DSA (3-1-3)

Data-type : A method of interpreting data.
≡ finite set of operations consistent with a concept.

Two types of data { "Native" ; "Abstract" → what we wish to get done

ADT → Data + finite set of operations
ADT → Rational nos. → p/q
Data : Two int x & y
condition : $y \neq 0$

$r \rightarrow r[0] \rightarrow N_r$
$r[1] \rightarrow D_r$

[OP1] mul (r_1, r_2) $\quad \frac{r_1[0]}{r_1[1]} \quad \frac{r_2[0]}{r_2[1]}$

mul [0] = $r_1[0] * r_2[0]$
mul [1] = $r_1[1] * r_2[1]$

[OP2] $\quad \frac{r_1[0] * r_2[1] + r_1[1] * r_2[0]}{r_1[1] * r_2[1]}$

[OP0] make rational (a, b)
- Precondition - $b \neq 0$

Post condition — make rational [0] = a
make rational [1] = b

All these operations are finite set of operations

01/08/19

3-5 pm → Tue/Wed/Thu

Assg

Marks distribution (Revised)

Exam (10×2+30)	Tutorial Test – (10)
Lab (7)	Others (8)
Progg Assign (25)	(Lab test/Quiz)

Part A → Regular → 5 question 100 marks

Part B → Penal/Risk → 1 or 2 qn → 120 marks → 80 marks

---

(1)

```
int a, i;
float avg=0;
for(i=0; i<5; i++)
{ scan(a)
  avg += a;
}
Printf(avg);
```

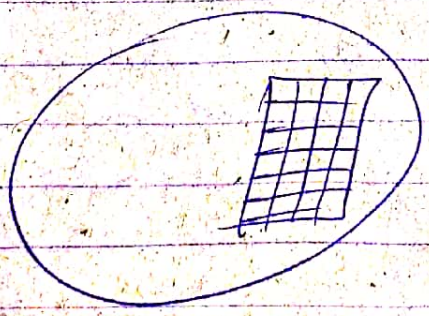
(2)

(3)

10000 int

ADT ?

"Rack"



Make Rack(int size, size)

Store( )

Extract( )

Make Rack(int size, size)

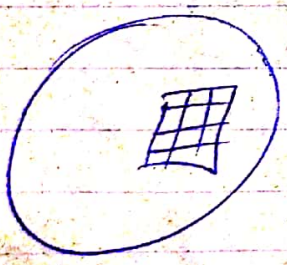
{

Rack = (int *)malloc(size of(int) * size);

}

```
Store (int Rack ; int value ; int position)
{
    *( Rack + Position) = value
}
```

```
Extract ( int* Rack ; int position)
{
    return *(Rack + position);
}
```



- Capacity constraint
- Pre-allocation
- Only bags.
- Min size based on

Q. Matching Brackets.

(((1+B) * (C-D) + E) * F)

↑

count the opening & closing brackets

count = 0

count ≥ 0

Q. Matching Brackets

({ [

{ [(A+B) * (C-D) + E] * F }

T(1)

$T(n) = T(n/2) + T(n/2) + n$

$= 2T(n/2) + n$

$T(n) = 2^1 [T(n/2^1)] + n$

05/8/19

Matching Brackets:

{ [(A+B) * (C-D) + E] * F }

⇓ (what we care about)

{ [( ) ( )] }

Effort = Computational requirement

Time

$T(n) = T(n/2) + T(n/2) + n$

$= 2T(n/2) + n$

$T(1) = 1$

$= 2^1 [T(n/2^1)] + n$

$= 2^1 [2T(n/2^2) + n/2] + n$

$= 2^2 T(n/2^2) + 2n$

$= 2^3 T(n/2^3) + 3n$

i times

$$\boxed{T(n) = 2^i T(n/2^i) + in} \quad \forall i$$

$$\frac{n}{2^i} = 1 \Rightarrow i = \log_2 n$$

$$T(n) = 2^{\log_2 n} T(1) + n\log_2 n$$

$$\boxed{T(n) = n + n\log_2 n}$$

{[( )( )]}

ADT method

{ Last in first out (LIFO)

→ **STACK**

- Push
- Pop
- Top

- Is empty
- Size
- Make stack

- clearing-up

"stack overflow"

Algo: while (not done)

{

- Read the next symbol, S.
- If S is '(' || '{' || '[' then

PUSH (stack, S)

- else if S is ')' || ']' || '}' then
- if Empty (stack), then invalid (exit)

c = 0
c = 1
c = 2
c = 3

- else
  c = POP (stack)

& check whether c is the closing version of s

If (c != complement (s)) then invalid

end while (if they are same, repeat)

- At the end, if empty (stack) then valid (else invalid)

- stack does not require size to be given beforehand.

```
#define MAXSIZE 100
typedef struct {
        int top;
        int items[MAXSIZE];
} stack;

PUSH (stack *s, int value)
{ s->items[++s->top] = value; }

int POP (stack *s)
{ if (!Empty (s))
    return (s->items[s->top--]);
```

```
int Empty (stack *s)
{ if ((s->top) == -1)
    return (TRUE)
  else
    return (FALSE)
}
```

Lab

(1)

```
int main( ).
{ char stack [100]; int counter = 0;
  char a[100];
  scan(a);
  size = strsize(a);
  for(i=0; i<size; i++)
      if (a[i] = '(' or '{' or '[')
        -stack [counter] = a[i]; counter++;
      elseif (a[i] = ')' or '}' or ']')
        -counter--;
        -Check, if stack [counter] is
         counterpart of a[i]
        -if yes, continue
        -if no, invalid
```

07/08/19

A + B
↑ operator
↙ Operand 1
↘ Operand 2

(A+B) * C — Infix Notation
A + (B * C) — Infix Notation

AB +
AB + C *
AB C * +

Postfix Notation

Step: Define the set of operators

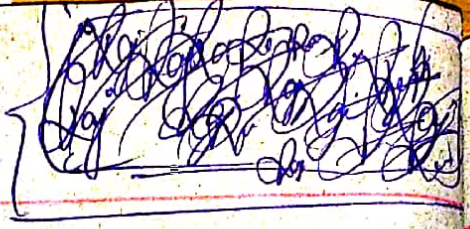
while (not done)
- Read the next symbol
- If it is an operand
  - Push ( )
- If it is an operator
  - opr2 = pop ( )
  - opr1 = pop ( )
- Do push (opr1 op opr2)

end while
ans = pop ( )

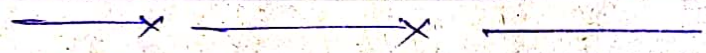
Q. 37 + (44 * 17)

* Implement stack using array

Substitute → array



```
Struct substack
{
    int top;
    int items [MAXSIZE];
    struct substack *previous;
};
```



```
int pop (stack *top)
{
    stack *temp; int x;
    if (Empty (top))
        exit (1);
    else
    {
        temp = top
        top = top -> previous;
        x = temp -> item;
        free (temp);
        return (x);
    }
}
```

```
Empty (stack * top)
{ if (top == NULL)
    return (TRUE)
  else
    return (FALSE)
{
typedef struct substack
{ int localtop ;
  int items [MAXSIZE];
  struct substack * previous;
{ stack

PUSH (stack * top, int value)
{ stack * temp;
  if (top -> localtop < MAXSIZE)
     && (top != NULL)
{ top -> items [++top -> localtop] = value;
{
else
  { temp = (stack *) malloc (sizeof (stack));
    temp -> localtop = 0
    temp -> items [0] = value ;
    temp -> previous = top;
```

top = temp;

}

{

09/08/19

Recursion:

```
Rec ( )
if (terminating condition)
  return (...)
else {
    Rec( );
  }
```

```
int factorial (int n)
{
  if (n == 1)
  return (1);
else
  return (n * factorial (n-1))
}
```

n = 3

2 * f(1)

3 * factorial (2)

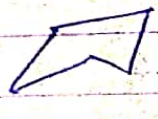
Recursion

```
while (top -> n > 1)
{ push (top, top -> n-1)
{
top -> ans = 1
```

```
while (top -> previous != NULL)
{
    ans = pop(top);
    top -> ans = ans * top -> n
}

Answer = top -> ans
```

Polygon :



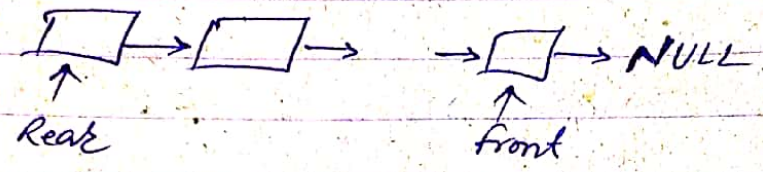
concave

convex

Given a set of points, construct a polygon which contains all these point with the least perimeter
Solve it using the concept of stack
Input → (x_1, y_1) , (x_n, y_n)
Output → Perimeter

```
typedef struct substack
{
    int localtop;
    int localMaxsize;
    struct substack * previous;
}
Stack
```

Queues :



"first out"
first out"

Adding a node → enqueue()
Remove a node → dequeue()

```
typedef struct
{ int items [MAXSIZE];
  int front, rear;
{
```

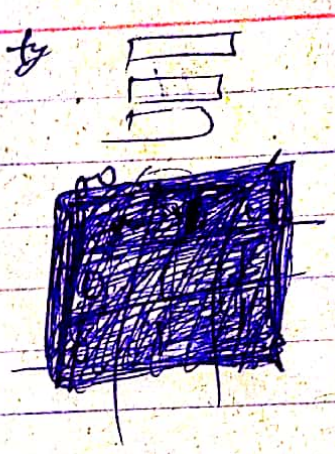
```
int Enqueue (Queue *Q, int
{ if (Q->rear < MAXSIZE )
    Q->items [++ Q->rear]=
  else
    overflow;
```

```
int Dequeue (Queue * Q)
{ if (!Empty(Q))
    return (Q->items [Q->front++ ]);
{
```

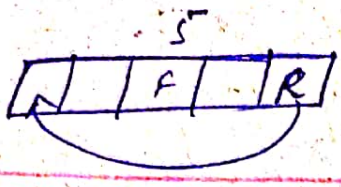
```
Empty (Queue * Q) {
  if (Q->rear - Q->front+1 == 0)
    TRUE;
  else FALSE;
{
```

10/08/19

```
typedef struct queue
{
 int items[MAXSIZE];
int front, rear ;} Queue;
int Enqueue(Queue *Q, int value)
{ if (Q -> rear < MAXSIZE-1)
     Q -> items [++Q->rear ] =value
  else
     overflow;
}
int - Dequeue (Queue *Q)
{ if (!Empty Q)
   return (Q -> items [Q ->front++])
}
Empty (Queue *Q)
{ if (Q->rear -Q->front +1==0)
   TRUE ;
  else
     FALSE;
}
```



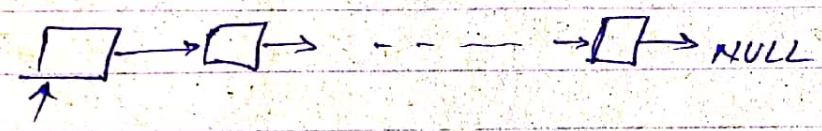
M3



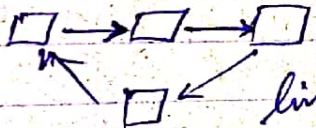
circular queue

LIST :

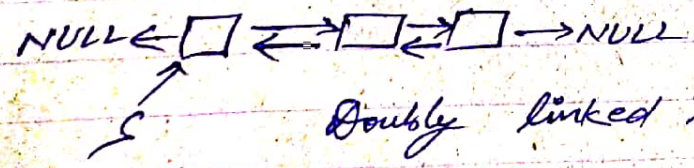
```
typedef struct node
{
    int item;
    struct node * next;
} list;
```



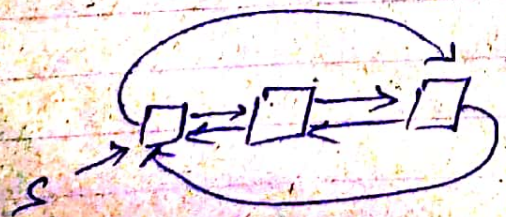
List are the special case of stack.



circular linked list



Doubly linked list



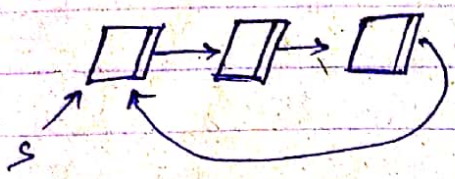
circular Doubly linked list.

