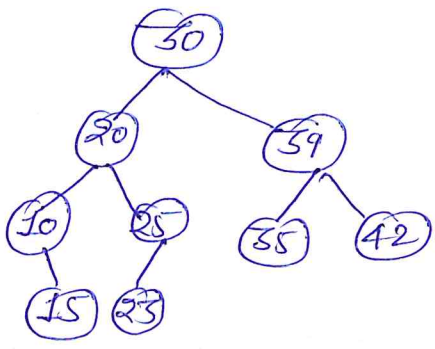
(B) 15, 10, 25, 23, 20, 42, 35, 39, 30

(C) 15, 20, 10, 23, 25, 42, 35, 39, 30

(D) 15, 10, 23, 25, 20, 35, 42, 39, 30

Solution: Pre – 30, 20, 10, 15, 25, 23, 39, 35, 42





Post : 15, 10, 23, 25, 20, 35, 42, 39, 30

Everything is working here, because it is binary search tree.

[44.5] GATE 2006

→ In the binary tree, the no. of internal nodes of degree 1 is 5. And no. of internal nodes of degree 2 is 10. The no. of leaf nodes in the binary tree is.

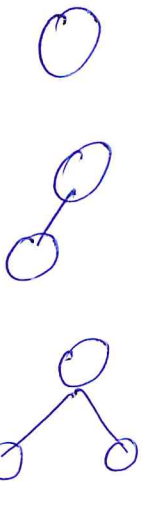
(A) 10 (B) 11 (C) 12 (D) 15

Solution:

Degree of node: No. of other nodes, which are connected to this node.

5 internal nodes — deg 1

10 internal nodes → deg 2



node	Edges
1	0
2	1
3	2

→ In tree for n node (n-1) edges

5 internal nodes of deg 1 → 5 edges

10 internal nodes of deg 2 → 10 × 2 = 20

25 edges

Binary tree can have degree 0, 1, 2

If there are 25 - edge → 25 -

n - nodes → n - 1 edges

26 nodes ← 25 edges

leaf nodes = 26 − 5 − 10 = 11

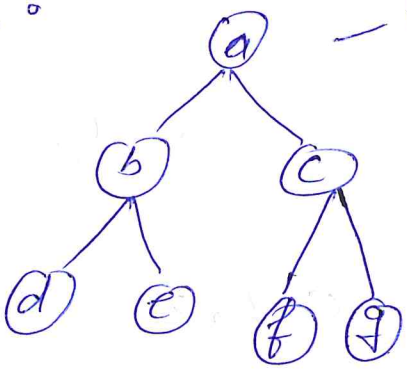
So there are 11 leaf nodes in tree.

[44.6] GATE 2006

A scheme for storing binary trees in an array X is as follows. Indexing of X starts at 1, instead of 0. the root is stored at X[1]. for a node stored at x[i], the left child, if any, is stored in X[2i] and the right child, if any, in X[2i+1]. To be able to store any binary tree on n - vertices the the Min size of x should be

(A) $\log_2 n$ (B) n (C) $2n+1$ (D) $2^n - 1$

Solution :

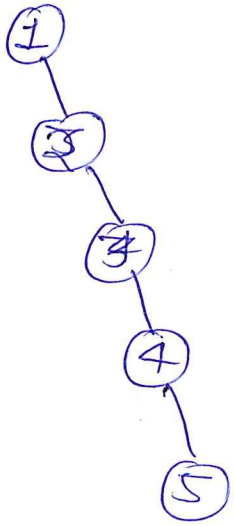


— Perfect Binary tree

1	2	3	4	5	6	7
a	b	c	d	e	f	g

$left = 2 * i$

$right = 2i + 1$



Skewed tree { Array implementation ↓ Wastage

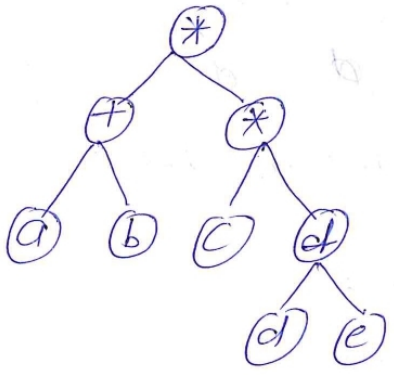
1	–	2								

Only 5 places are utilized, rest are wastage as saving NULL values.

$\Rightarrow 2^n - 1$

Chapter ⇒ Application : Expression Evaluation

[45.1] Postfix to expression tree



d+e

⇒ all leaf nodes are operands
⇒ all internal nodes operators

⇒ We are going Bottom to up

$(a+b) * (c*(d+e))$

⇒ Expression trees are used mostly in compiler.

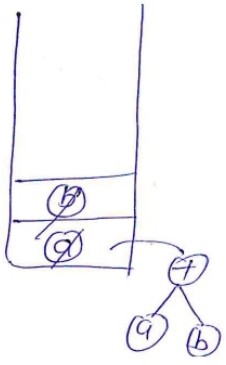
$x = 2*3 + 2/1$ Compiler takes this exp and converts it into machine readable lang

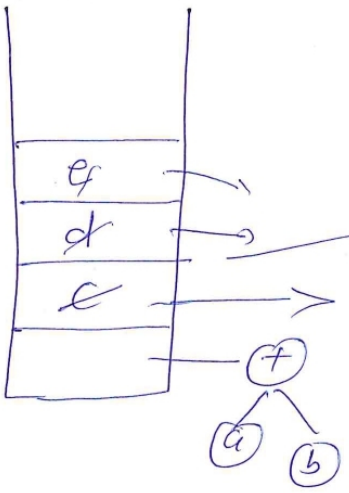
Postfix expression → expression Tree

$ab+cde-**$ We have to use stack for it

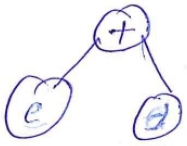
↑ ↑

- If input is operand → Create a node & Push it to stack
- If input is operator → ⓐ Pop top 2 nodes
  ⓑ build a new tree using the operator & operand
  ⓒ Push new tree into stack



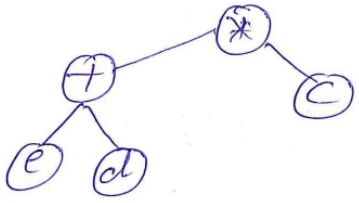


When get +

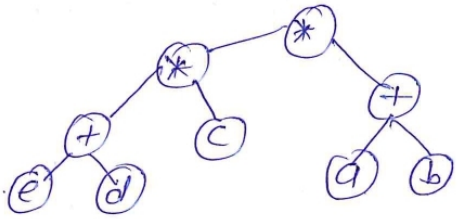


Store tree in stack

When get *



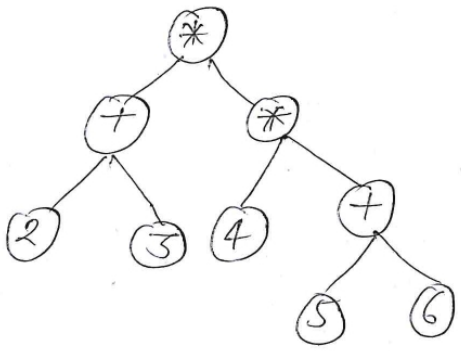
When get * (last)



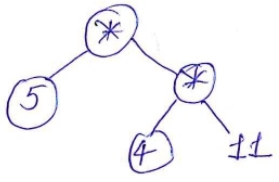
(5) → If you reach at the end of input
Pop the stack & return the resultant tree

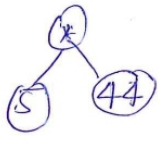
return whole tree root pointer

[45.2] Evaluating an Expression tree



LST * RST
↓
value ?
again get operator do recursively





44*5 = 220

eval (node) eval (root)

```
If node is operand
    return operand
else // node is operator
    l = eval (node.left)
    r = eval (node.right)
    return (l operator r)
```

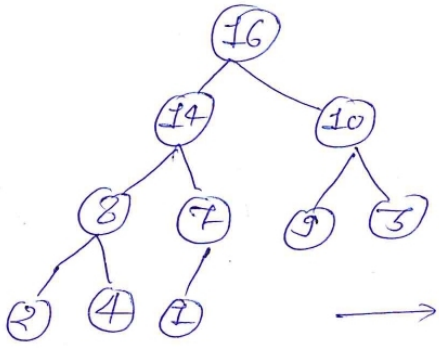
Chapter ⇒ Heap Sort

[46.1] Heap: What and Why

⇒ New data structure : Heap → Priority Queue
Sorting → heap sort

What is a heap ?

Max-heap – Complete Binary Tree, where every Parent value is greater than its children



→ In last level we fill from left to right.

	1	2	3	4	5	6	7	8	9	10
A:	16	14	10	8	7	9	3	2	4	1

No Null place

Min-heap : Every Parent node value is less than equal to child node.

Why do we need another DS.

This DS looks like Binary search tree

If I want to find max element very fast

Root of tree in Max-heap

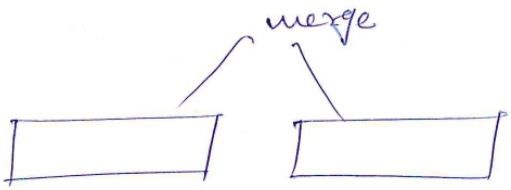
$O(1)$ time — In TST → $O(\log n)$

$O(n)$ wc

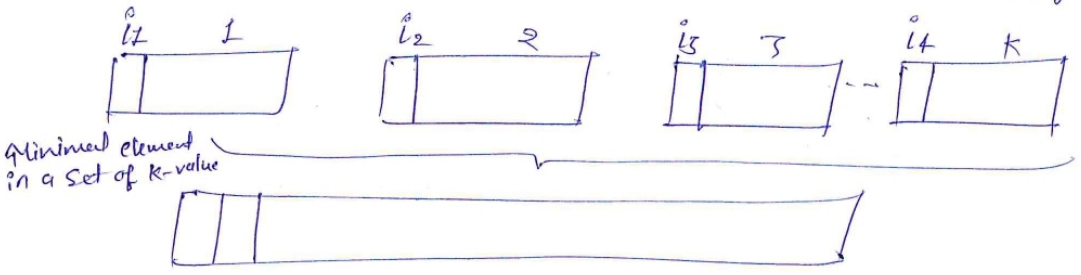
If I want to find min element very fast

Root of min heap

merge Sort :



2-Way merge

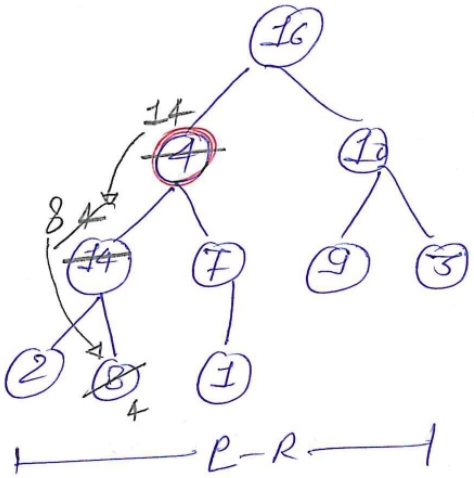


In this example, we can use Min-heap

Heapify [46.2]

Max-heap

Task:
node s.t.
LST is heap
RST is heap



but node does not satisfy the heap property

FIX IT & convert whole tree into heap.

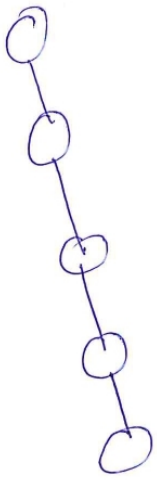
Max-heapify (A, 2)
A.length = 10

Compare 4 with (14, 7)
Pick largest value
Swap (4, 14)

Max heapify time complexity
when tree has n-nodes
$h = O(\log n)$
$T.C = O(\log n)$

again do it recursively
Compare 4 with 2, 8
take 8 & swap

Max no. of comparisions & swaps = to max-heapify = $O(\log n)$



heap: n - nodes CBT
depth = O(log n)

↳ That's why we designed heap & use sorting algorithm i.e. heap sort.