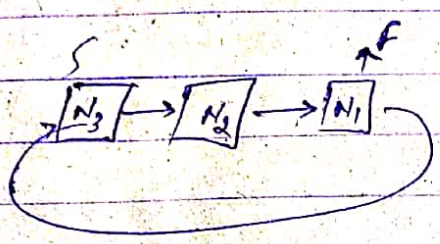
← In this case we can traverse both the way

14/08/19

Queue:

Using circular linked list





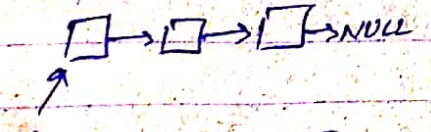
MAZE

(1,1) 1	(1,2) 1	(1,3) 0
(2,1) 1	(2,2) 1	(2,3) 0
(3,1) 0	(3,2) 1	(3,3)

19/08/19

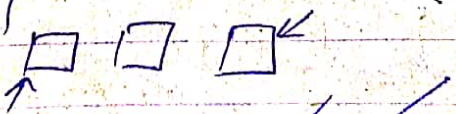
LIST

```
{ int item;
  struct node *next;
}
```



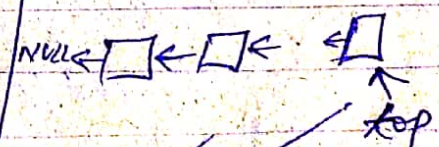
QUEUE

```
{ int item,
  struct node *rear;
  struct node *front;
  struct node* ;
}
```



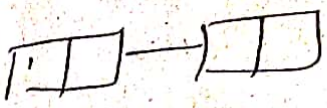
STACK

```
{ int item;
  struct node *previous;
}
```



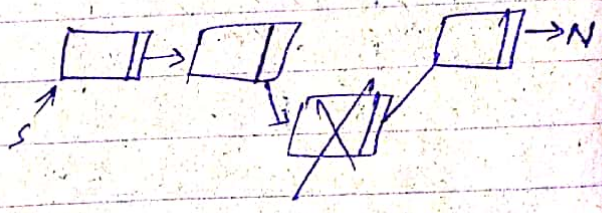
LIST:

```
typedef struct node
{ int item;
  struct node *next;
} list;
```



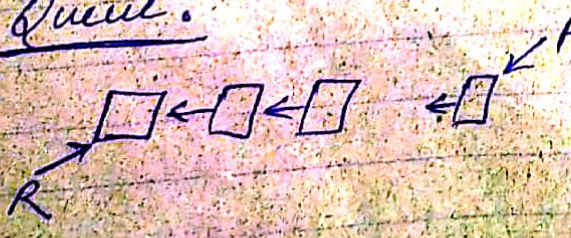
```
Insert (list *p, int value)
{
  list *new;
  //allocate memory to "new";
  new->item = value;
  new->next = p->next;
  p->next = new;
}
```

```
Delete (list *p)
{ list *temp;
  temp = p->next;
  p->next = (p->next)->next;
  free (temp);
}
```




```
Printlist (list *S)
{
  while (S->next != NULL)
  { S = S->next
    print (S->item);
  }
}
```

Queue:



Saturday → 11 → 07 pm

Q. Implement a queue using circular singly linked list using only one pointer (either "front" or "rear").

MAZE:

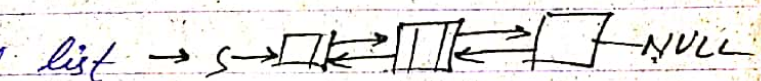
```
1011
1110
1111
```

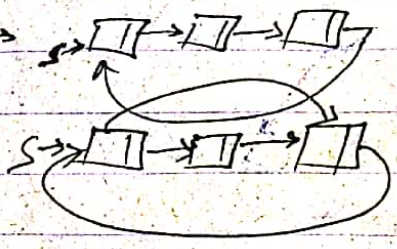
node
{x, y,
dist
{

Queue

19/08/19

Lab

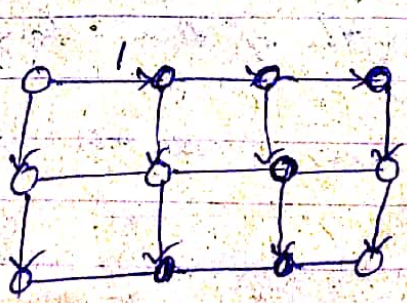
1. RACK implementation (for integers)
2. Matching Brackets
3. factorial using stack
4. Standard List & Queue (node)
5. Variants of List
   - Doubly linked list → 
   - Circular →
   - Circular Doubly →



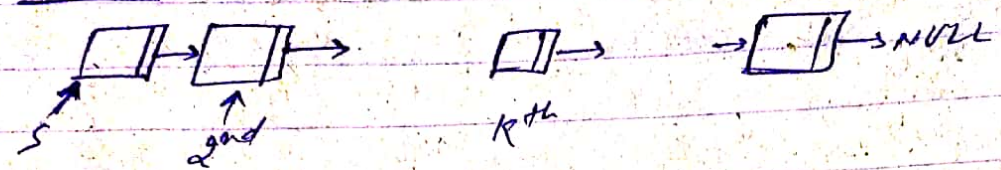
21/08/19

MAZE problem:

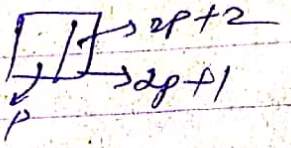
```
1 1 0
1 0 1
1 1 1
```



LIST :



~K steps ~K pointers



$0 \to 1 \to 3 \to 7 \to 15 \to \cdots$

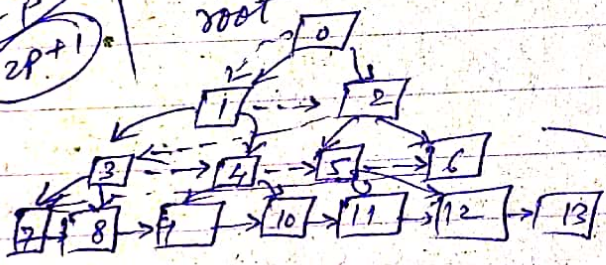
$2^K - 1$

$t = 2^K - 1 \Rightarrow K = \log(t+1) \simeq \log(t)$

apply for p

2p+1

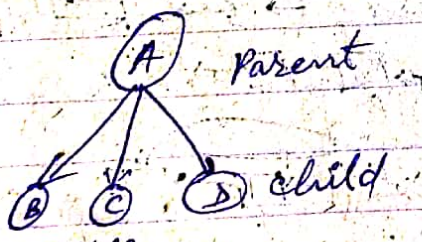
2p+2



Leaf nodes

Tree

depth = 3

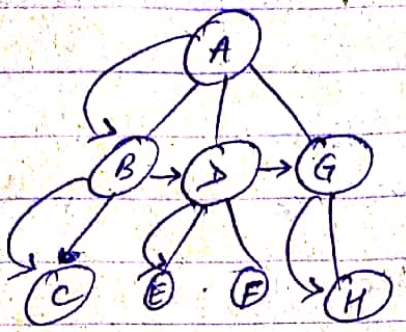


"depth of tree" = length of the longest path from root

depth of tree = depth of root node

22/08/19

Tree



Depth A = 2
B = 1
C = 0

Representing trees

```
typedef struct node
{ int item,
  struct node* firstchild, * nextsibling;
} Tree;
```

Parent
Firstchild
Nextsibling

Priority

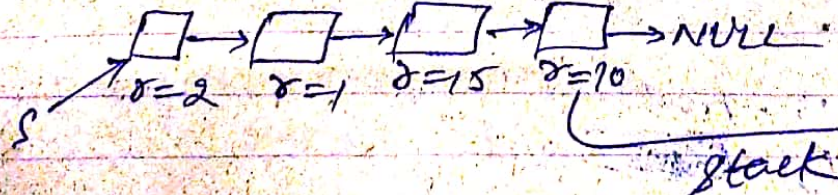
```
PrintTree (Tree*T)
{ Tree * temp;
  if (T != NULL)
     print (T->item);
  temp = T->firstchild;
  while (temp != NULL)
  { PrintTree (temp);
     temp = temp->nextsibling;
  }
```

I
A
temp = B
PrintTree(B)
temp D.
PrintTree(D)

II
B

III
C

LIST :

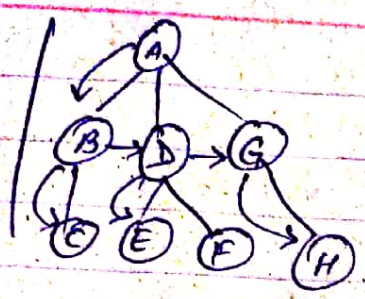


stack

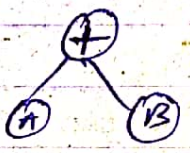
Please write the steps

23/08/19

11 AM to 1 PM - Saturday

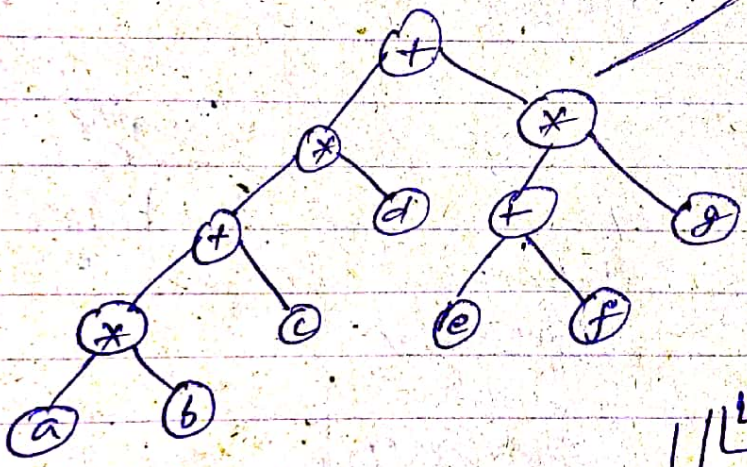



```
typedef struct node
{
  int item;
  struct node *left, *right;
} BinTree;
```



(((a*b)+c)*d)+((e+f)*g) → Infix notation

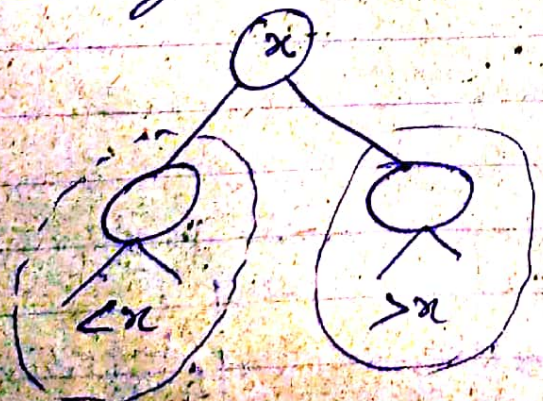
change to Postfix notation

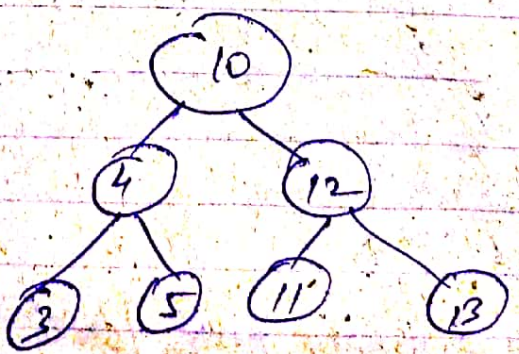


Expression Tree

ab*c+d*ef+g*+

Binary Search Tree





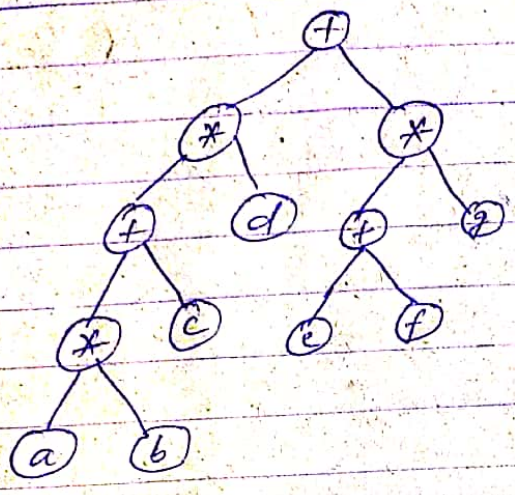
find
find min
find max
Insert

K_1 - Find
K_2 - Insert
K_3 - Delete
$K_1 >> K_2 + K_3$

24/08/19

Expression Tree

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



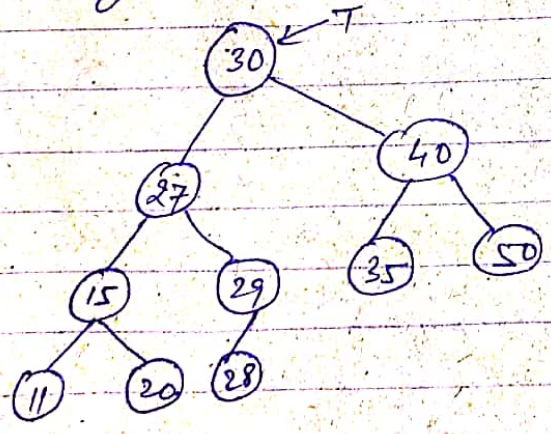
Infix
- Left child
- parent
- right child

Postfix
- left child
- right child
- parent

Prefix
- Parent
- left
- Right

Postfix → ab*c+d*ef+g*+

Binary Search Tree (BST)



depth of ~~tree~~ = 3

```
Tree *find ( int value, Tree * T)
{ if (T == NULL)
    (Not found)
  elseif (T→ item > value)
    return (find (value, T→left) ;
  elseif (T→item < value)
    return (find (value, T→ right) ;
  Else {return T;}
}
```

```
Tree * findMin (Tree *T)
{
    if (T == NULL)
        (Empty Tree)
    if (T->left != NULL)
        return (findMin (T->left));
    else
        return T;
}
```

```
Tree *findMax (Tree *T)
{
    if (T == NULL)
        (Empty Tree)
    if (T->right != NULL)
        return (FindMax (T->right));
    else
        return T;
}
```

```
Tree * Insert (int value, Tree *T)
{
    if (T == NULL)
    {T = malloc (size of (Tree));
        T->item = value;
        T->left = T->right = NULL;
}
```

```
elseif (T→item > value)
    T→left = Insert (value, T→left);
elseif (T→item < value)
    T→right = Insert (value, T→right);
return T;
}
```

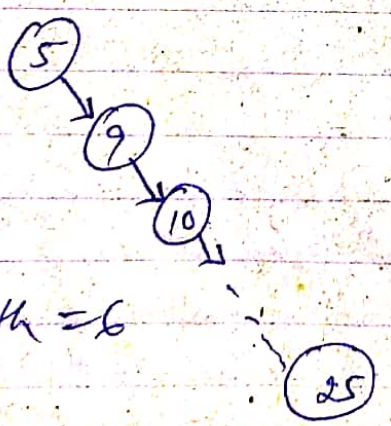
#

```
Tree * Delete (int value, Tree *T)
{ Tree * temp;
  if (T == NULL) print (Error);
  elseif (T→item > value)
      T→left = Delete (value, T→left),
  elseif (T→item < value)
      T→right = Delete (value, T→right)
  else if (T→left != NULL && T→right != NULL)
  { /* 2 children case */
    temp = findMin (T→right);
    T→item = temp→item;
    T→right = Delete (temp→item, T→right);
    free (temp);
  }
  elseif (1 child)
  { if (T→left == NULL)
```

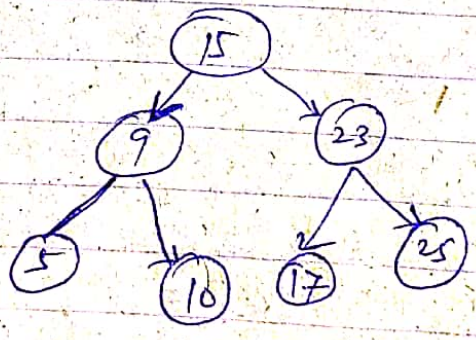
```
{ Tree *z;
  z=T; T=T->right;
  free(z);
}
else
{ Tree *z;
  z=T, T=T->left;
  free(z);
}
return T;
}
```

elseif (0 child)
{ z=T
free(z);
}

Problem with BST → "bad" insert sequence
5, 9, 10, 15, 17, 23, 25



depth = 6



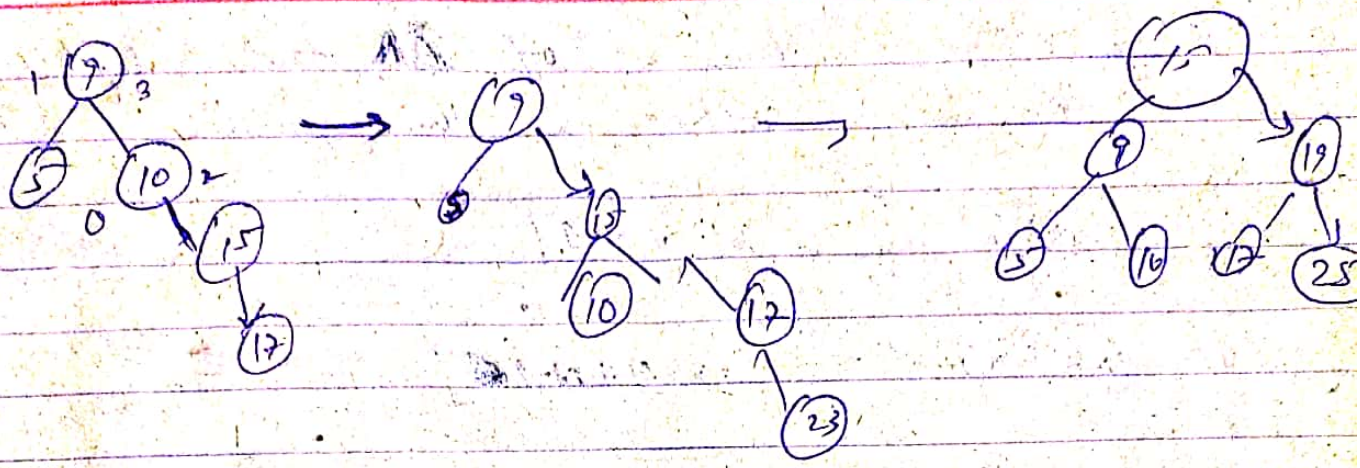
depth = 2

if T → NULL
depth = -1

depth of 5 is 0

Optimum depth
n nodes
depth = log(n)

depth = d
max no. of nodes
$n \leq 2^{d+1} - 1$



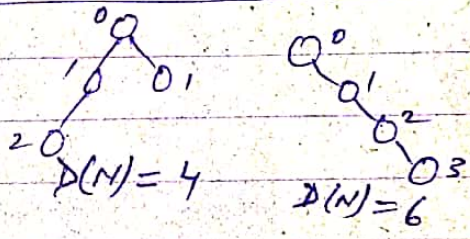
26/08/19

BST:

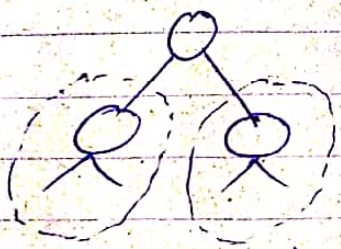
Assume all trees are equally likely (for "N" nodes)

Let $D(N)$ = sum of depths of all nodes in a BST

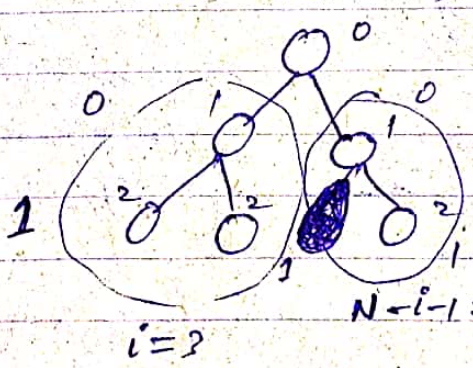
Average depth $davg = \frac{D(N)}{N}$



for N nodes



$$D(N) = D(i) + D(N-i-1) + N-1$$

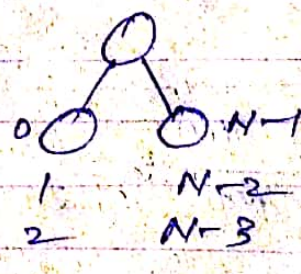


$$D(N) = D(i) + D(N-i-1) + N-1$$

Avg $D(N)$ for an "N" nodes BST

$$\overline{D(N)} = \frac{1}{N} \sum_{i=0}^{N-1} [D(i) + D(N-i-1) + N-1]$$

N nodes



for N nodes ← nodes in

On an average, the no. of nodes in the left subtree & the right subtree is the same

$$N\overline{D(N)} = \sum_{i=0}^{N-1} [2D(i) + N - 1]$$

N times N-1

$$(N-1)\overline{D(N-1)} = \sum_{i=0}^{N-2} [2D(i) + N - 2]$$

(N-1) times N-2

2N-2

$$N\overline{D(N)} - (N-1)\overline{D(N-1)} = 2D(N-1) + 2N - 2$$

$$\frac{N\overline{D(N)}}{N(N+1)} = \frac{(N+1)D(N-1)}{N(N+1)} + \frac{2N-2}{N(N+1)}$$

$$\frac{D(N)}{N+1} = \frac{D(N-1)}{N} + \frac{2N-2}{N(N+1)}$$

Replace N by (N-1)

$$\frac{D(N-1)}{N} = \frac{D(N-2)}{N-1} + \frac{(2N-1)-2}{(N-1)N}$$

Replace N by N-2

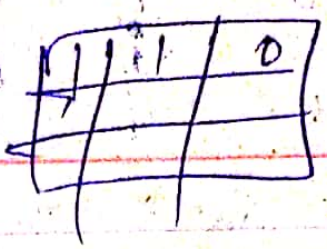
$$\frac{D(N-2)}{N-1} = \frac{D(N-3)}{N-2} + \frac{2(N-2)-2}{(N-2)(N-1)}$$

$$\frac{D(2)}{3} = \frac{D(1)}{2} + \frac{2(2)-2}{(N-2)(N-1)}$$

$$\frac{D(N)}{N+1} = \frac{D(1)}{2} + \sum_{i=0}^{N-2} \frac{2(N-i)-2}{(N-i)(N-i+1)}$$

$$\therefore D(N) = (N+1) \sum_{i=0}^{N-2} \frac{2(N-i)-2}{(N-i)(N-i+1)}$$

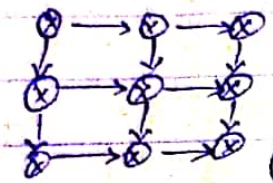
26/08/19



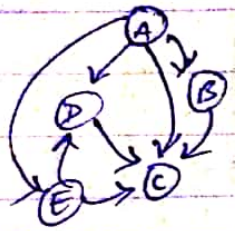
Lab

- Maze Problem

Rectangular Maze
- Basic Version-1 (Binary weights; find the min dist)
- Basic Version-2 (binary weights, find the min dist and print the corresponding path(s))
- Basic Version-3 (non-negative weights; find ---")

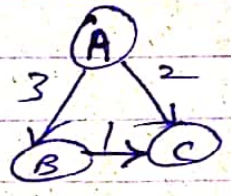


- Generalised Version (")
(for any directed graph)



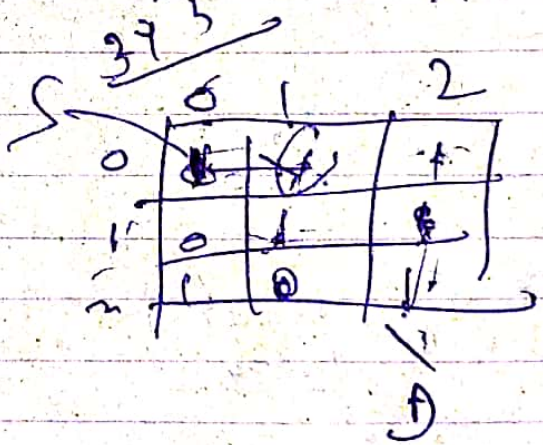
BST — Representing a Maze
Use a 2D array

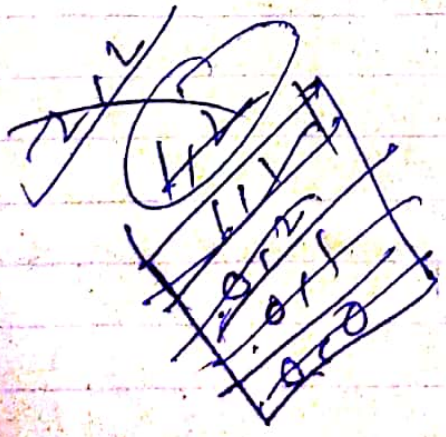
	A	B	C
A	0	3	2
B	0	0	1
C	0	0	0



Ans 1:

```
typedef struct node
{ int x, y;
  int d;
  struct node *previous;
} node;
```



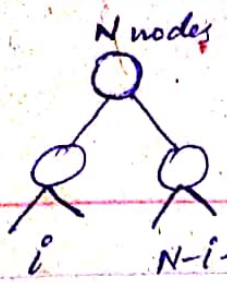


0 → don't move
1 → move

Distance → +1

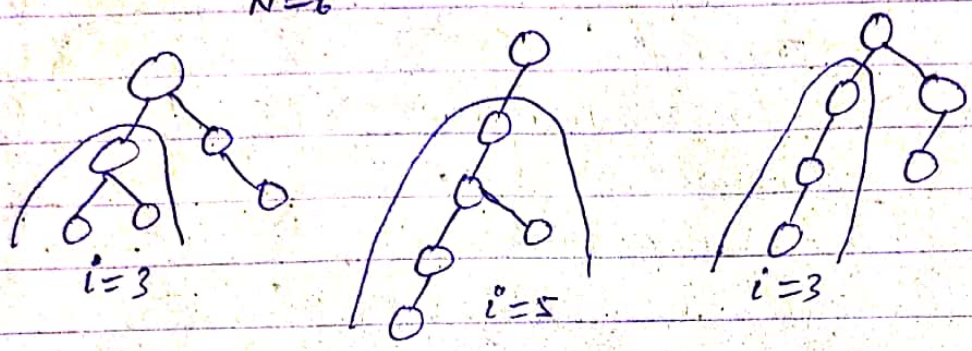
Distance = 0, +1 = 2, +1 = 3, +1 = 4

27/08/19



$D(i) + D(N-i-1) + N-1$

$N=6$



$$D(N) = \frac{1}{N}\sum_{i=0}^{N-1}\left[D(i) + D(N-i-1) + N-1\right]$$

Left subtree ← LST: 0, 1, ..., N-1 ; RST: N-1, N-2, ..., 0

sum of all depths of all nodes

$$D(N) = (N+1)\sum_{i=0}^{N-2}\frac{2(N-i)-2}{(N-i)(N-i+1)}$$

$$\therefore D(N) = \sum_{i=0}^{N-2}\frac{2(N+1)}{N-i+1} - \sum_{i=0}^{N-2}\frac{2(N+1)}{(N-i)(N-i+1)}$$

Linear / Linear — Linear / Quadratic → for very large N tend to 0.

$$D(N) \approx \sum_{i=0}^{N-2}\frac{2(N+1)}{N-i+1} = 2(N+1)\sum_{i=0}^{N-2}\frac{1}{N-i+1} = 2(N+1)\left[\sum_{i=3}^{N+1}\left(\frac{1}{i}\right)\right]_2$$

$$\simeq 2(N+1)\left[\log N - \frac{3}{2}\right]$$

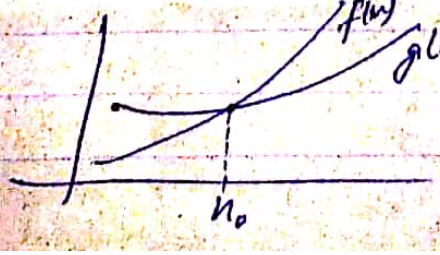
$$D(N) = O(N \log N)$$

[For given functions $g(N)$ & $f(N)$, $g(N) = O(f(N))$ ⇒ growth rate of $f(N)$ > growth rate of $g(N)$]

sum of harmonic series

If $\exists\ c, n_0 > 0$, such that for $n > n_0$ (sufficiently large n)

$g(n) \le c \cdot f(n)$ (g is upper bounded by f)