$n \rightarrow O(n)$

A: Array representation

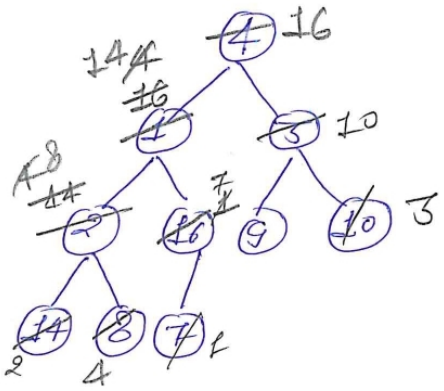
```
Max_heapify (A, i)
  l = left (i)
  r = right (i)
if l <= heapsize (A) AND A[l] > A[i]
      largest = l  else  largest = i

if r <= heapsize (A)  AND A[r] > A[largest]
      largest = r

If largest ≠ i
   swap A[i] & A[largest]
   Max_heapify (A, largest)
```

[463] Build a Heap

An Array A is given as

A: 4, 1, 3, 2, 16, 9, 10, 14, 8, 7

Build a heap from the given array elements



(1) n-nodes CBT

1 to $\lfloor \frac{n}{2} \rfloor$: internal nodes

$\lfloor \frac{n}{2} \rfloor + 1$ to n : leaf nodes

CBT with n = 10

$\lfloor \frac{n}{2} \rfloor = 5$

(2) Build_Max-heap(A)

for $i = \lfloor \frac{n}{2} \rfloor$ to 1

max_heapify(A, i) n = A.length

Comp (7, 16) no Swap

Comp 2 with (14, 8) place largest @ 2 Swap (2, 14)

Comp 3 with (9, 10) Place largest @ node Swap (3, 10)

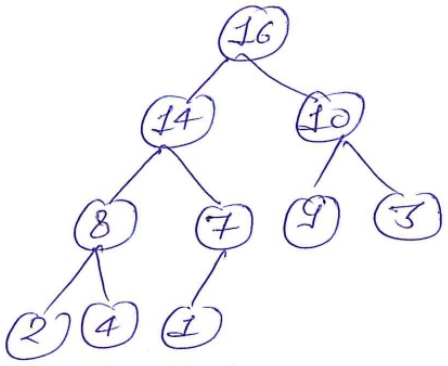
Comp 1 with (14, 16) Place largest at Place 1' Swap (16, 1)

Comp. 4 with (16, 10) → Swap (4, 16)

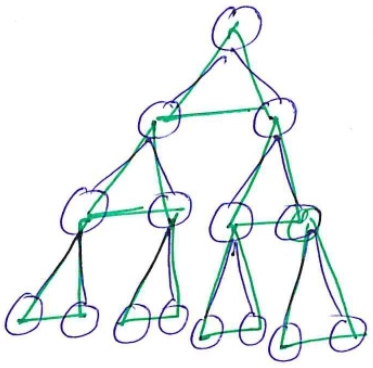
Recursively do Comp 4 with (14, 1) → Swap (4, 14)

Comp. 4 with (2, 8) → Swap (4, 8)

After doing max-heapify



[46.4] Time complexity of Build_max_heap



$0 \rightarrow 2^0 \rightarrow 3 \Rightarrow h-k=3$

$1 \rightarrow 2^1 \rightarrow 2 \Rightarrow h-k=2$

$2 \rightarrow 2^2 \rightarrow 1 \Rightarrow h-k=1$

$3 \rightarrow 2^3 \rightarrow 0 \Rightarrow h-k=0$

$h = \lfloor \log_2 n \rfloor$

find_max_3_elem $\rightarrow O(1)$

total time complexity $= (2^3 * 0) + (2^2 * 1) + (2^1 * 2) + (2^0 * 3)$

Worst case : No. of times I am going to call find_max_3_elem

$$\sum_{i=1}^{h} 2^{h-i} * i \qquad h = \lfloor \log_2 n \rfloor$$

$$2^h = O(n)$$

$$2^h \left\{ \sum_{i=1}^{h} \frac{i}{2^i} \right\}$$

$$\lim_{h \to \infty} \frac{h}{2^h} = 0$$

$$= 2^h \left\{ \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \cdots \frac{h}{2^h} \right\}$$

$$\leq 2^h \left\{ \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \cdots \infty \right\} \Rightarrow 2$$

Geometric series

$$\left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots\right) + \left(\frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} \cdots\right)$$

$$+ \left(\frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \cdots\right) + \left(\frac{1}{16} + \frac{1}{32} + \cdots\right) + (\cdots)$$

$$a + ar + ar^2 + ar^3 + \cdots \infty$$

$$\downarrow$$

$$1/2 \qquad \frac{a}{1-r} = \frac{1/2}{1-1/2} = 1 \qquad a = 1/4, \; r = 1/2$$

$$= 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} \cdots \infty \qquad \frac{1/4}{1/2} = 1/2$$

$$= \frac{1}{1-1/2} = \frac{1}{1/2} = 2$$

Time complexity to build a heap = $O(n)$

[46.5] Heap Sort

⇒ use a heap to sort an array

⇒ unordered array : A , $n = 10$

↓

visualize array

as CBT → ~~heap~~

Depth $\log_2(n)$

⇒ Build_max-heap(A) → heap $\Theta(n)$

⇒ ② Sort the array A

n = effective size of my heap

③ Swap A[i] with A[n]

$n = n-1$ 9 elements

max_heapify (A, 1)

} $\Theta(\log n)$

④ Repeat till $n = 0$

total time complexity = $n + n \log n$

$= \Theta(n \log n)$

[46.6] Time & Space Complexity of Heap Sort

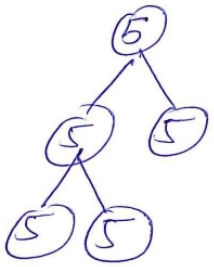
⇒ heapsort → $O(n \log n)$ → Best, Worst, avg. case

One Case: $O(n)$: Best

↳ all elements are same

{ 5 (a, b, c) → $O(1)$

$O(n)$



Space Complexity

A - Input

↳ Max - heapify
build max_heap

$O(1)$

[46.7] Priority queues: Applications of Heaps

⇒ One DS is known as priority queue
- It is implemented binary - heap.

Collection or set of similar types of values

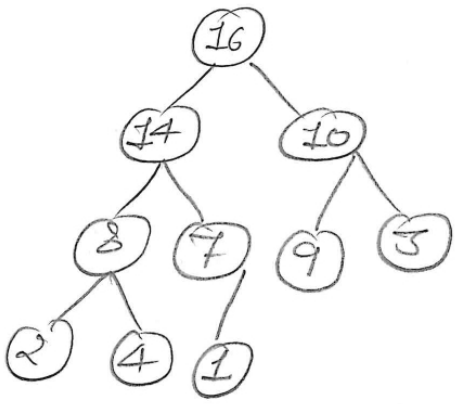
operations: (i) Max(S): return the max value in S

(ii) Extract_max(S): returns the max-value deletes this value from S.

(iii) insert (S, x) : insert x into S.

(iv) increase-key (S, x, k) : $k \geq x$

increase value x to k



(i) return root value

(ii) ⓐ $H = 10$;
$A[1] = A[H]$

ⓑ $H = H - 1 = 9$

ⓒ max_heapify (A, 1)

$\Theta(h) \downarrow \Theta(\log H)$

(iii) insert (S, 15)

$A[n] = x$

$n = 9$ (heap size)
$H = H + 1 = 10$

$\Theta(\log H)$

(iv) incr_key (S, 3, 16)

$\Theta(\log H)$

You might have gone from root to leaf node in worst case

Priority Queue →

Graph algorithm
operating system (Priority Scheduling)
k-way merge

[46.10] Solved Problem GATE 2005

Suppose there are $\lceil \log n \rceil$ sorted lists of $\lfloor n / \log n \rfloor$ elements each. The time complexity of producing a sorted list of all these elements is (Hint: use heap data structure)

(A) $O(n \log\log n)$

(B) $\Theta(n \log n)$

(C) $\Omega(n \log n)$

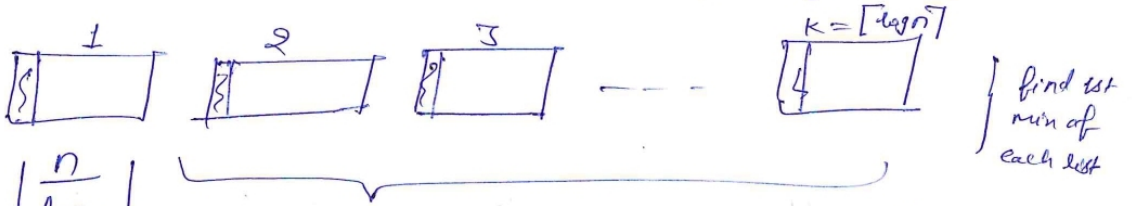
(D) $\Omega(n^{3/2})$

Solution

heap → min

↓

app^n → k-way merge



k-way merge → K → min heap → $\Theta(K)$

(1) extract min

(2) insert a new element

$\Theta(\log K)$

$$\left\lfloor \frac{n}{\log n} \right\rfloor \lceil \log n \rceil = \Theta(n \log K)$$

$$= \Theta(n \log \log n)$$

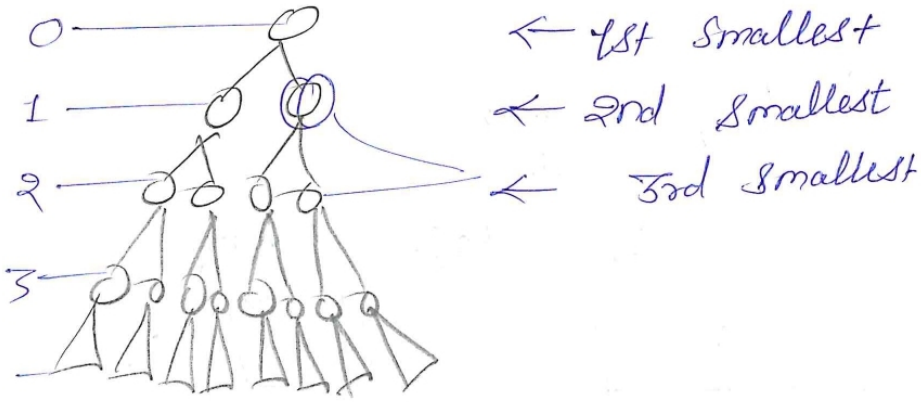
[46.11] Solved Problem GATE 2003

In a heap with n-elements with the smallest element at the root, the 7th smallest element can be found in time

(A) $\Theta(n \log n)$ (B) $\Theta(n)$

(C) $\Theta(\log n)$ (D) $\Theta(1)$

Solution: Min-heap



at depth 6, you can get 7th smallest element

$0 \rightarrow 1 = 2^0$

$1 \rightarrow 2 = 2^1$

$2 \rightarrow 4 = 2^2$

$\vdots$

$6 \rightarrow 2^6$

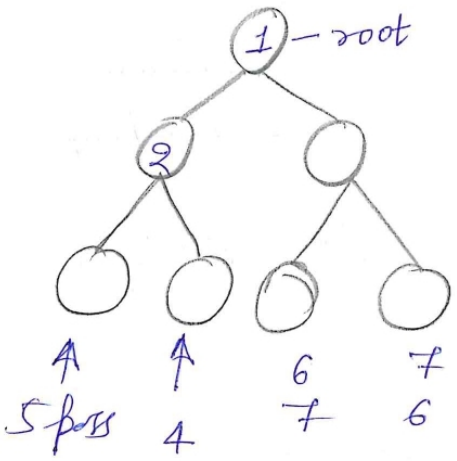
$2^7 - 1 = 127$

7th smallest element will be one of these

[46.12] Solved Problem GATE 2018

The no. of possible min-heaps containing each value from {1, 2, 3, 4, 5, 6, 7} exactly once is ___

Solution :



possibilities :

{3, 4, 5, 6, 7}

2 * 5 * 4 * 1

= 80

[46.13] Solved Problem GATE 2016

An operator delete(i) for a binary heap data structure is to be designed to delete the item in the ith node. Assume that the heap is implemented in an array and i refers to the ith index of the array. If the heap tree has depth d (no. of edges on the path from the root to the farthest leaf) then what is the time complexity to re-fix the heap efficiently, after the removal of the element.