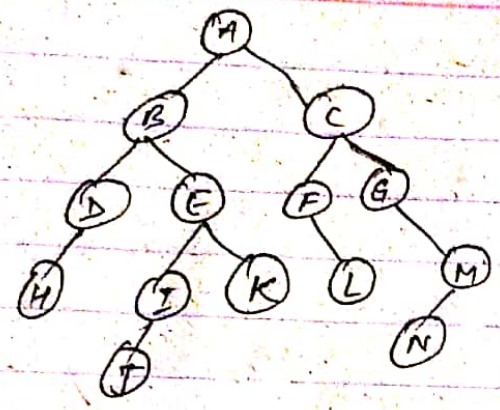


$davg = O(\log N)$

↳ Average depth of a node

$N=4$ $\quad \frac{0+1+1+2}{4} = 1 \approx \log 4 = 2$

28/08/19



Preorder :
ABDHEIJKCFLGMN (P-L-R)

Inorder :
HDBJIEKAFLCGNM (L-P-R)

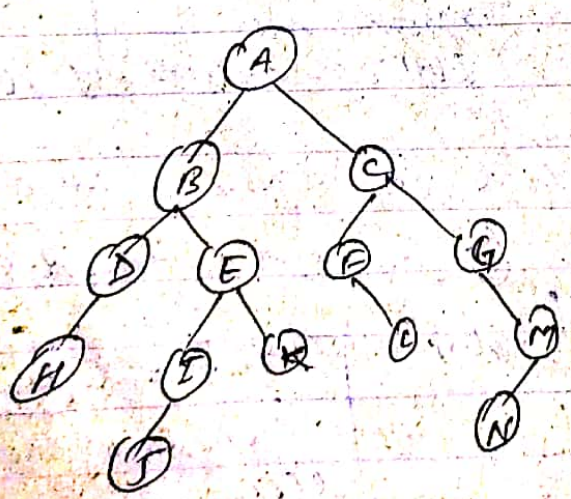
Postorder :
HDJIKEBLFNMGCA (L-R-P)

Problem : Given inorder and pre-order traversal, construct the tree

- Exam (H)
- Lab (2-4:50)
- Homework
- Prog. Assignment (25%)

→ (1) convex hull
→ (2) stack with variable sized array.
→ (3) circular Queue (use either rear or front)
→ function, pseudo-codes, prog-execution
→ 10+10+30
MS-I MS-II → ES → Final

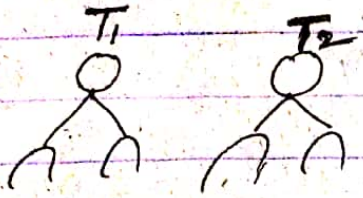
Preorder
→ ABDHEIJKCFLGMN (P-L-R)
Inorder
HDBJIEKAFLCGNM (L-P-R)



29/08/19

BST

① Compare BST (BST T_1, BST T_2)



$T_1 \rightarrow value = T_2 = value$

compare ($T_1 \rightarrow left$, $T_2 \rightarrow left$)

② Height BST (BST T)

$1 + max[Height(T \rightarrow left), Height(T \rightarrow right)]$

③ find the K^{th} smallest element in a BST (T)

↳ K^{th} element in the inorder traversal
↳ (L-P-R)

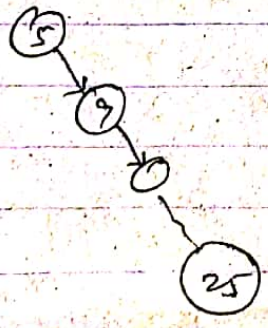
④ K^{th} largest → RPL (Right - Parent - Left)

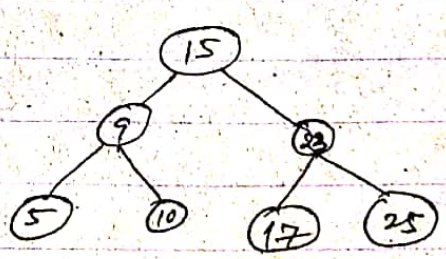
AVL Trees

Problem with BST

→ "bad" insert sequence

5, 9, 10, 15, 17, 23, 25





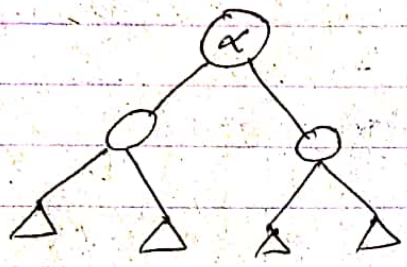
for all nodes

depth of LST = depth of RST

$|(\text{depth of LST}) - (\text{depth of RST})| \leq 1$

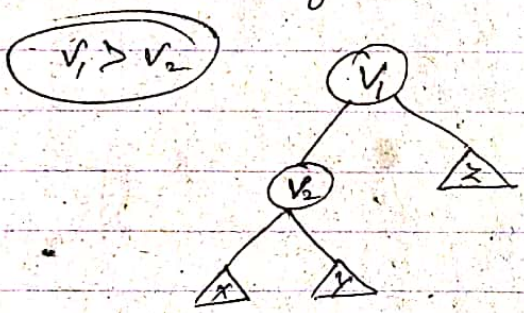
$c O(\log N) \qquad d = 1.44 \log N$

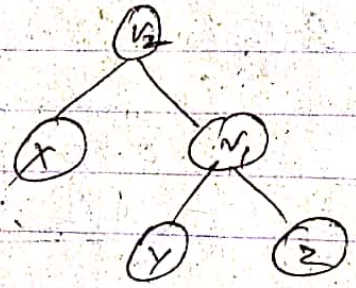
Insert in AVL :



Δ → denote subtree

case-1: Insert happened in the left subtree of the left child of α.



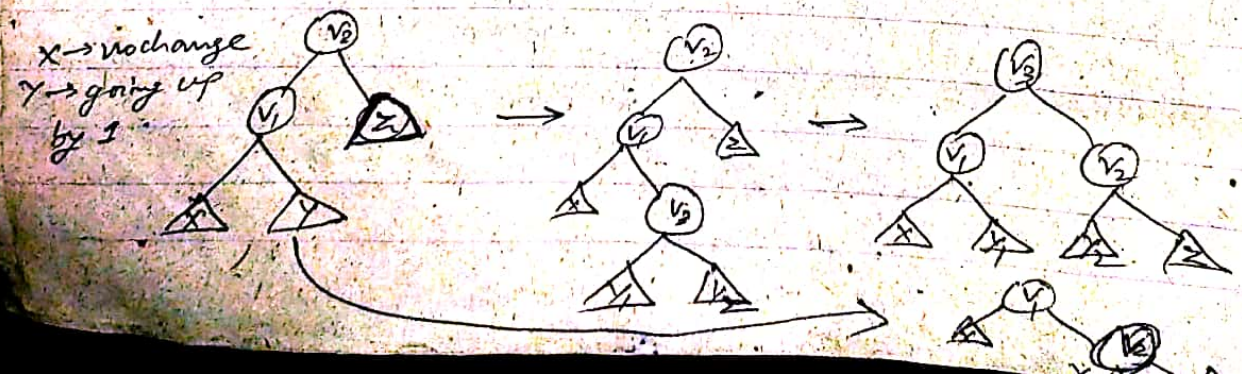


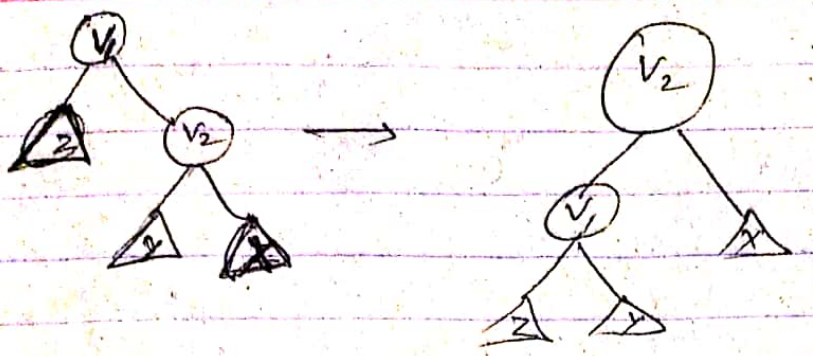
Case! Insert happened in the right subtrees of the right child.

30/08/19

AVL Tree :

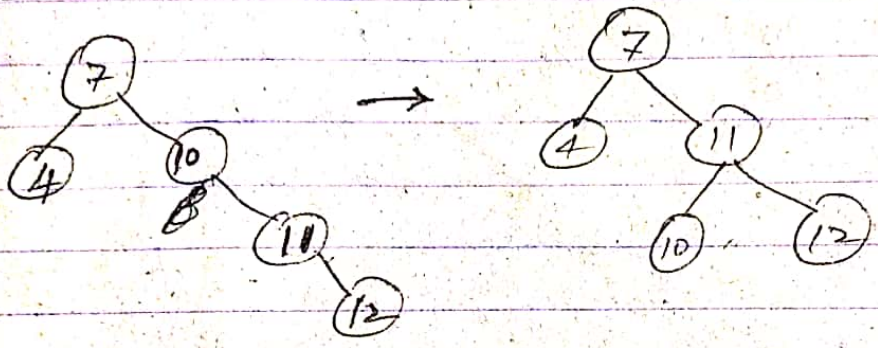
case-2 : Insert happened in the right subtree of left ch



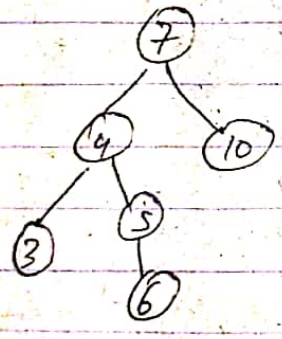


$V_1 = 10$
$V_2 = 11$ $Y, Z = NULL$
$X = 12$

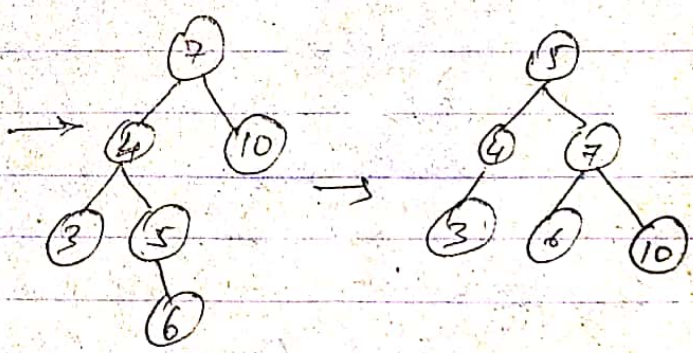
Single rotation



Case - 3



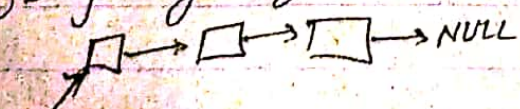
$V_2 = 7$
$V_1 = 4$
$X = 3$
$V_3 = 5$
$Y_1 = NULL$
$Y_2 = 6$
$Z = 10$



Some problems:

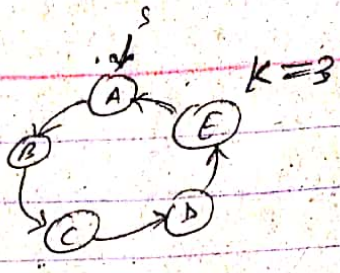
LIST:

① Self-adjusting list



insert → only in the beginning
find → make it the first node.

(2.) "_____ suicide"

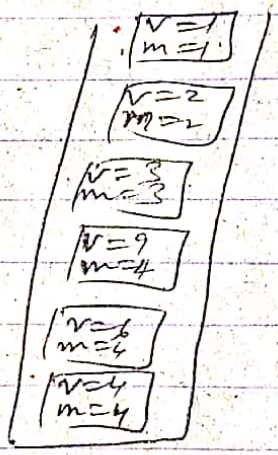


(3.) Stack → Push, Pop, find Min

```
typedef struct node
{ int val;
  struct node *prev;
} node
```

O(1)

(4.) non-leaf
node, # leaf node



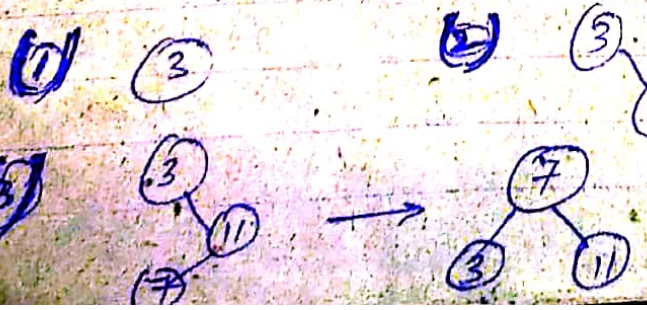
No. of non-leaf nodes
No. of leaf nodes
in BST

09/09/19

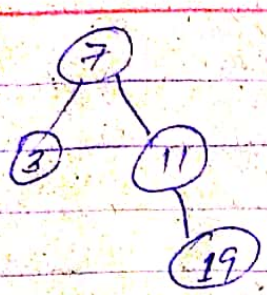
AVL Trees
- fast searching (in all cases)
- fast sorting