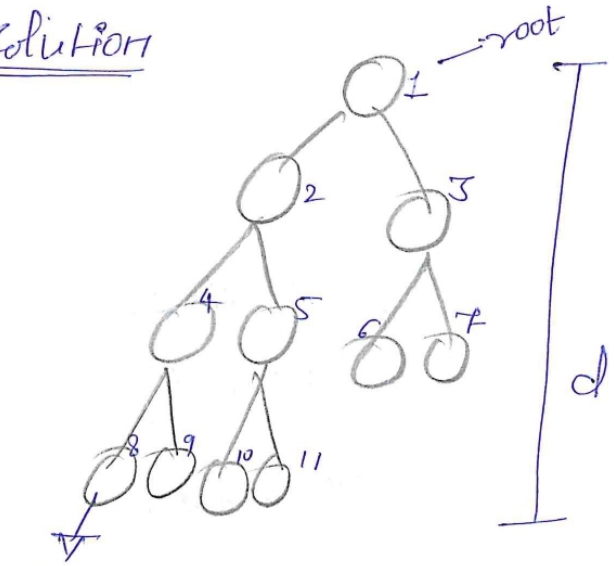
(A) $O(1)$

(B) $O(d)$ but not $O(1)$

(C) $O(2d)$ but not $O(d)$

(D) $O(d\,2d)$ but not $O(2d)$

Solution



delete (i)

⇓

delete (4)

↳ delete

↳ maxheapify

$O(d)$ $O(\log n)$

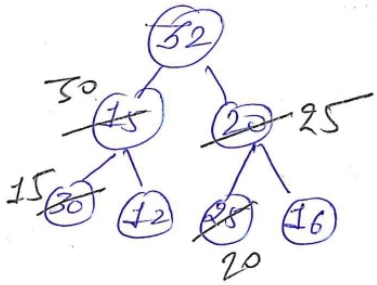
$O(1)$

If I am deleting leaf node

[46.15] Solved Problem GATE 2004-2

The elements 32, 15, 20, 30, 12, 25, 16 are inserted one by one in the given order into a Max-heap. The resultant max-heap —

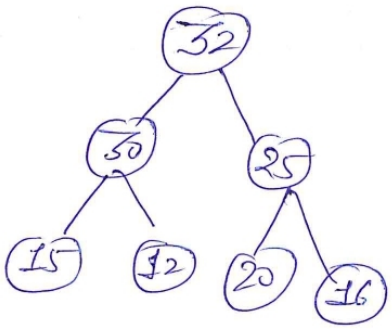
root > child



comp 15 with (30,12)

Pick largest

Swap (15, 30)



option (a) is correct.

[46.16] Solved Problem GATE 2014

A priority queue is implemented as a max-heap. Initially, it has 5 elements. The level order traversal of the heap is: 10, 8, 5, 3, 2. Two new elements 1 and 7 are inserted into the heap in that order. The level order traversal of the heap after the insertion of the elements is:

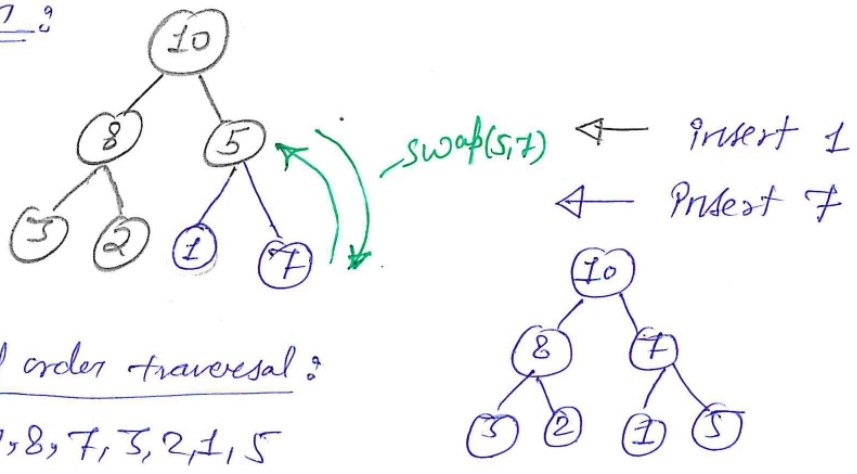
(A) 10, 8, 7, 3, 2, 1, 5 ✓

(B) 10, 8, 7, 2, 3, 1, 5

(C) 10, 8, 7, 1, 2, 3, 5

(D) 10, 8, 7, 5, 3, 2, 1

Solution:



level order traversal:

10, 8, 7, 3, 2, 1, 5

Chapter ⇒ Balanced Trees : AVL Trees

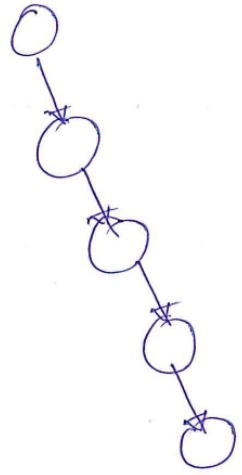
[48.1] AVL Trees : What and Why

(Adelson – velsky – Landis)

⇒ Binary Search trees : Major Problem

↓

Worst case $O(n)$



LL

extremely skewed

Why AVL tree ?

Because objective of AVL tree is to fix the worst case problem when tree becomes like linked-list

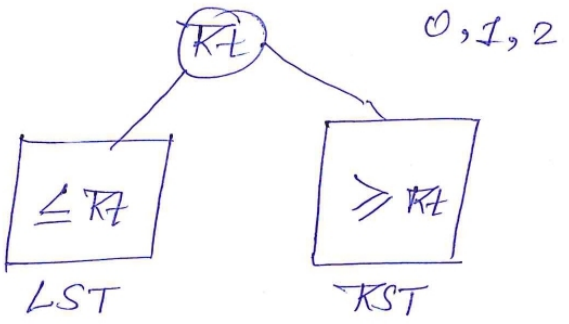
⇒ AVL trees are the idea of solving this worst case problem

insert / delete / search ⟶ $O(\log n)$

⇒ Red-Black Trees

AVL Tree : Self-balancing binary-search Trees

Balance (n)



0, 1, 2

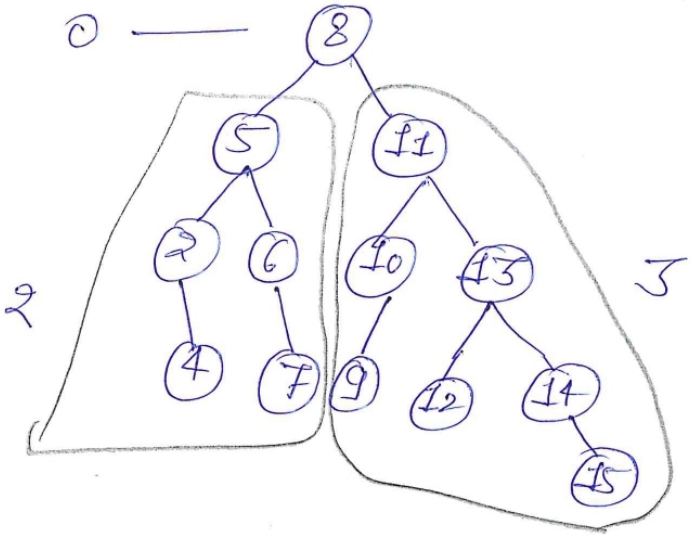
$$\text{Balance}(n) = \text{height}(\text{LST of } n) - \text{height}(\text{RST of } n)$$

Binary Search Tree :

every node :

LST ≤ nodes

RST ≥ nodes



Balance (8) = 2 - 3 = -1

Balance (5) = 2 - 2 = 0

balance (14) = 0 - 1
= -1

AVL Tree : Binary Search Tree

AND

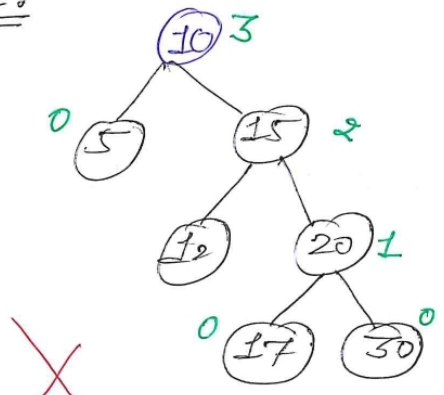
every node's balance should be -1, 0, 1

balance(n) ∈ {-1, 0, 1} for all nodes n in T

⇒ height of a Null tree = -1

⇒ height of a single node tree = 0

Case1:

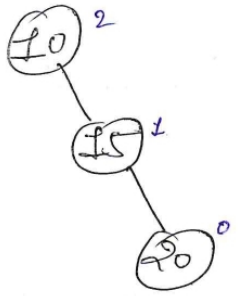


for every node height of the tree rooted at that node

Balance (10) = 0 - 2
= -2

✗ Not an AVL tree

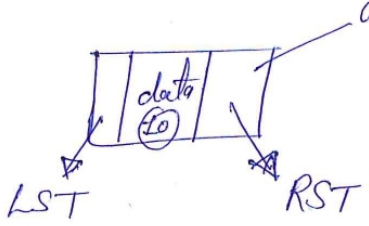
Case2:



Balance (10) = -1 - 1
= -2

✗ Not an AVL tree

node of an AVL

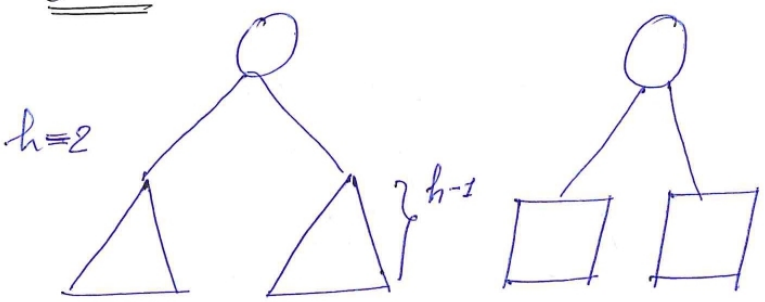


additional field (compute balance very fast)

height of the tree rooted here.

[48.2] Height of an AVL tree & Searching

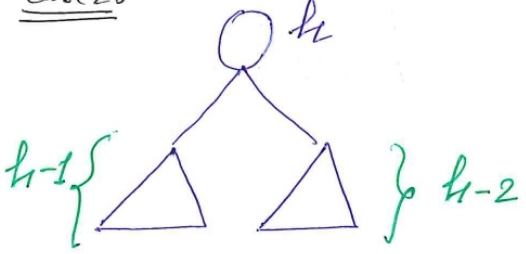
Case 1:



Balance $\in \{-1, 0, 1\}$

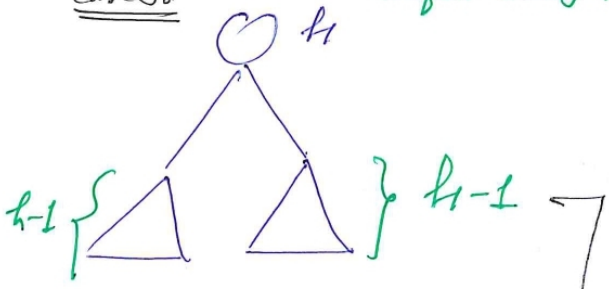
$N(h)$ = no. of nodes in an AVL Tree

Case 2:



Case 3:

Perfect Binary Tree



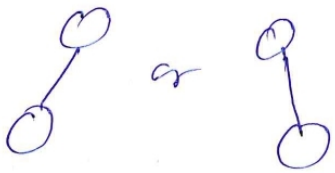
$$N(h) = N(h-1) + N(h-2) + 1$$

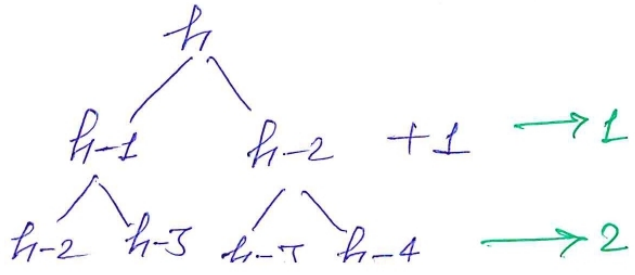
$$N(h) = 2N(h-1)$$

Boundary Case:

$N(0) = 1$

$N(1) = 2$





$$2^0 + 2^1 + 2^2 + 2^3 + \cdots\cdots 2^{h-1}$$

$\boxed{2^h - 1 = n(h)}$ no. of nodes in AVL tree of height h

$h = \Theta(\log n)$

h — height; n — nodes

$n(h) = n(h-1) + n(h-2) + 1$

Fib(n) = fib(n-1) + Fib(n-2)

Fib(0) = 0
Fib(1) = 1

base case
$n(0) = 1$
$n(1) = 2$

AVL n-nodes

$$\log_2(n+1) \leq h \leq c \cdot \log_2(n+2) + b$$

$c \simeq 1.44$, $b \simeq -0.328$

$$h \leq 1.44 \log_2(n+2)$$

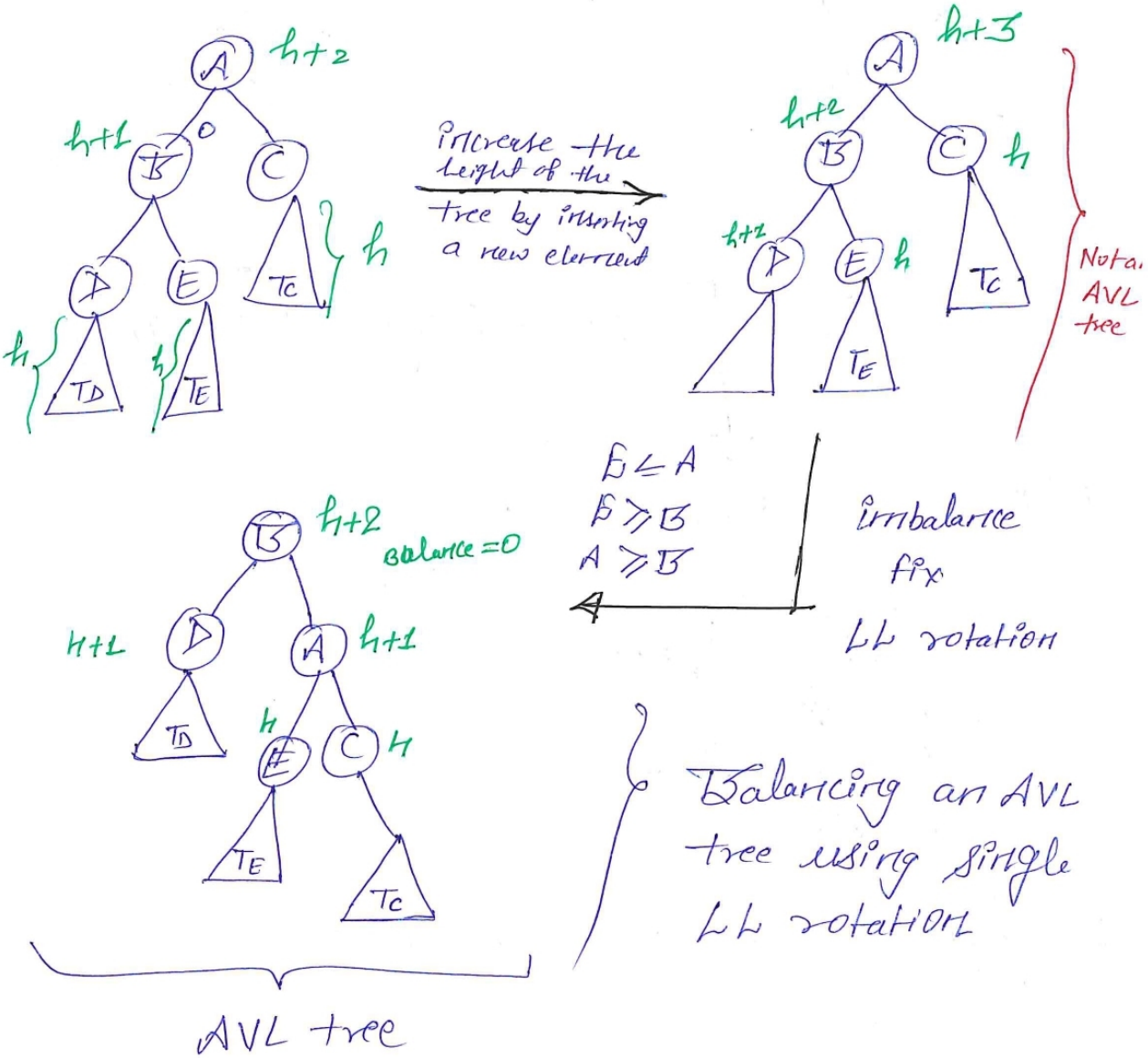
Searching an AVL is same as searching in BST

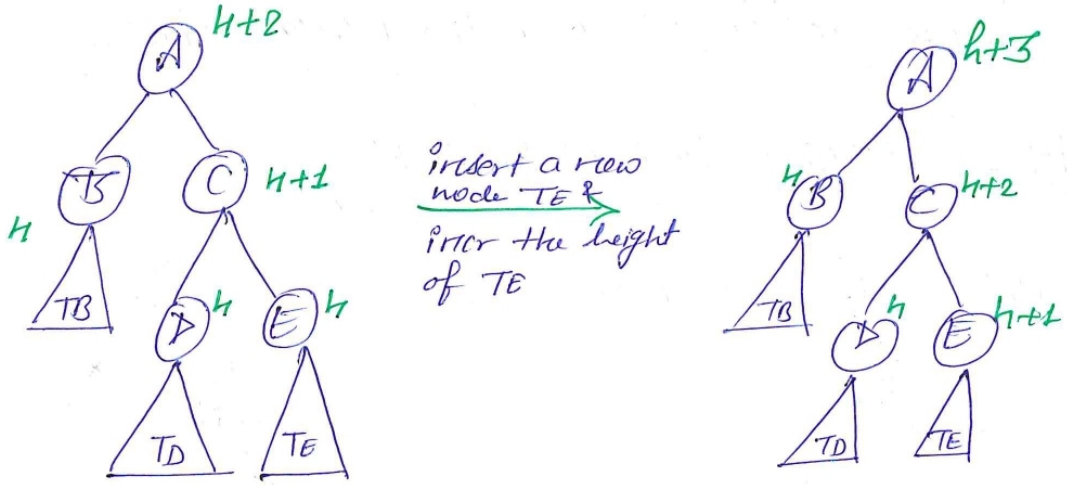
$\Rightarrow \Theta(\log_2 n)$

Chapter ⇒ Balancing a Tree using rotation

[49.1] Single rotation : LL, RR

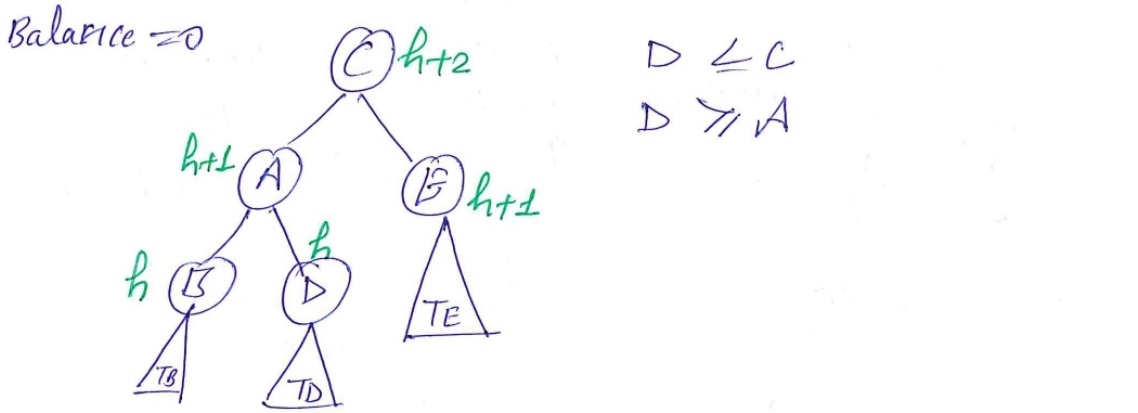
⇒ AVL : Self - balancing BST





Not an AVL tree

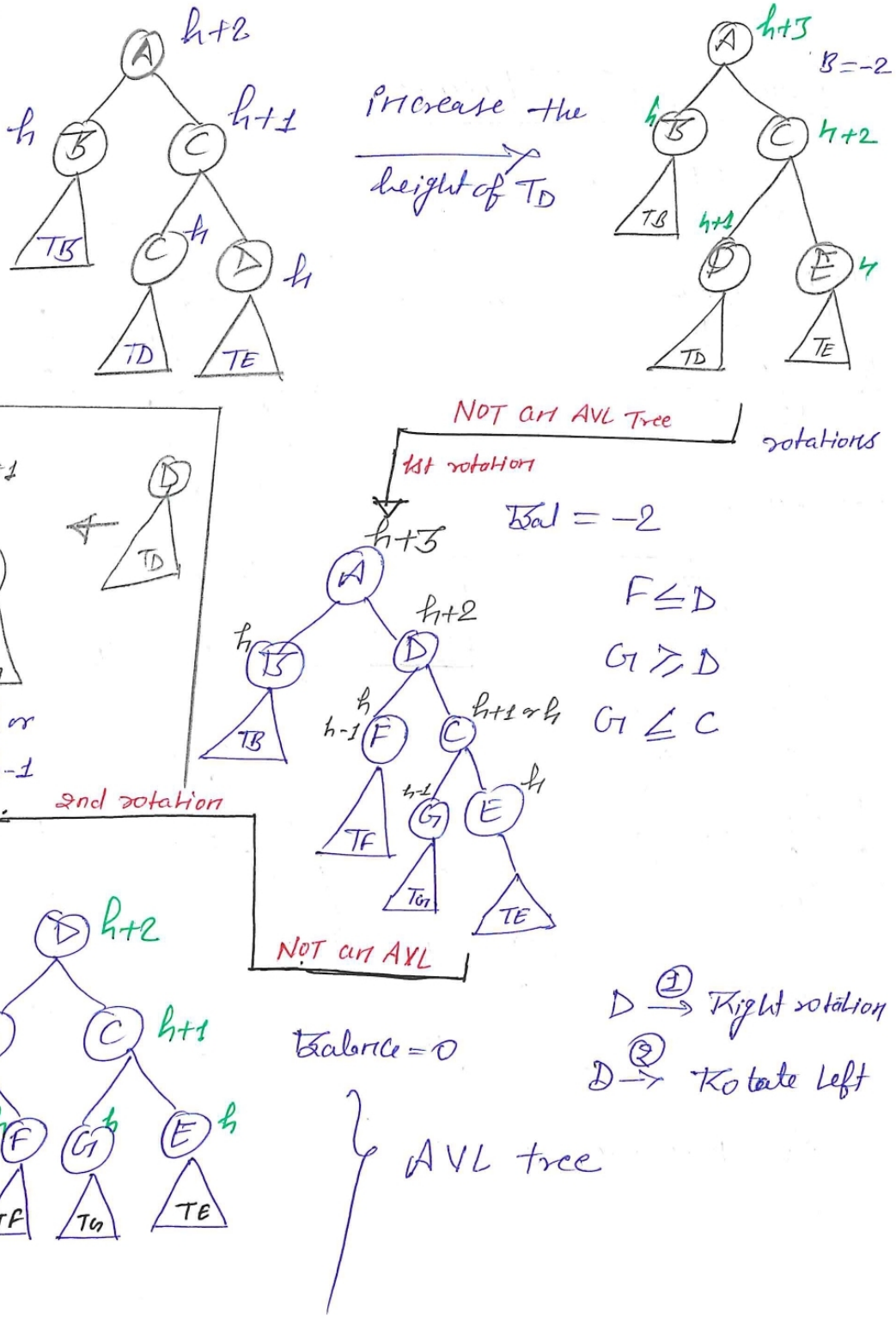
fix it [Single rotation (RR)]