Idea:
- Insert the number in an AVL Tree
- Inorder traversal ≡ Ascending order

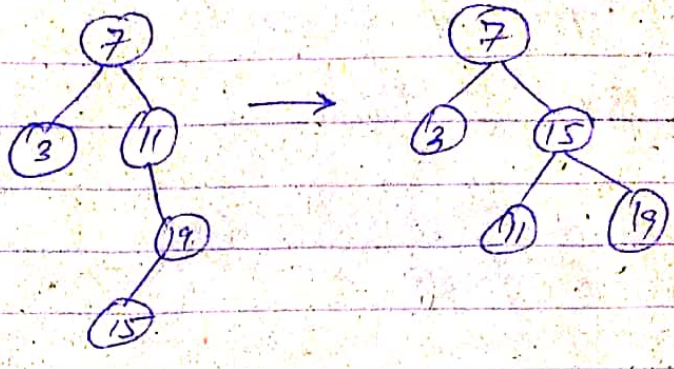
Eg: 3, 11, 7, 19, 15, 9



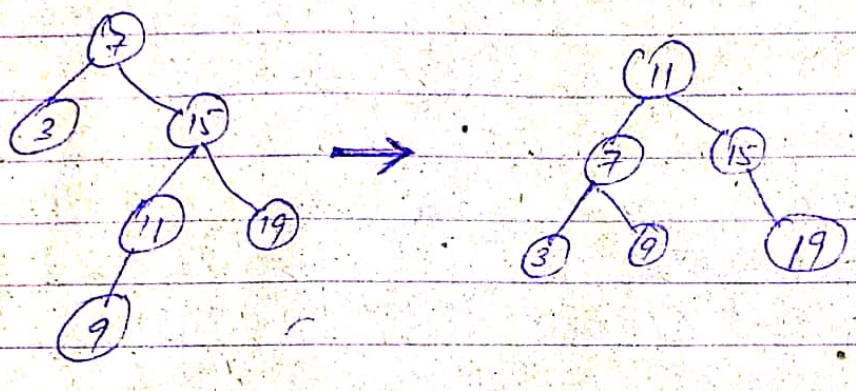
(4)



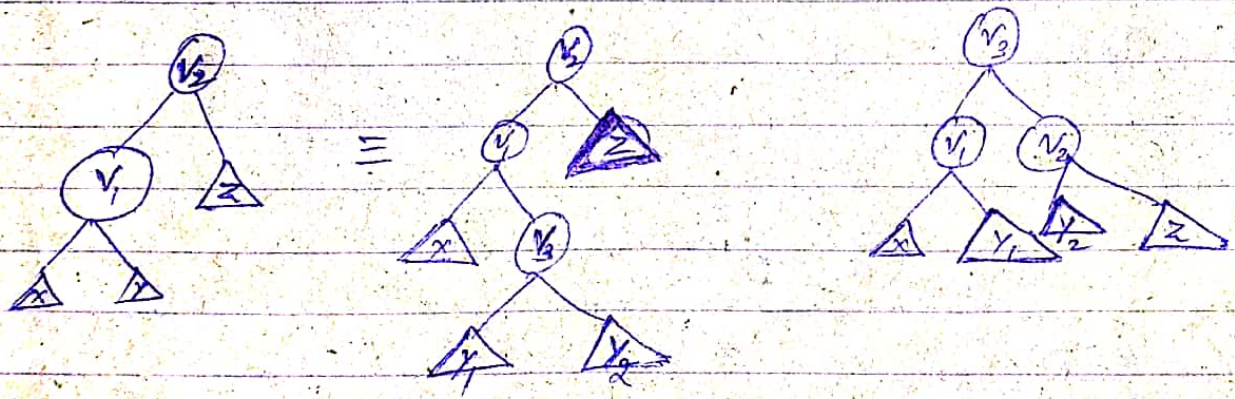
(5)



(6)



Fact: Any comparison based sorting algorithm takes at least $c.n\log n$ steps.

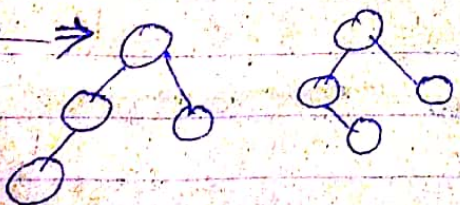


AVL Trees:

(1) $h=0 \Rightarrow$ 0 (1 node)

(2) $h=1 \Rightarrow$ 0 0 (2 nodes)

(3) $h=2 \Rightarrow$ (4 nodes)



} minimum

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

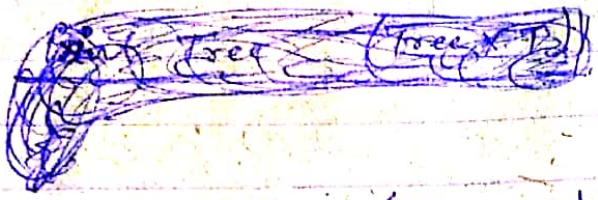
no. of nodes ← H → Height

Leaf node → which nodes have no child

No. of nodes ↑ → Height

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

$N(10) = 232$



```
Print Tree (Tree *T)
{
  Tree * temp ;
  if (T != NULL)
    print (T->item)
  temp = T->firstchild ;
while (temp != NULL)
  {
    Print Tree (temp) ;
    temp = temp -> nextsibling ;
  }
```

09/09/19

Lab :

- BST
- Height of a BST
- Preorder/Inorder/Postorder traversals
- Selfadjusting list
  - using array
  - using node
- #Node and #Leaf nodes in a BST
- "findMin" in stack in O(1)

- Stack using { array, node }
- Queue using { array, node }
- Variants of queue
- List using { array, node }
- Variants of list
  - Doubly, Circular, ----
- Maze problem

Assignments:
- Convex Hull, Substack, Circular Queue using rear/front

Upto Midsem-1

→ - Insert is always at the beginning.
- "Find" brings that element at the beginning, w/o changing the order of rest.

11/09/19

Complexity

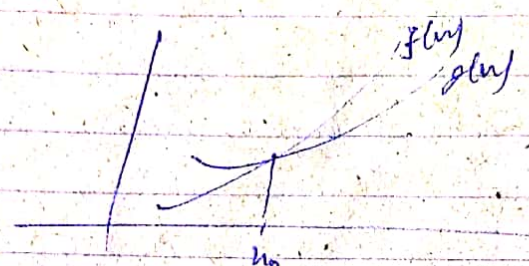
$g(n)$, $f(n)$

$g(n) = O(f(n))$

{ $\Rightarrow$ growth rate of $f(n)$ > growth rate of $g(n)$ }

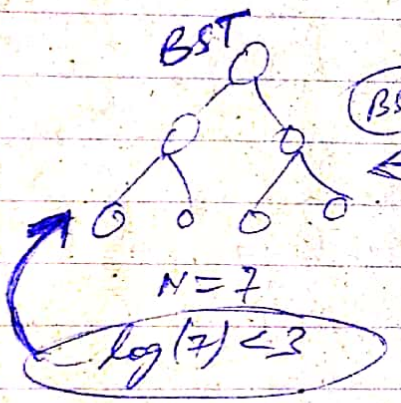
$\equiv \exists\ c, n_0 > 0$ such that for all $n > n_0$, $g(n) \leq c f(n)$

$\equiv g(n)$ is upper bounded by $f(n)$



N^2 N

$n_0 = 1$



BST $\leftarrow$ O(log(N))

list $\rightarrow$ O(N)

Queue $\rightarrow$ O(1)

Constant

faster access than list

BST $\leftarrow$ Any time O(logN) only if All trees are equally liked

Worst case is bad $\rightarrow$ To avoid this, we move to AVL

$\rightarrow$ "Near balanced" ; depth O(log

always

Every operation takes O(logN) time in AVL trees

for a sequence of M operations, the complexity is O(M

$\rightarrow$ "Amortised O(logN)"

i.e. O(M(logN))

Splay Tree

Splay trees are amortised $O(\log N)$

After every operation, the tree is "splayed"

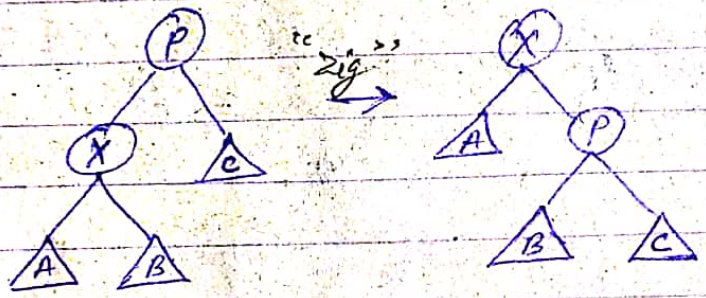
≡ Access a node and bring that node to the root while maintaining BST property.

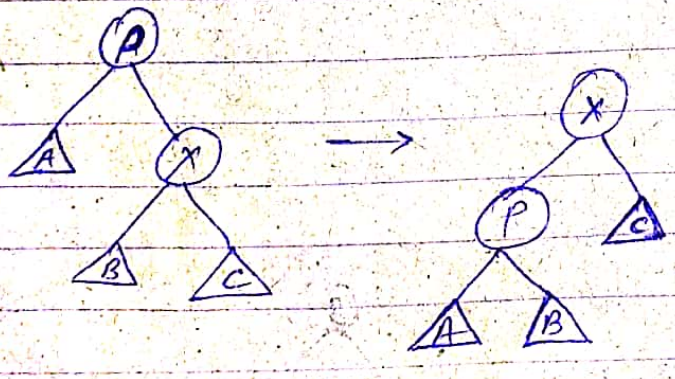
Case 0: Access is at root node — do nothing.

Case 1: - Access is at root's child.

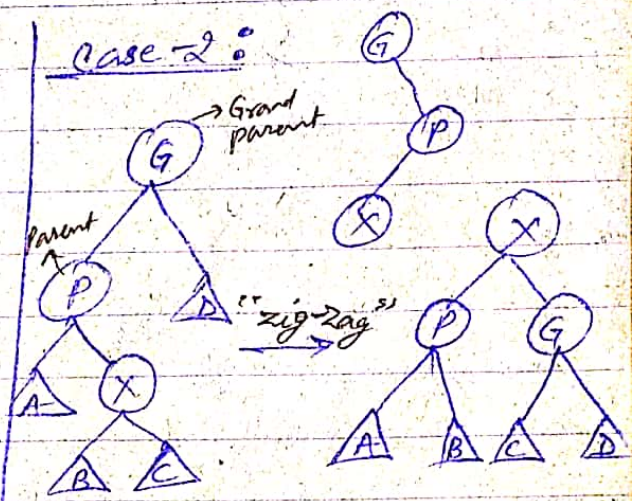
↳ Root's child becomes the root keeping the BST property intact.

- "Zig" operation (≡ single rotation in AVL)





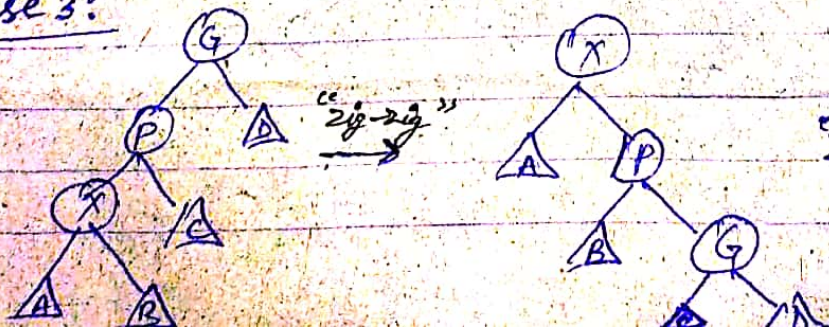
Case 2:



"zig-zag" operation

AVL double rotation

Case 3:



"zig-zig" operation

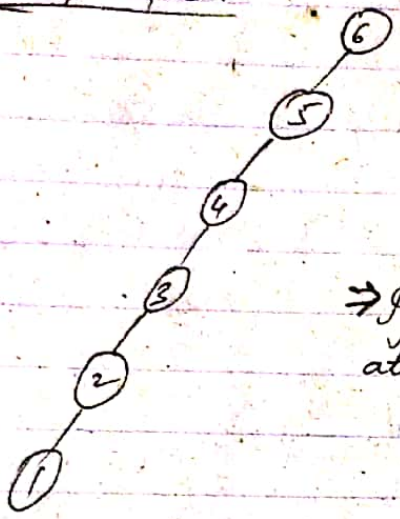
11 12/09/19

zig-zig mean / or \

zig-zag means < or >

Splay Tree

Example:



After every access perform "splay"

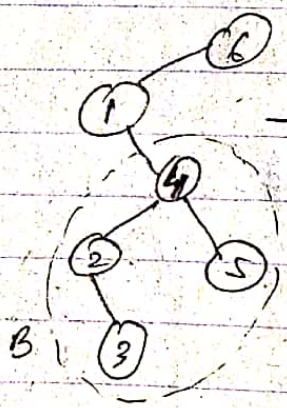
⇒ Move 'x' to root via a sequence of zig-zig/zig-zag operations + ≤1 zig ope

⇒ If depth is odd, we will use a zig ope at the last step.