⇒ These rotations are non-trivial to write codes

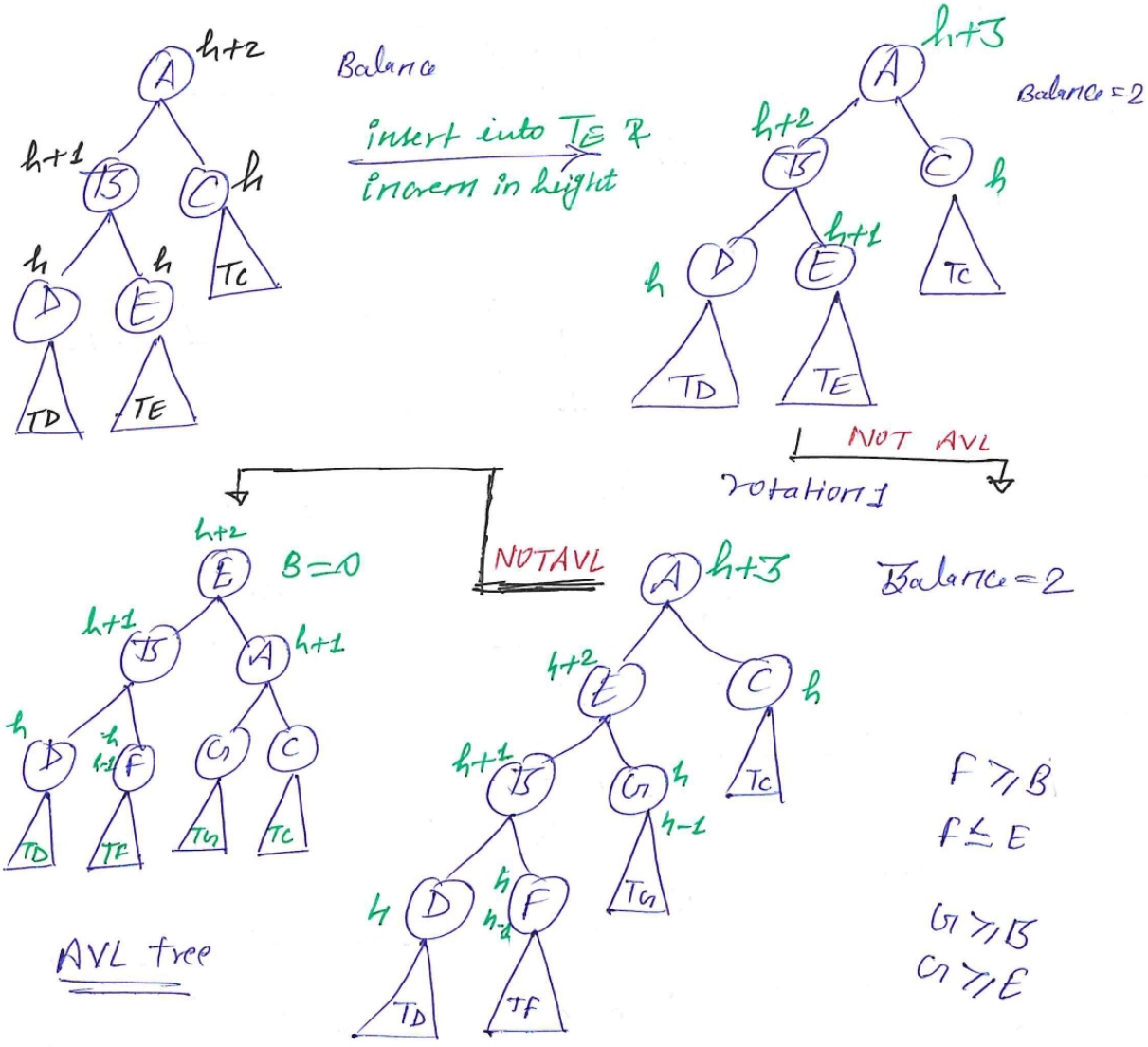
Chapter ⇒ Double Rotation

[50.1] RL Rotation



⇒ For writing code is extremely error-prone. (error-prone)
You have to be careful when writing code.

[50.2] LR Rotation



Self balancing (BST)

Single rotation

Chapter ⇒ Insertion with example

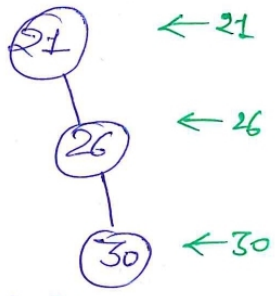
[51.1] Insertion with example

AVL tree (Build using insertions): Single/double

21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7

AVL : BST + balanced

(a)

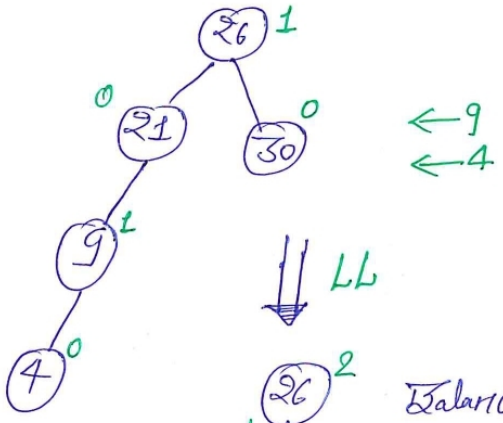


← 21
← 26
← 30

(i) insert using BST proce
(ii) update the height from the new node to root
(iii) check for imbalance
If imbalance balance using rotations

RR ↓

(b)



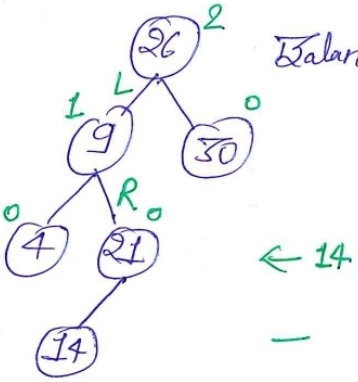
← 9
← 4

RR: Take the middle node, rotate it left & Pull it out.

LL ↓

LL: Middle node, right rotation, Pull it out

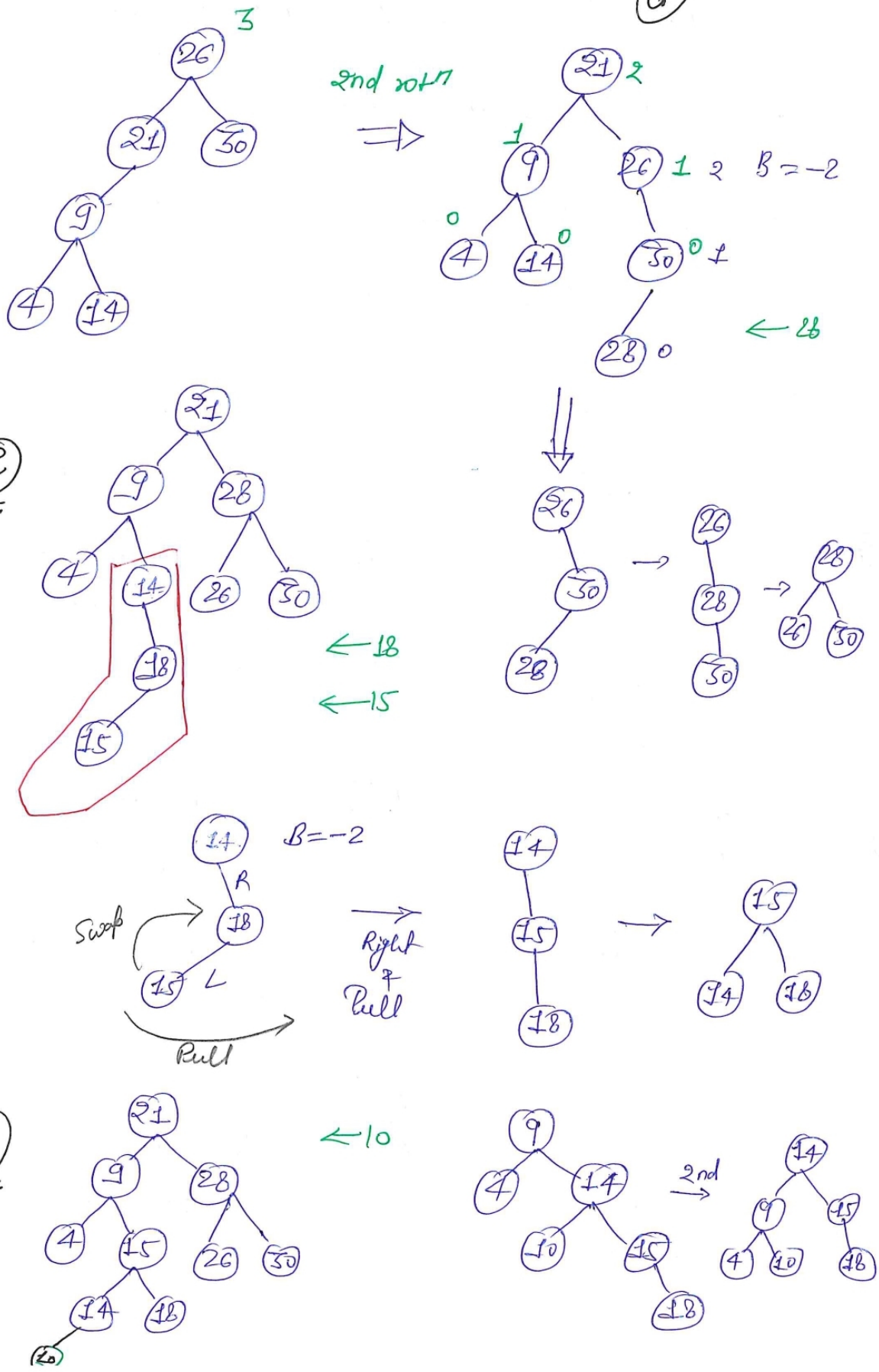
(c)