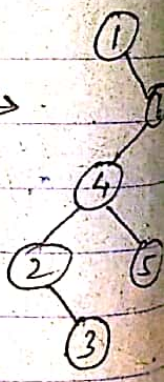
Access '1':

zig-zig

zig-zig

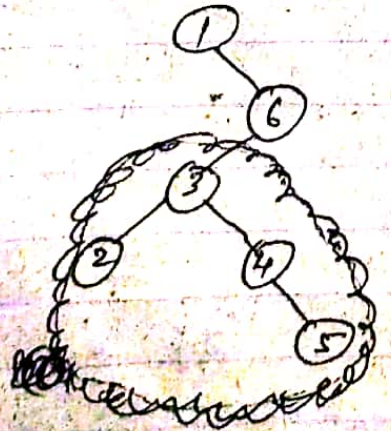


zig

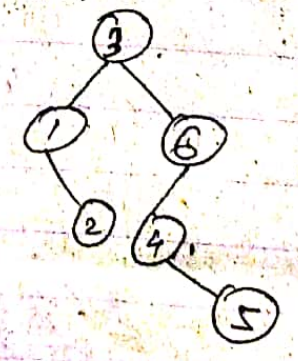


Access '3':

zig-zag

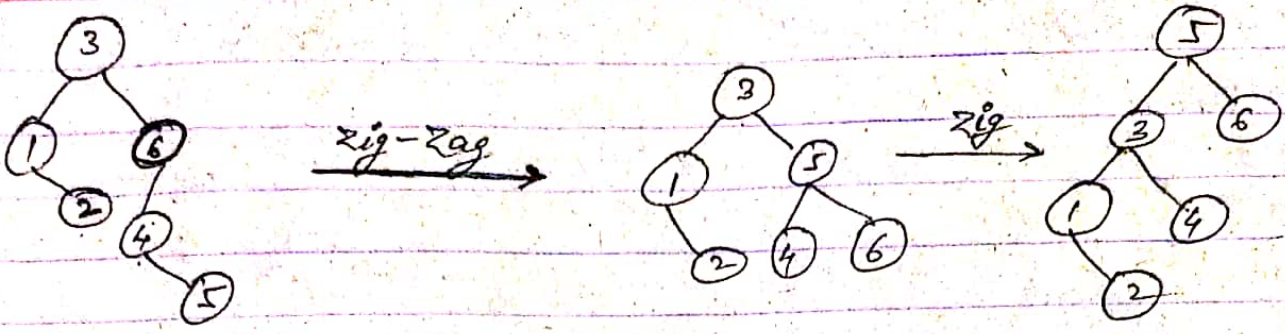


zig-zag

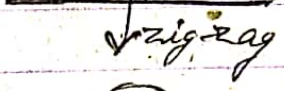


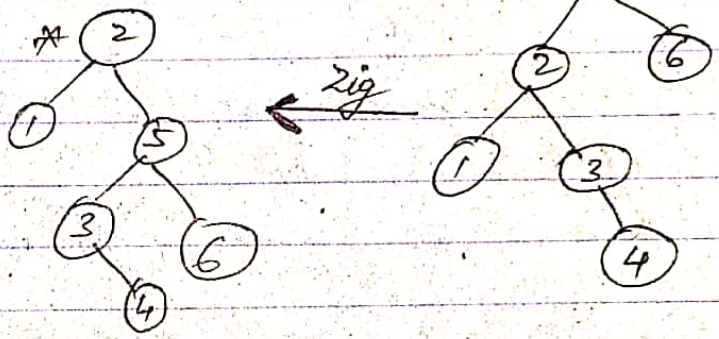
13/09/19

Access '5'

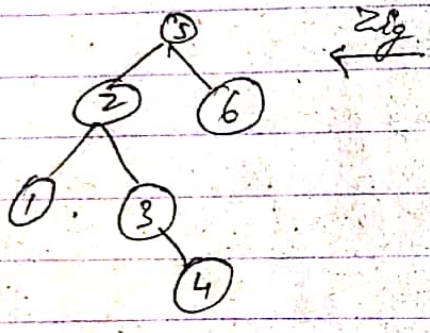


Access '2'



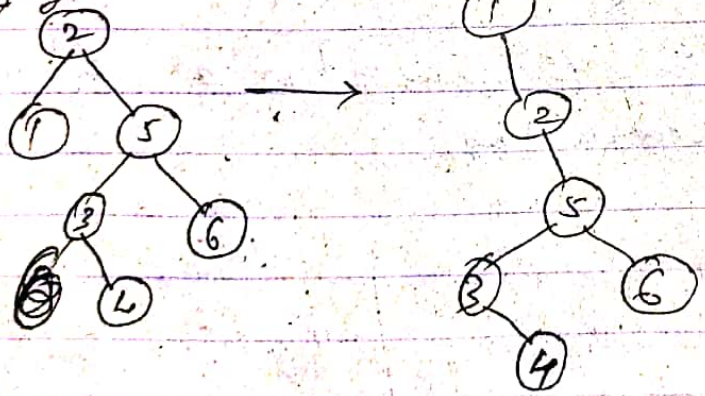


Access '5'



zig-zig
zig+zig

Access '1'

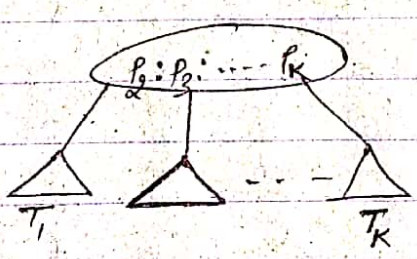


Here splaying has resulted in a worse (less balanced) BST.

* following a very "cheap" access, splaying can make things worse.
* following a very "good" (costly) access splaying makes things better

B-Tree (of order 'm')

- Root has 2 to m children
- Every internal node (other than root) has $\lceil \frac{m}{2} \rceil$ to m children
- All leaves are at the same depths
- All values are stored only at leaves.
- Every leaf stores $\lceil \frac{m}{2} \rceil$ to m values.

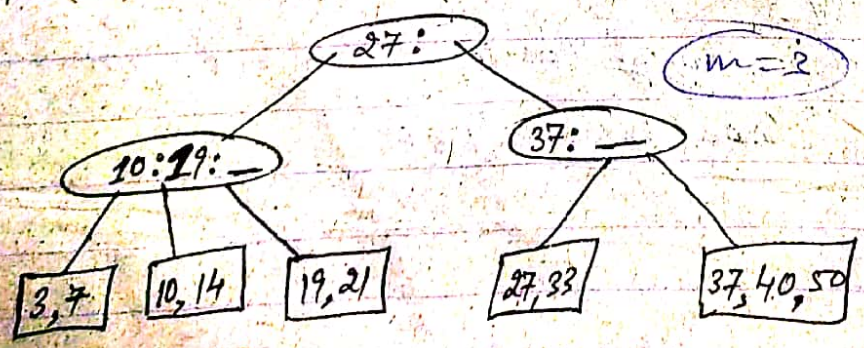


P_2 → smallest value in T_2 the same tree and so on

If $P_{i-1} \leq x < P_i$ → search T_{i-1}

$\lceil \frac{3}{2} \rceil = 2$ ceil operator

Ex: B-Tree of order 3
[(2-3) -tree]



if m=4
[(2-3-4) -tree]
$\lceil \frac{4}{2} \rceil = 2$

e ' ' implies null pointers.
treat every node as a sorted list.

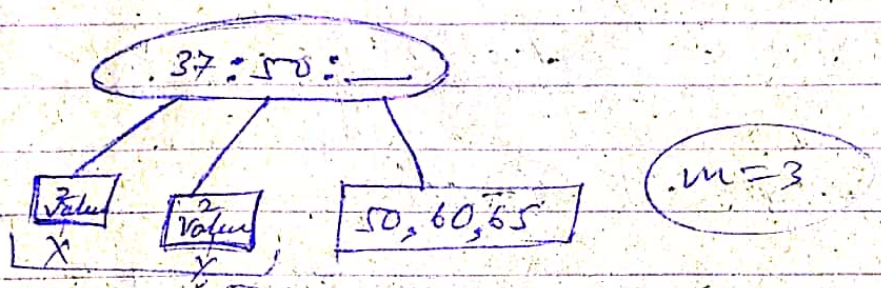
16/09/19

B-Tree (of order 'm')

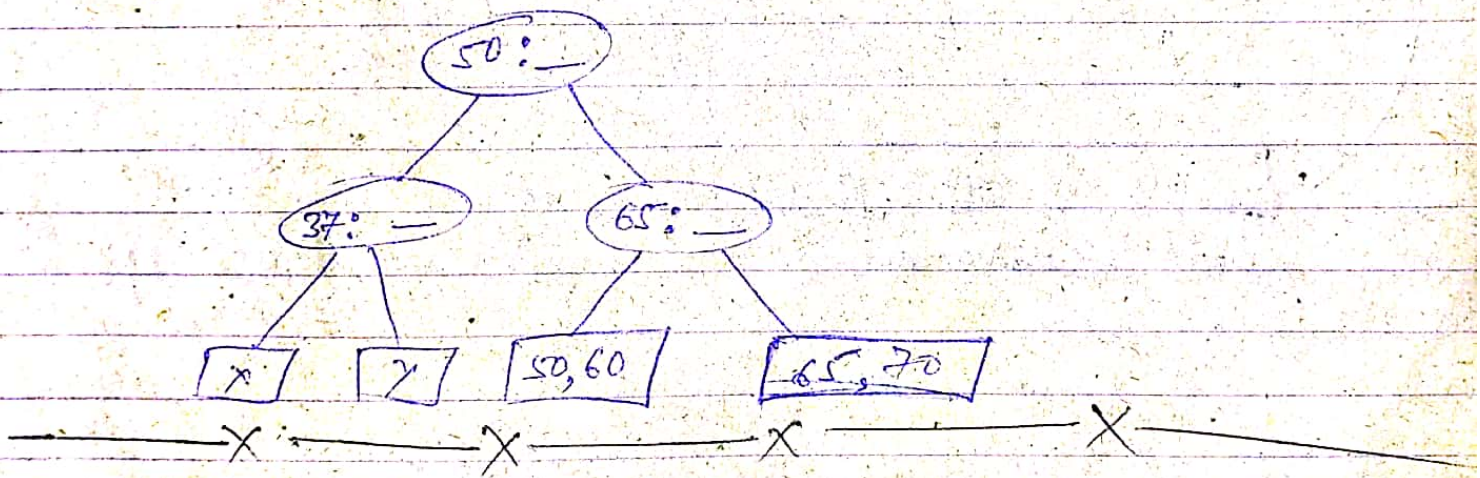
Time complexity : $O(m) \times O(\log_m N) + O(m)$

$= O(m \log_m N + m)$ (overall)

Insert 65

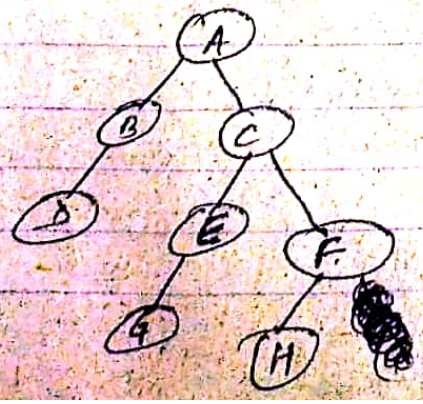


Insert 70



Lab

- Level order traversal in BST



→ ABCDEFGH

- AVL Tree (Update "Insert" & "Delete" functions of BST)
- Reorder
- Inorder
- Postorder

---

(1) Level-order traversal in BST is like maze problem

Q print even alegueue elements

(2) use height(T) function.

A	B	C	D	E	F	G	H
2	1	1	0	2	1	1	0

Code (in C++)

```
// Print nodes at a given level
void printGivenLevel (struct node* root, int level)
{
  if (root == NULL)
    return;
  if (level == 1)
    printf ("%d ", root -> data);
  else if (level > 1)
  {
    printGivenLevel (root -> left , level -1);
    printGivenLevel (root -> right , level -1);
  }
}
```

// function to line by line print order traversal a tree

```
void printLevelOrder(struct node* root)
{
    int h = height(root);
    int i;
    for (i=1; i<=h; i++)
    {
        printGivenLevel(root, i);
        printf("\n");
    }
}
```

2.

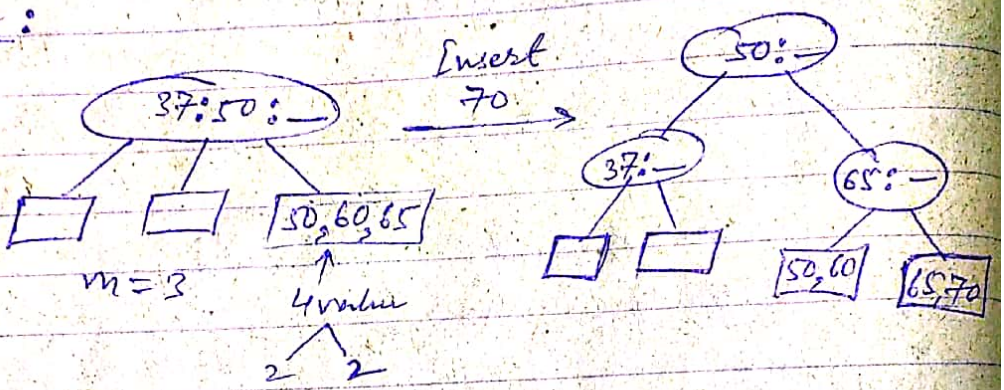
```
#include <stdlib.h>
#include <stdlib.h>

struct Node                    // An AVL tree node
{
    int Key;
    struct Node *left;
    struct Node *right;
    int height;
};
int max(int a, int b);         // A utility function to get max of two integers
int height(struct Node *N)     // A utility function to get the height of tree
{
    if (N==NULL)
        return 0;
    return N->height;
}
```

```
int max(int a, int b)   // utility function to get max of two integers
{return (a>b) ? a:b;
}
```

18/09/19

B-Tree :



For AVL, $O(\log_2 N)$

For B-Tree, $O(\log_m N)$

Page fault

Insert Delete Find findMin findMax	Linked list → slow (linear) BST → worst case slow Expected log time (when all trees are equally likely) AVL → Worst case log time Splay tree → worst case amortized log time

→ If we want only the first three functions & don't mind if remaining take more time then we use Hash tables → expected constant time

Hash Table :

Hash function
Input → element
Output → index of table

H(x) = (x) mod (Table size)

7
10
20

19/09/19

B-Tree

As 'm' increases it gets worse than BST } $m \leq$ block size

common B-Trees :

2-3, 2-3-4

Usually $m = 3/4$

$O(m) * O(\log_m N) + O(m)$

$= m \times \frac{\log_2 N}{\log_2 m}$

Hash Table :

↓

Uses a Hash function,

Input → element

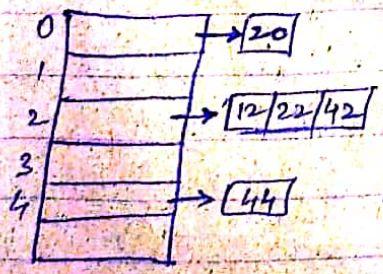
Output → index of table

→ x mod (Table size)

Ex :

Collision resolution (How to handle collision ?)

(1.) Separate chaining - Hash table implemented as a table of linked list ⇒ Treat each index of table as handles of a linked list.



Hash Table = Array of linked lists.

If we insert '22' again, we can define in the structure something to take case of multiplicity.

Load factor

$$\lambda = \frac{\text{Total no. of elements}}{\text{Table size}}$$

On an average, each list will have λ elements

$\frac{35}{10} = 3.5$

Complexity: Insert: $O(\lambda)$
(Expected time) Delete: $O(\lambda)$

find $\begin{cases} 1+(\lambda/2) \rightarrow \text{successful} \\ O(\lambda) \rightarrow \text{unsuccessful} \end{cases}$

Linear Probing

```
i=0
while(Not free)
{
  index = (H(x)+i^2) mod 10;
  i++ ; //wrap around
}
```

0	20
1	31
2	40
3	
4	34
5	24

x=40
x=34
x=22

Q. Start (Insert 10 numbers) in the order
29, 11, 30, 9, 19, 10, 49, 23, 17, 99

$H(x) = x \% 10$

Table size = 10

```
i=0
while {not free}
{ index = (H(x) + i^2) mod 10 ;
i++ , // wrap around
}
```

20/09/19

Linear Probing

```
i=0
while (not free)
{
  index = H(x) + i
  i++ ; // wrap around.
}
```

↳ Here, $F(i) = i$

In general,
Start with $H(x)$ $\xrightarrow[\text{go to}]{\text{If not possible}}$ $H(x) + F(i)$ till empty cell found.

How many probes are expected before finding an empty slot?

→ Number of

α → Load factor

↳ $\frac{\#\text{ elements}}{\text{Table size}}$ $\quad \alpha \in [0,1]$

Table size $* (1-\alpha) \in [0,1]$ → Probability that we hit empty

$$\boxed{\frac{1}{1-\alpha}}$$

$\alpha = 0.2$	$\frac{1}{0.8} = 1.$
$\alpha = 0.5$	$\Rightarrow 2$
$\alpha = 0.8$	$\Rightarrow 5$

Linear Probing

Successful find : for this, the time would dep on the load factor that was existing at time of inserting of that element.

Let,

'y' is the load factor when 'x' was inserted.

$$\left(\frac{1}{\alpha}\right) \int_0^{\alpha} \frac{1}{1-y}\, dy = \boxed{\frac{1}{\alpha} \ln\left(\frac{1}{1-\alpha}\right)}$$

↑ Normalization factor

$\left(\frac{1}{\alpha}\right)$ → No. of probe occupied

1. Separate chaining (Linked List → slow)
2. Open Addressing
   - Linear Probing → $H(x)+i$ $\quad (i=0,1,2$
   - Quadratic probing → $H(x)+i^2$ $\quad (i=0,1^2$
   - Double Hashing

If table size(T) is a prime no. and if $\lambda < 0.5$, then insert always succeeds

for $0 \le i, j \le T/2$ & $i \neq j$, $(H(x)+i^2) \bmod T \neq (H(x)+j^2) \bmod T$

(first $T/2$ probes are all distinct)

23/09/19

Open Addressing:

- Linear Probing
- Quadratic Probing → If table size(T) is prime and $\lambda < 0.5$, then insert always succeeds
- Double Hashing

Double Hashing

$(H(x) + iH'(x)) \bmod (\text{Table size})$

Ex- $H(x) = X \% 10$

$H'(x) = X \% 7$

Table size = 10

Insert: 29, 11, 30, 9, 19, 10, 49

0	30
1	11
2	
3	9
4	19
5	
6	10
7	
8	
9	29

To avoid this, we make two changes:

(1) $H'(x)$ should be such that $H'(x) \neq 0$ always.

Ex - $H'(x) = R - (X \% R)$

(2) choose table of prime size.

Ex. $H(x) = x \% 7$

$H'(x) = 5 - (x \% 5)$

Insert → 29, 11, 30, 9, 19, 10 in a table of size of 7

0		
1	29	29
2	30	30
3	9	9
4	11	11
5	19	19
6	10	

Rehashing :

- Choose a new hash table of prime size double the current size (Prime no. greater than two times the current table size)

- Transfer one by one

current T = 7

New T = 17

R = 13

$H(x) = X \% 17$

$H'(x) = 13 - (X \% 13)$

Rehashing takes place when
- it crosses a cut-off (eg: 0.5)

λ → Load factor
T → Table size

- when some insert fails
Time required for rehashing : Add. Told

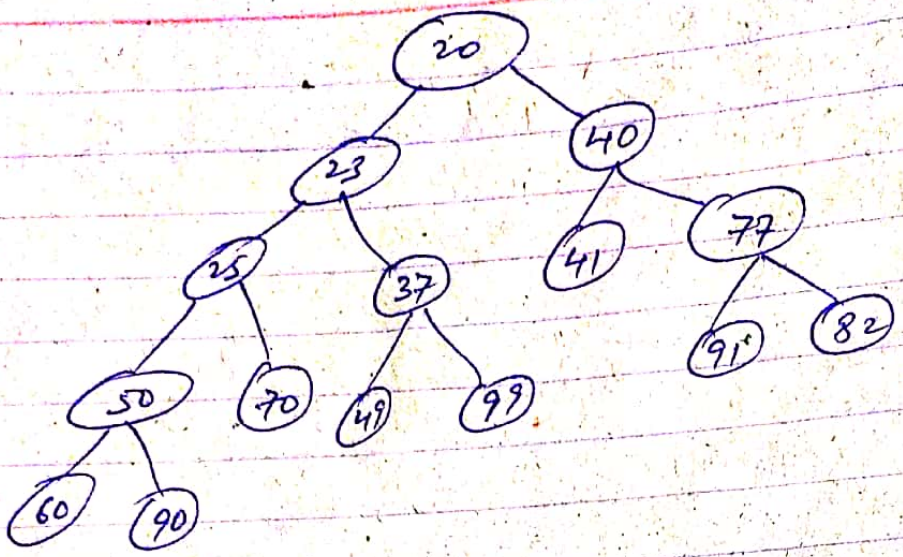
When we use
Linear probing - it's refessred as "primary clustering"
Quadratic " - " " "secondary clustering"

Priority Queue :
In Priority Queue, "Enqueue" is the same as in queue, but "Dequeue" is on the basis of priority.
Dequeue ⇒ Delete Min

Implementation :
Done using a "Binary Heap"
(the term "Heap" generally mean Binary Heap)

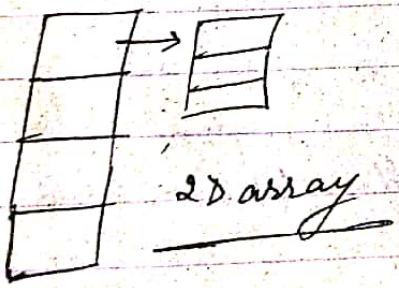
Structure property
Binary Tree is balanced as much as possible (all higher levels are completely balanced). In the last level, fill from left to right.

→ Order Property
What is stored in every node should be smaller than what is stored in sub-tree rooted below.



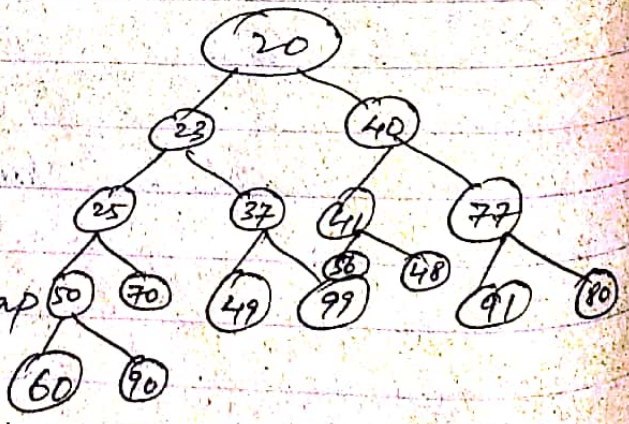
Lab:

- AVL Tree (Update "Insert" & "Delete" operations of BST)
- Splay Tree (Update "find" operation of BST)
- Hash Table (constant time for Insert/Delete/find)
  - Basic Version ($H(x) = X \% T$)
  - Linear Probing, Quadratic Probing, Double Hashing
  - As a table of Linkedlist/BST/Hashtables.



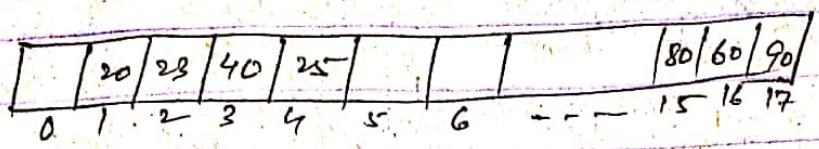
25/09/19 **Priority Queue:**

Enqueue Delete Min



Implemented using Binary Heap
- Structural Property
- Order Property

Array Representation



≡ Level traversal

for index "i" : Left child → $2i$

Right child → $2i+1$

for "i" element : Parent → $\lfloor i/2 \rfloor$

Insert: "Percolate up"

↳ swap with parent

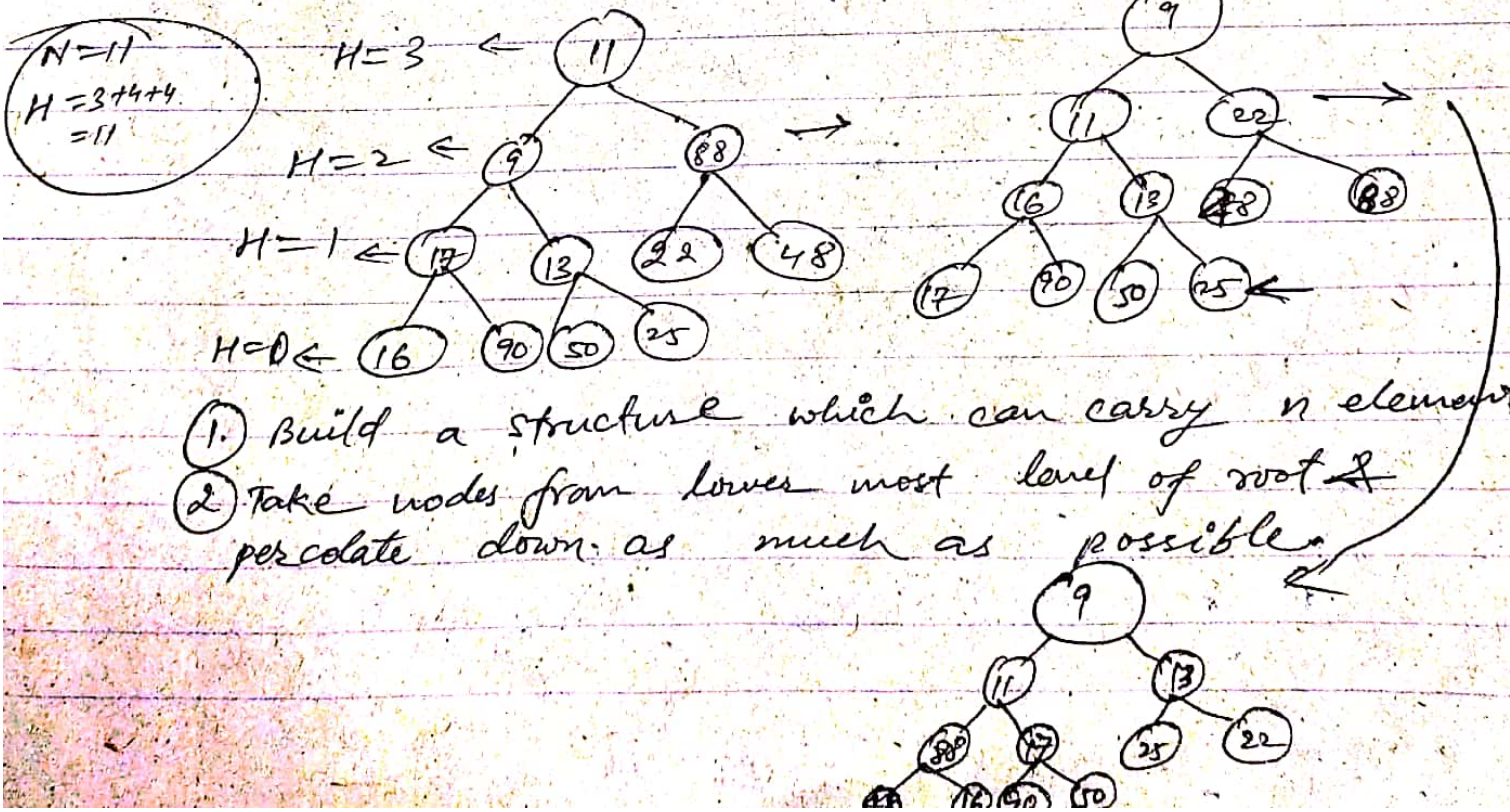
DeleteMin: "Percolate down"

↳ find min of the two children & swap.

Build Heap:

Given 'n' elements, build a heap.

11, 9, 88, 17, 13, 22, 48, 16, 90, 50, 25



1. Build a structure which can carry n elements.
2. Take nodes from lower most level of root & percolate down as much as possible.

Total complexity:

- Max no. of percolations each node can do = Height of that node

Complexity = sum of heights of all node

$O(N)$ $\quad N \leq 2^{H+1} - 1$

Ex- Previous exp - $3+4+4 = 11$

$N = 11$

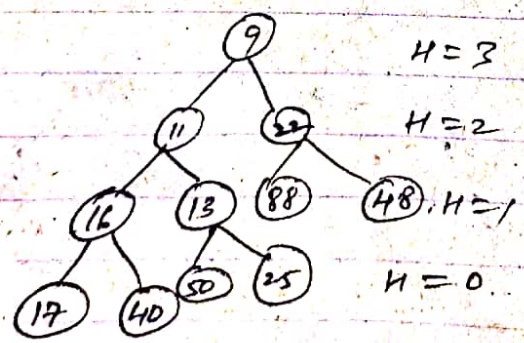
$$\text{Sum} = \sum_{i=0}^{H} 2^{(H-i)} \cdot i$$

26/09/19

Build Heap:

Step-1: Build a structure which can carry the given data

Step-2: Takes nodes from lower most level to root & percolate down as much as possible.