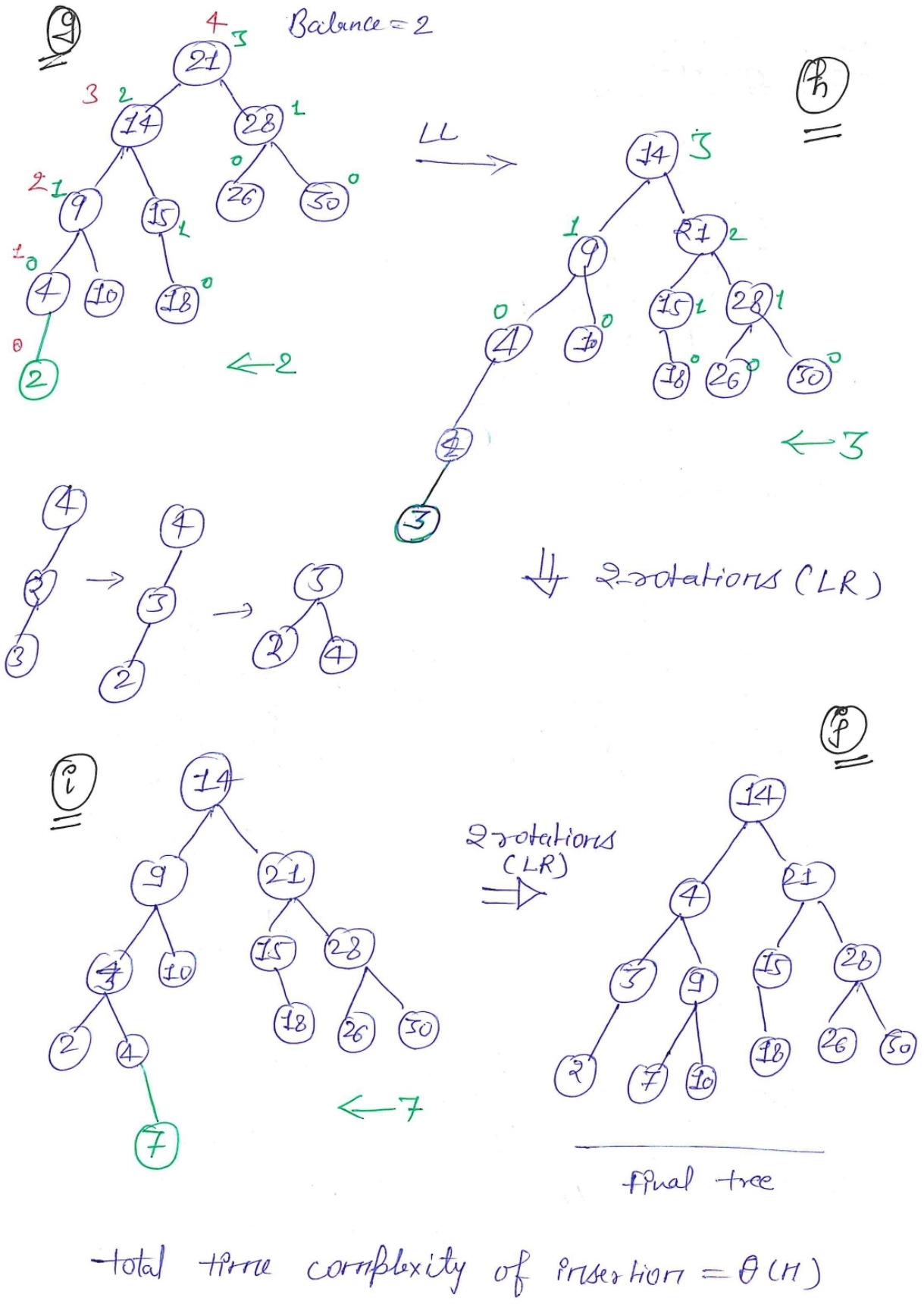


Balance = 2

← 14

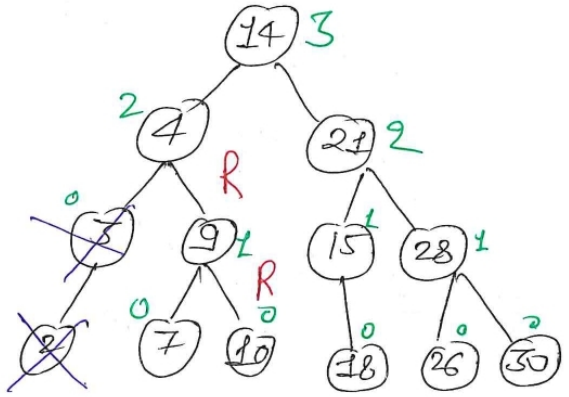
LR: 2 - rotations
node at the end of L & R
1st rotation - Left & Pull


d
3
26
21
30
9
4
14
2nd rotn
21
2
1
9
26
1
2
B = -2
0
4
14
0
30
0
1
28
0
← 28
e
21
9
28
4
14
26
30
18
15
← 18
← 15
26
30
28
26
28
30
28
26
30
14
B = -2
R
Swap
18
15
L
Right
+
Rull
Rull
14
15
18
15
14
18
f
21
9
28
4
15
26
30
14
18
16
← 10
9
4
14
10
15
18
2nd
14
9
15
4
10
18



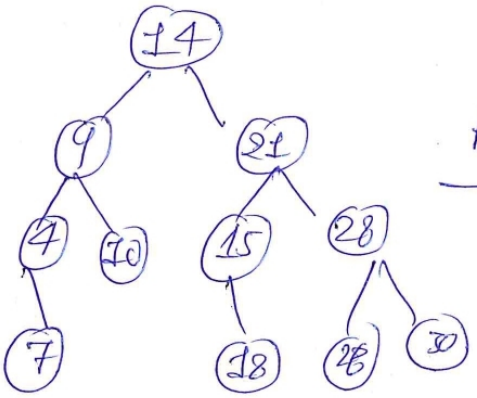
total time complexity of insertion = $\theta(H)$

[51.1] Delete

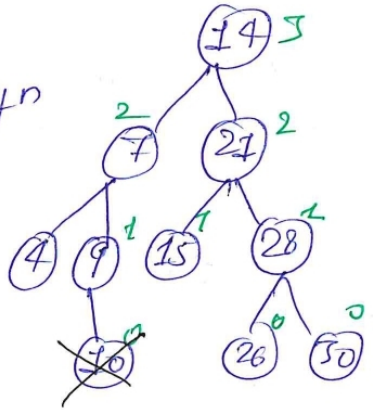


del 2, 3 →

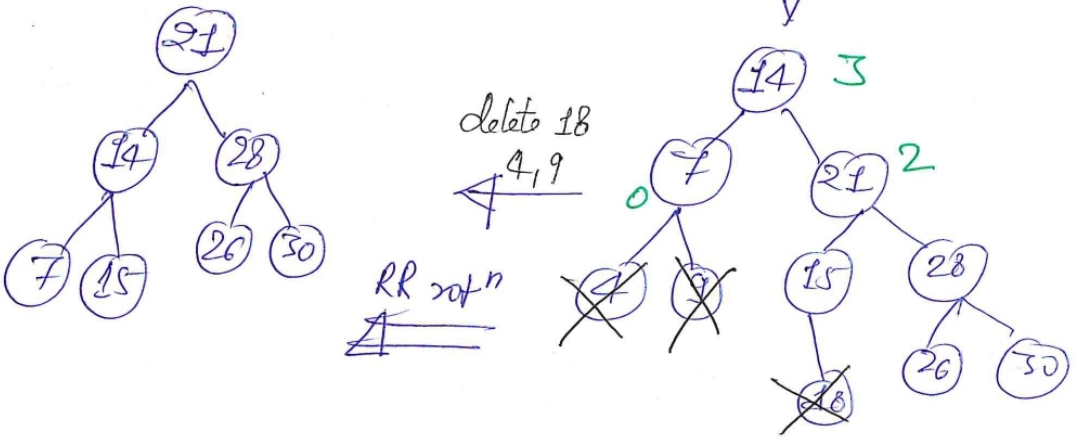
KK rotation



RL rotn →

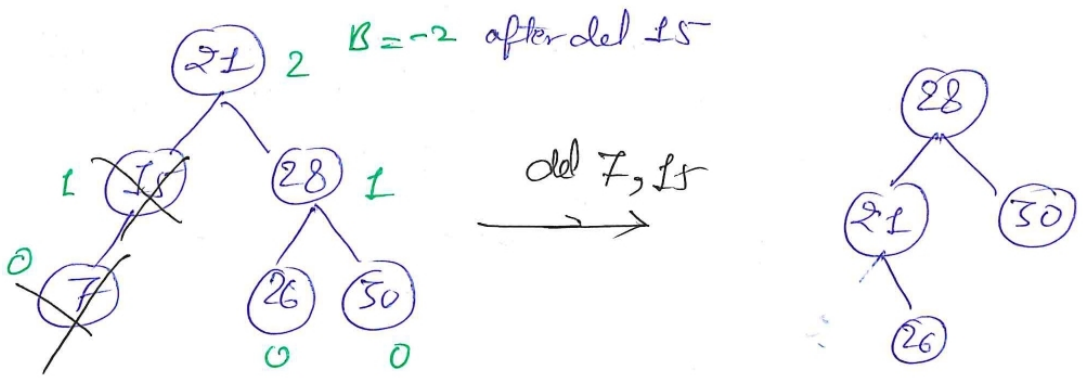


delete 10



delete 14

→ Inorder successor (BST)



⇒ very important property of delete in AVL

insert → const number of rotations ⇒ O(log n)

del → Propogate its impact till the root-node. (heights)

Solved Problem

[53.1] Gate 2009

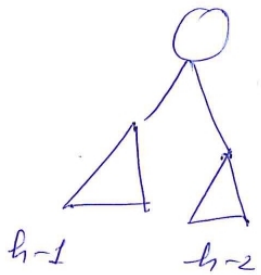
What is the maximum height of any AVL-tree with 7-nodes. Assume that the height of a tree with a single node is 0.

(A) 2 (B) 3 (C) 4 (D) 5

Solution:

$$n(h) = n(h-1) + n(h-2) + 1$$

↑ Min no. of nodes in an AVL tree of height h



$n(0) = 1 \quad n(1) = 2$

$n(0) = 1, \; n(1) = 2$

$n(2) = 1 + 2 + 1 = 4$

$n(3) = 1 + 4 + 2 = 7$

AVL $h = 3$

Max height $\longrightarrow 7 = n$

GATE 2008

Which of the following is true.

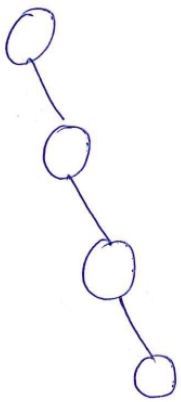
(A) Cost of searching an AVL tree is $\theta(\log n)$ but that of binary search tree is $O(n)$.

(B) Cost of searching an AVL tree is $\theta(\log n)$ but that of a complete binary tree is $\theta(n \log n)$.

(C) Cost of searching a binary search tree is $O(\log n)$ but that of an AVL tree is $\theta(n)$

(D) Cost of searching an AVL tree is $\theta(n \log n)$ but that of a binary search tree is $O(n)$

Solution:



$n \log n$ → cost of building an AVL of n-elements by inserting one after the other.

Sample Question

Given two balanced binary search trees, B1 having n-elements & B2 having m-elements. What is the time complexity of the best known algorithm to merge these trees to form another balanced binary tree containing m+n elements.

(A) $O(m+n)$ (B) $O(m \log n)$

(C) $O(n \log m)$ (D) $O(m^2+n^2)$

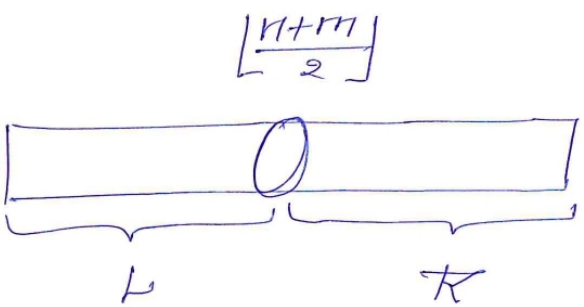
Solution:

B1 → n → Inorder $O(n)$ → Sorted (n)

B2 → m → Inorder $O(m)$ → Sorted (m)

$O(m+n)$

Sorted list of (m+n) elements



Chapter ⇒ Hash-Tables

[54.1] Hash tables: What & Why

Hash table: The DS which is most widely used in the real-world.

BST, AVL → search → $\theta(\log n)$
add → $\theta(\log n)$
delete → $\theta(\log n)$

search → Phone no, google search, Amazon

↳ fast 20 years → 9 M/s / RAM has increased dramatically

15 years → 256 MB 16 GB

2003-2019

Cost of RAM lowered

Extremely fast ways to search → extremely low time complexity

avg case → O(1) time for searching

Hash-table → collection of similar items.

Insert (S, x), Delete (S, x), search (S, x) } very-2 fast

[54.2] Direct Access table : Simplest Way

Ex: store Names & rollno. of students in a class.

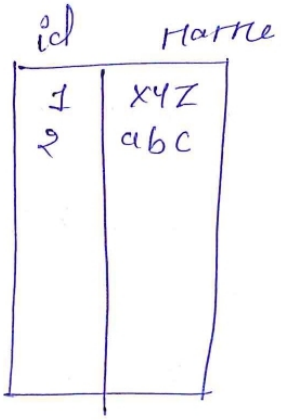
↓

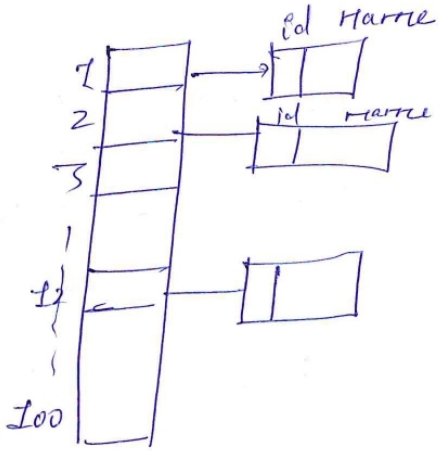
unique identifier (key)

$\{1, 2, \ldots 100\}$

$U = \{1, 2, 3, \ldots 100\}$

Set of all distinct key





Student {
id
Name
}

Simply using an array of pointers/references

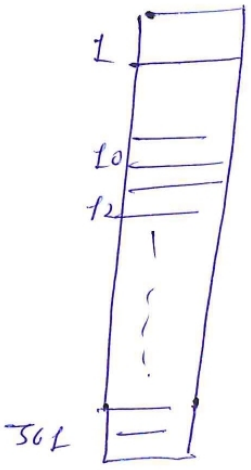
Size = |U|

directly going to index 12 & accessing

search Name → O(1)

id / key = index in the array

Limitations: range of keys is large



id's → 10, 12, 561

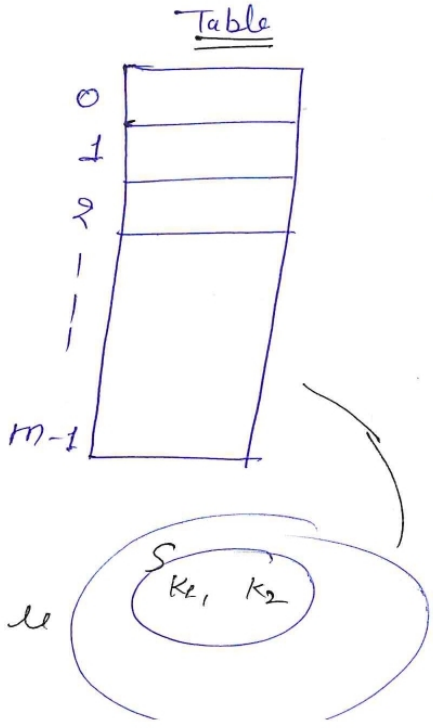
id → index

id → 64-bit no. → 0 to $2^{64}-1$

→ If Range of no. is larger then DAT will not work

[54.3] Hash functions & collisions

Because of limitation of DAT with range of no. is large.



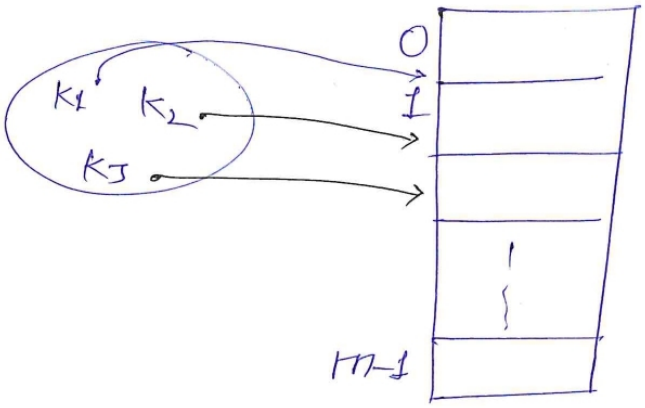
Array T has size m

U – universe of keys
S – set of keys/ids

S ⊂ U

function (Ki) → index

hash function



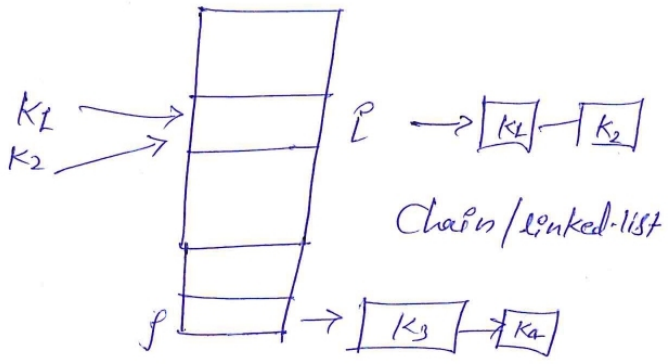
⇒ If two keys are $h(K_2) = 3$, $h(K_3) = 3$ } Collision

⇒ Both are hashed in same slots so colliding

Search = O(1)

[54.4] Chaining & load factor

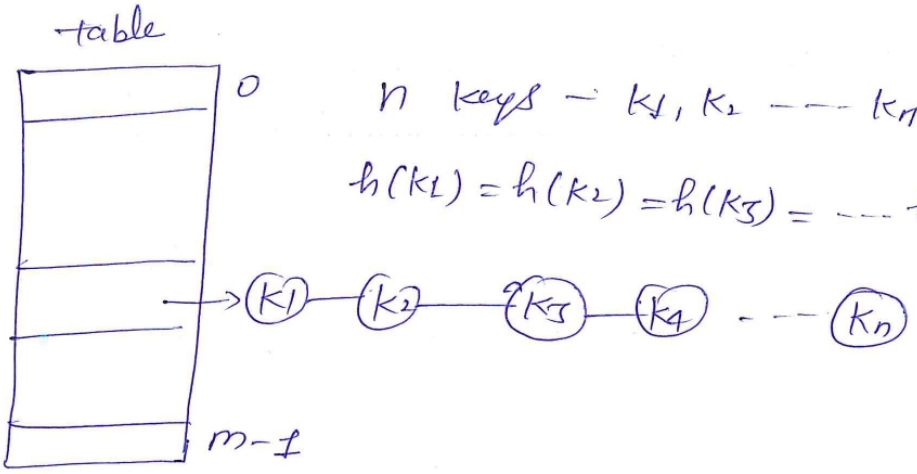
Hash collision :



$h(K_1) = i$

$h(K_2) = i$

Worst Case :



n keys — $k_1, k_2 \dots k_n$

$h(k_1) = h(k_2) = h(k_3) = \dots h(k_n) = i$

⇒ If all the keys collide

$|S| = n$

search, Insert, delete $= \Theta(n)$

Simple uniform hashing

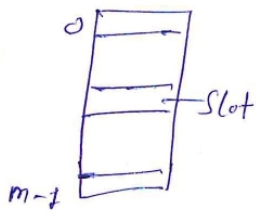
for each $k \in S$ is equally likely to be hashed to any slot in T $\{0, 1, 2 \dots m-1\}$

(because every key has equal chance)

n = no. of keys in S

m = no. of slots / indices in T

load-factor (α) = avg. no. of keys / slot.



average length of each chain

$$\alpha = \frac{n}{m}$$

Time complexity to search for a key is
hash function $\Theta(1)$

avg. No. of comparison $= \alpha$
$= \Theta(\alpha)$

$= \Theta(1+\alpha)$ = load factor

$\alpha = O(1)$ then $\Theta(1)$

Overall time complexity

Worst case
all keys are hashed to the same slot/bucket
$= \Theta(n)$

Hash function

[55.1] Division method (Modulo Hash function)

=> Collisions → Chaining

$h(k)$ = slot / index / bucket

Good hash function : simple uniform hashing
↓ hard to generate

=> Should distribute the keys uniformly into the slots in our Table T. $\{0, 1, 2, 3 - - - - m-1\}$

Division method :

(K) ← Key , T → Size m

$h(k) = k \bmod m$ — modulo hash function

k % m = k mod m = remainder by dividing k by m

$k = 56, m = 101$

$h(k) = 56 \bmod 101 = 56$

$k = 206, m = 101$

$h(k) = 206 \bmod 101 = 4$

m = $\boxed{h(k) = k \bmod m}$

Let $m = 2^r = 2^6$ (let) k (decimal)

$k = 1011001110101_0$

$k \bmod m = 101010$ (last 6 digits)

This is very fast to compute

⇒ h(k) is dependent only on a few bits of k

Recommendation : Pick m to be a prime no.

m = Prime no.

[55.2] Multiplication method

⇒ It is a hash function

$0 < A < 1 \qquad m = 2^p$

$h(k) = \lfloor m * (kA \bmod 1) \rfloor$

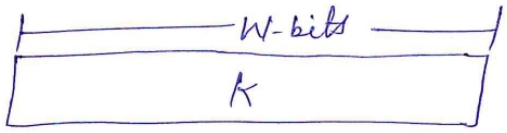
division Modulo → Most widely used hash function

$k \cdot A \bmod 1$ → fractional

↳ real no.

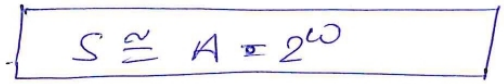
eg: 2.6132

$kA \bmod 1 = 0.6132$

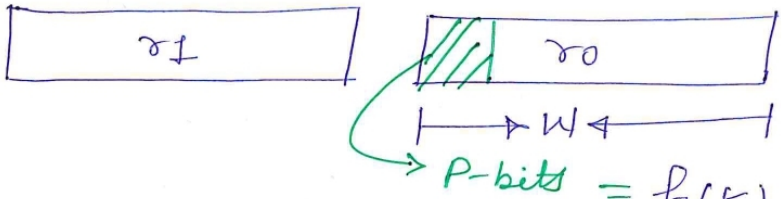


binary representation of k

*



$m = 2^p$



$= \lfloor m * (kA \bmod 1) \rfloor$ → very complex

$A \rightarrow \quad 0 < A < 1$

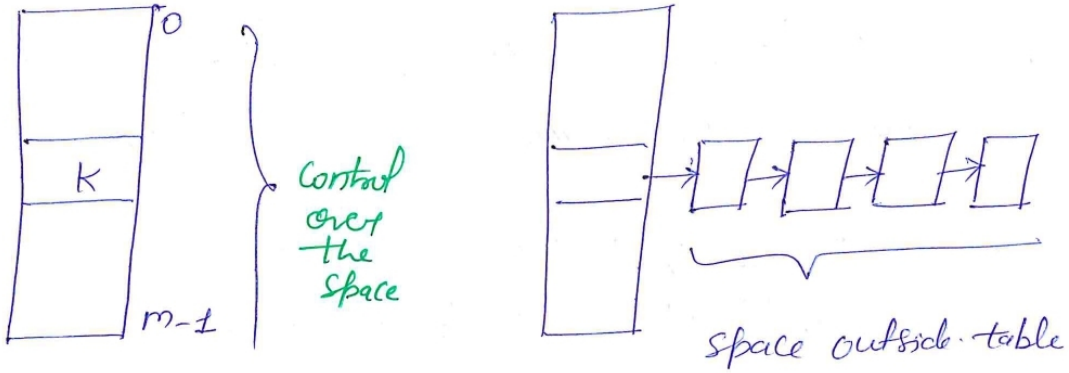
$A = \frac{\sqrt{5} - 1}{2} \simeq 0.618$

Chapter ⇒ Collision Resolution

[56.1] Open Addressing (Closed Hashing)

Chaining → Collisions

Core idea → What if I want to use only the space in it



Typical

$h \rightarrow u = \{0, 1, 2, \ldots m-1\}$

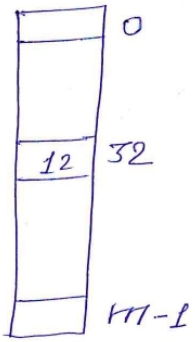
Open addressing $h : u \times \{0, 1, \ldots m-1\}$

insert K — Iteration i — $h(K, 0)$

① : $h(K=26, 0) = 32$

② : $h(K=26, 1) = 15$

③ : $h(K=26, 2) = 21$



Multiplicative / Division } $h(k) =$ index

$h : U \to \{0, 1, 2 \dots m-1\}$

open-addr — $h : U \times \{0, 1, 2 \dots m-1\} \to$

key — Prob-num — index in T

Linear Probing [open addressing]

$h : U \times \{0, 1, 2 \dots m-1\} \to \{0, 1, 2 \dots m-1\}$

key — Prob-num — index

$h'(x)$: multiplicative or division

$h'(x) : U \to \{0, 1, 2, \dots m-1\}$

$$h(k, i) = (h'(k) + i) \bmod m$$

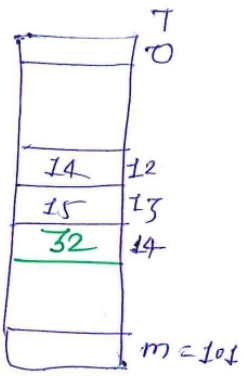
(key, Prob-num) — Linear Probing

Let $k = 32$ $\quad h'(k) = 12$

① $h(k=32, i=0) = (12+0) \bmod 101 = 12$

② $h(k=32, i=1) = (12+1) \bmod 101 = 13$

③ $h(k=32, i=2) = (12+2) \bmod 101 = 14$



Important :

Prob_num : $0, 1, 2 \dots m-1$

Prob_num : Can be any permutation of

$\{0, 1, 2 \dots m-1\}$

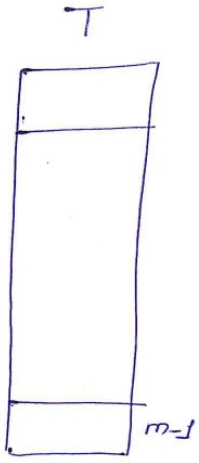
$\{2, 4, 6, 8, 0, 1, 3, 5\}$

Double Hashing

open addressing

$h_1(x)$ & $h_2(x)$: key $\longrightarrow$ index

h_1 & h_2 : $u \longrightarrow \{0, 1, 2 \dots m-1\}$



Standard hash function :

$h(k, i) =$

key, Prob-num

$= [h_1(k) + \{i * h_2(k)\}] \bmod m$

Quadratic Probing

open_addressing

$h'(k) : U \rightarrow \{0, 1, 2, \ldots, m-1\}$ → Standard hash func

$$h(k,i) = \{h'(k) + (c_1 * i) + (c_2 * i^2)\} \bmod m$$

key, Prob_num

Linear, quadratic

$c_2 \neq 0$

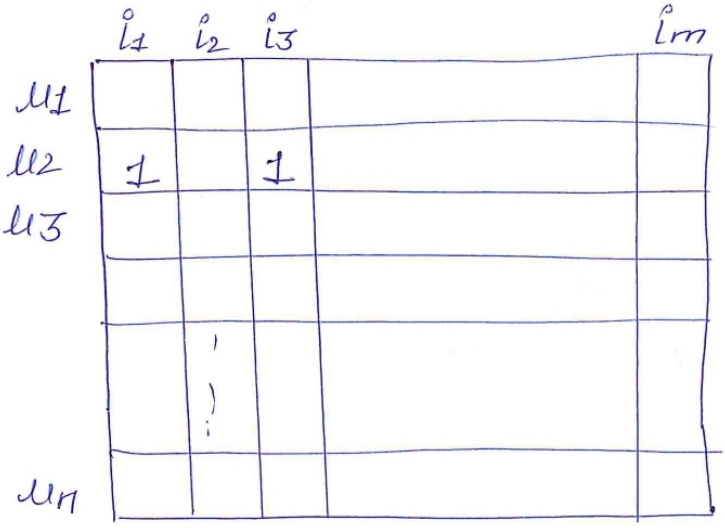
- If $c_2 = 0$, become linear Probing

$h(k, i=0)$
$h(k, i=1)$
$\vdots$
$h(k, i=m-1)$

Chapter ⇒ Applications

Sparse Matrix representation [5.7.1]

Ex: E-commerce like Amazon



m x m
↓ no. of users
→ no. of items

Sparse

most cells will be 0
very few cells will be non-zero
Sparse

how do you store sparse matrix..?
- Instead of storing all the cells info, lets only store info of non-zero cells.

row	column	val
2	3	1
4	6	1
3	21	1

} no. of non-zero cells = 3

hash table

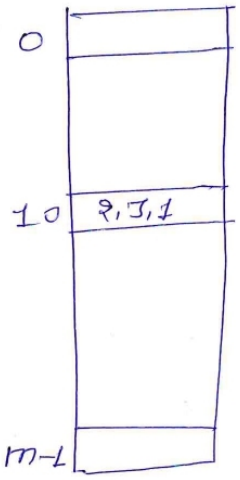
Key: (row, col)

Simple { String → Integer }

Key: concat (row, col)

h (key) = index

h (2:3) = 10



Total space complexity = O(m)

m = O(z)

[572] Super fast Search (→ Internet company)

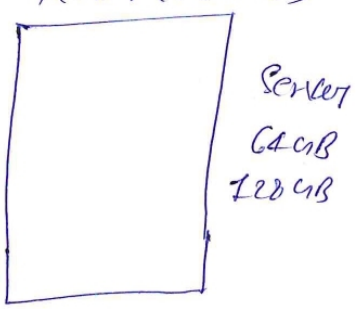
Login : username, Password

↳ v.v. fast

older/slower → DB

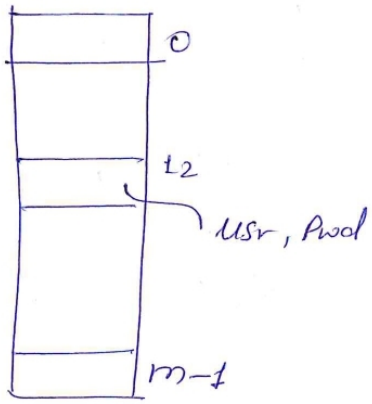
Faster : hash table

→ in RAM, ∃ create my hash table



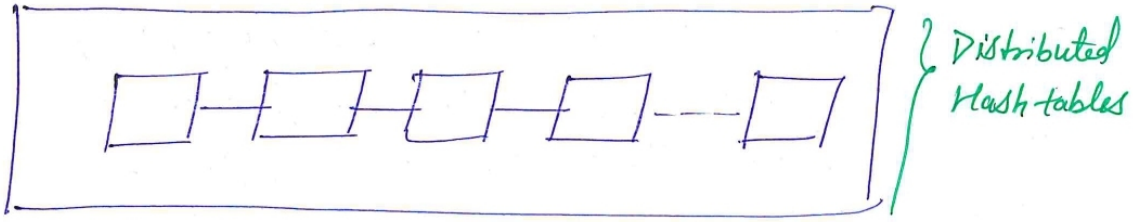
h (username) = index

h (Shrikanth. Verma) = 12



avg. case hashing TC = O(1)

Amazon, google have hash tables in distributed hash-table manner.



Solved Problems

[58.1] GATE 2004

Given that input (4322, 1334, 1471, 9679, 1989, 6171, 6173, 4199) & the hash function x mod 10 which of the following are true.

1. 9679, 1989, 4199 has to same value
2. 1471, 6171 hash to same value.
3. All elements hash to same value.
4. Each element hashes to a different value.

(A) 1 only (B) 2 only (C) 1 and 2 only (D) 3 or 4

150

Solution

$h(K) = K \bmod 10$

$9679 \bmod 10 = 9$

$1989 \bmod 10 = 9$

$4199 \bmod 10 = 9$

GATE 2014

Consider the has table with 100 slots. Collisions are resolved using chaining assuming simple uniform hashing. What is the probability that the first three slots are unfilled after the first 3 insertion.

(A) $(97 \times 97 \times 97) / 100^3$

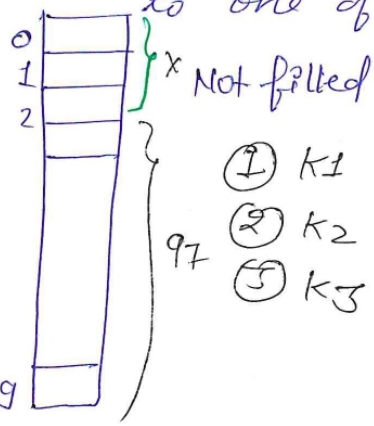
(B) $(99 \times 98 \times 97) \times 100^3$

(C) $(97 \times 96 \times 95) \times 100^3$

(D) $(97 \times 96 \times 95) / (3! \times 100^3)$

Solution:

Simple uniform hashing →
every key is equally-likely to be hashed to one of the slot



Prob these keys will be hashed in 97 slots.

$$= \frac{97}{100}_{K_1} \times \frac{97}{100}_{K_2} \times \frac{97}{100}_{K_3}$$

$$= (97 \times 97 \times 97) / 100^3$$

GATE 2015

Which one of the following hash functions on integers will distribute keys most uniformly over 10 buckets numbered 0 to 9 for ranging from 0 to 2020.

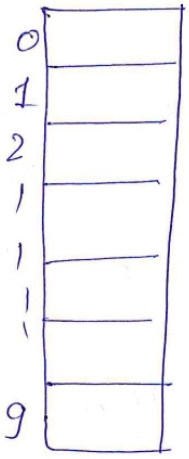
(A) $h(i) = i^2 \bmod 10$ (B) $h(i) = i^3 \bmod 10$

(C) $h(i) = (11 * i^2) \bmod 10$ (D) $h(i) = (12 * i^2) \bmod 10$

Solution

$i^2 \bmod 10$ $\{0, 1, 4, 5, 6, 9\}$ → 6 slots

$1 \to 1 \to 1$
$2 \to 4 \to 4$
$3 \to 9 \to 9$
$4 \to 16 \to 6$
$5 \to 25 \to 5$
$6 \to 36 \to 6$
$7 \to 49 \to 9$
$8 \to 64 \to 4$
$9 \to 81 \to 1$
$10 \to 100 \to 0$
$11 \to 121 \to$



$i^3 \bmod 10$

$1 \to 1 \to 1$
$2 \to 8 \to 8$
$3 \to 27 \to 7$
$4 \to 64 \to 4$
$5 \to 125 \to 5$
$6 \to 216 \to 6$
$7 \to 343 \to 3$
$8 \to 512 \to 2$
$9 \to 729 \to 9$
$10 \to 1000 \to 0$

} no collision

GATE 2006

Which of the following is true

(I) A hash function takes a message of arbitrary length & generate a fixed length code.

(2) A hash function takes a message of fixed length & generate a code of variable length.

(3) A hash function may give the same hash value for distinct messages.

(A) I only (B) 2 & 3 (C) 1 & 3 (D) II only

Solution: $h(K)$

$h: u \rightarrow \{0, 1, 2 ---- m-1\}$

any variable length i/p & generate fixed length o/p

GATE 2015

Given a hash table T with 25 slots that stored 2000 elements, the load factor α for T is

Solution :

$$\text{load factor } (\alpha) = \frac{n}{m} = \frac{2000}{25} = 80$$

= avg. no. of items that are hashed to same slot assuming simple uniform hashing

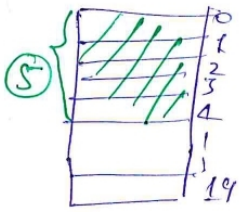
GATE 2007

Consider a hash function that distributes key uniformly. The hash table size is 20. After hashing of how many keys will the probability that any new key hashed collides with an existing one exceed 0.5.

(A) 5 (B) 6 (C) 7 ✓(D) 10

Solution :

Prob. of k will collide with existing keys → $\frac{5}{20} = 0.25$



$6 \rightarrow \frac{6}{20} = 0.3$, $\frac{7}{20} < 0.5$ $\frac{10}{20} = 0.5$

Sample Question – 8

The characters of the string K, R, P, C, S, N, Y, T, J, M are inserted into a hash table of size 10. using hash function

$$h(x) = ((ord(x) - ord(A) + 1) \bmod 10)$$

If linear probing is used to resolve collisions then the following insertions causes collision.

(A) Y (B) C (C) M (D) P

Solution :

A[1] B[2] C[3] D[4] E[5] F[6]
G[7] H[8] I[9] J[10] K[11] L[12]
M[13] N[14] O[15] P[16] Q[17] R[18]
S[19] T[20] U[21] V[22] W[23] X[24]
Y[25] Z[26]

$h(x) = (ord(x) - ord(A) + 1) \bmod 10$

$= K - A + 1$

$= 11 - 1 + 1 \quad \bmod 10 = 1$

R mod 10 = 8
P mod 10 = 6
C mod 10 = 3
S mod 10 = 9
N mod 10 = 4
Y mod 10 = 5
T mod 10 = 0
J mod 10 = 0 + 1 = 1 + 1 =
M mod 10 = 3

0	T
1	K
2	J
3	C
4	N
5	Y
6	P
7	M
8	R
9	S