Maximum no. of

Q. 90, 70, 79, 84, 5

H=3, $N^{max} = 15 = 2^{H+1} - 1$

$N \leq 2^{H+1} - 1$

Complexity = Sum of heights of all nodes.
↳ of "Build Heap"

$$N \leq 2^{H+1} - 1$$

Number of nodes at height i → 2^{H-i}

$$\text{Sum} = \sum_{i=0}^{H} (2^{H-i}) \, i$$

$$\text{Sum} = 0 \cdot 2^H + 1 \cdot 2^{H-1} + 2 \cdot 2^{H-2} + 3 \cdot 2^{H-3} + 4 \cdot 2^{H-4} + \cdots + (H-1)2^1 + H \cdot 2^0 \quad \text{— (1)}$$

$$2 \times \text{Sum} = 2^H + 2 \cdot 2^{H-1} + 3 \cdot 2^{H-2} + 4 \cdot 2^{H-3} + 5 \cdot 2^{H-4} + \cdots + (H-1) \cdot 2^2 + 2 \cdot H \quad \text{— (2)}$$

(2) — (1)

$$\text{Sum} = 2^H + 2^{H-1} + 2^{H-2} + \cdots + 2^1 - H$$

$$= (2^{H+1} - 2) - H$$

$$= (2^{H+1} - 2) - H + 1 - 1$$

$$= (2^{H+1} - 1) - (H+1)$$

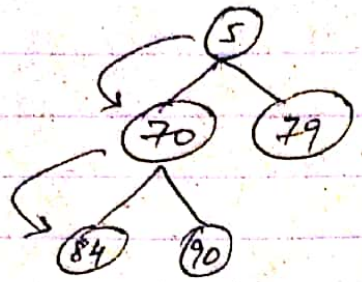
$$= O(2^H) \text{ or } O(N)$$

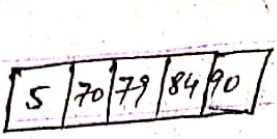
$\left[\because 2^H - 1 \leq N \right]$

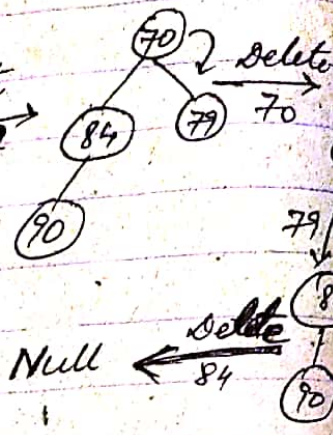
90, 70, 79, 84, 5

Build Heap → $O(N)$

Sequence of deleteMin → $O(N \log N)$







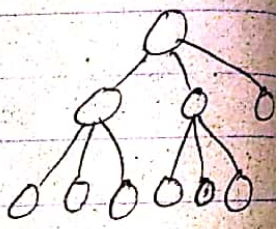
$H = \log N$

---

d-Heaps :

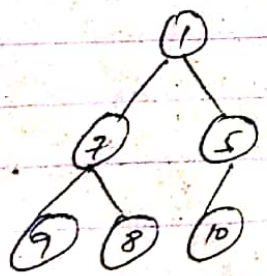
d = 3

Binary Heap III 2-Heap	d-Heap Exactly "d" childsen (last node(s) exempted)

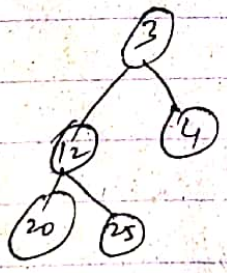


Insert → O(log

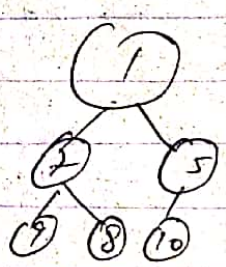
1.

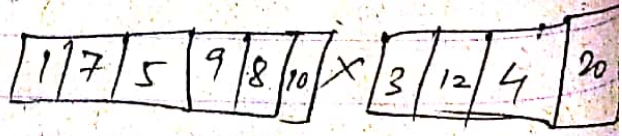


Primary



Merge





① Dump
② Percolate down → Build Heap

27/09/19

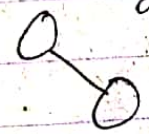
Binary Numbers:

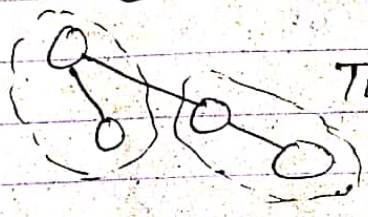
17 . 10001

Binomial Heap:

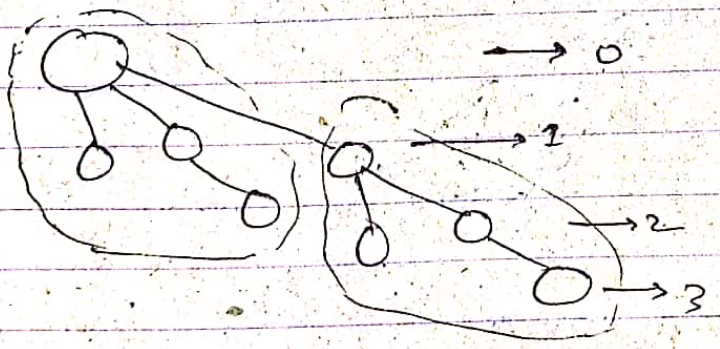
Collection of trees — each tree is a Binomial Tree

B_0 → has only one element (that is root) 0.

B_1 →  Two B_0's

B_2 →  Two B_1's

B_3 → 2 B_2's

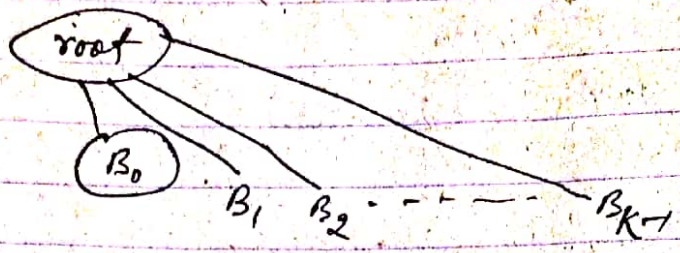


B_4 → B_3 B_3

No. of nodes in B_K → 2^K

- B_K will have levels: 0 — K (0 to K)
- No. of nodes at level d is → ^KC_d

B_K



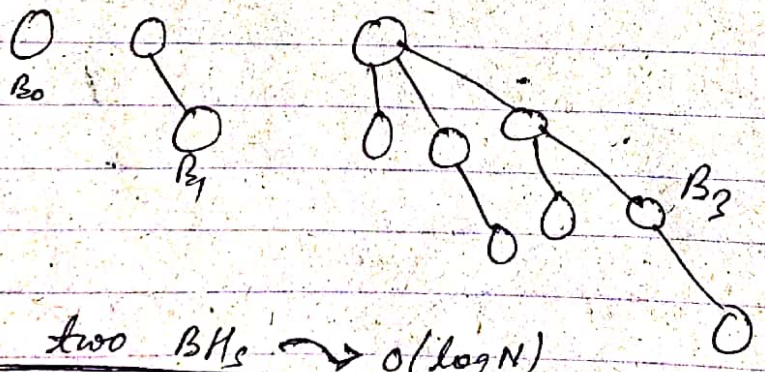
Binomial heap of 'N' nodes

Ex - N=17

1 0 0 0 1 ← Binary
4 3 2 1 0 → ∴ it has B_0 & B_4

Ex - N=11

1011 ← Binary
3210 ← ∴ has B_0, B_1 & B_3



Merging two BHs → O(log N)

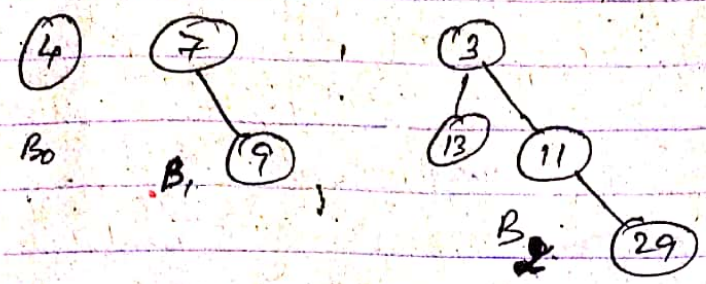
BH_1 — N_1 nodes
BH_2 — N_2 nodes
} BH — N_1+N_2 nodes

Ex - $N_1=11$, $N_2=17$
$N=N_1+N_2=28$

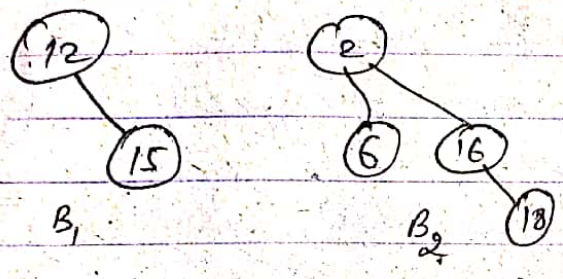
⇒ $BH_1 \rightarrow B_0, B_1, B_3$ | $BH_2 \rightarrow B_0, B_4$

$BH \rightarrow B_2, B_3, B_4$

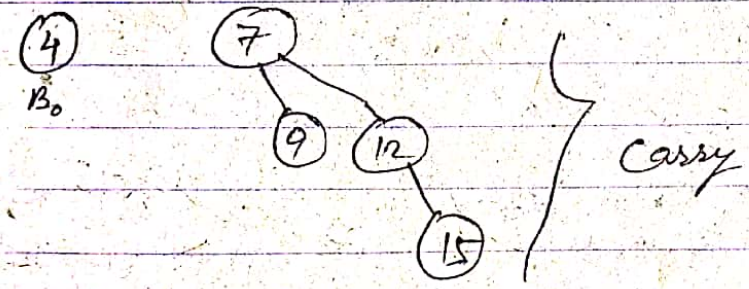
Ex – $N_1 = 7$ $BH_1 \rightarrow B_0, B_1, B_2$



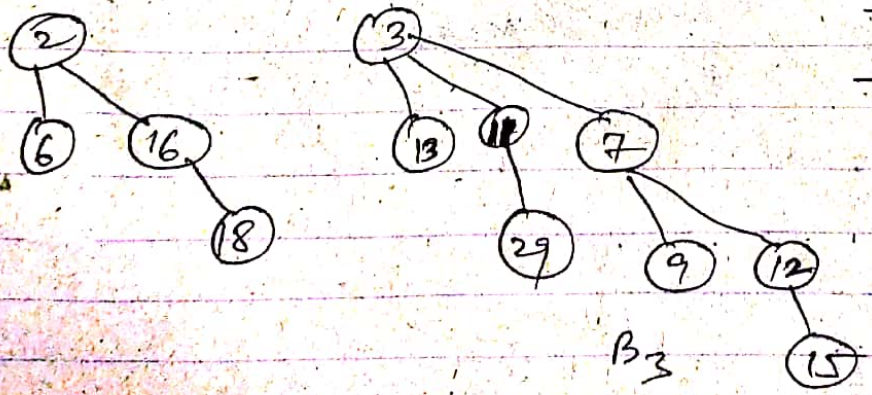
$N_2 = 6$ $BH_2 \rightarrow B_1, B_2$



$N = N_1 + N_2 = 13$



BH



111 → 7
110 → 6
1101
$B_3 B_2 B_1 B_0$ → 13

Out of the three B_2 s, retain one & merge the remaining two!

$N_1 = N$ $N_2 = 1$ → Means Inserting one values

$\equiv$ Insert

Merge is analogous to Binary addition.

DeleteMin: (a) scan all the roots (heap property is maintained in each heap)

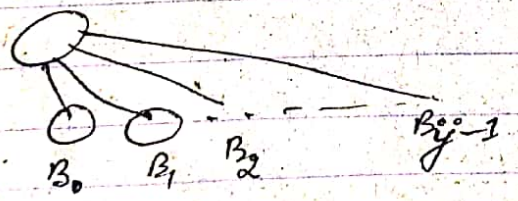
(b) Let B_{ij} be the minimum

↳ Split B_H into two parts.

B_H — $B_{i0}, \ldots, B_{ij}, \ldots B_{ik}$.

B_H — B_{ij}

B_{ij} → a root node connected to

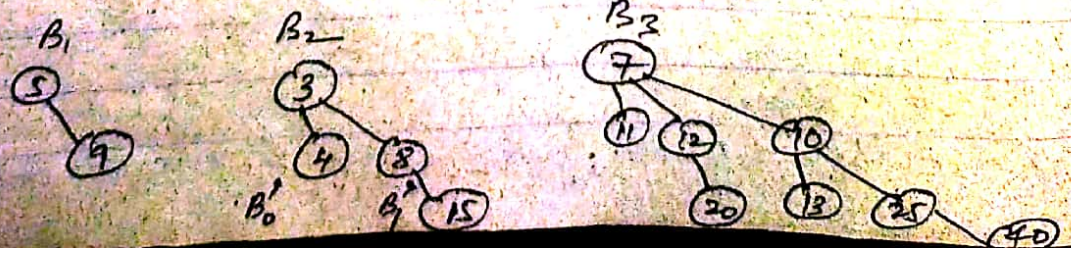


After deleting the root node:

B_H — B_{i0} B_{ik}

B_H — $B_0, B_1, \ldots B_{ij-1}$ } Merge

Ex - $N = 14$



Friday → 11:50 PM
Assignment 1

1 → List
2 → Stack
3 → Queue
4 → BT
5 → AVL

After doing deletemin (min is 3)

BH — B_1, B_3
B_0', B_1'

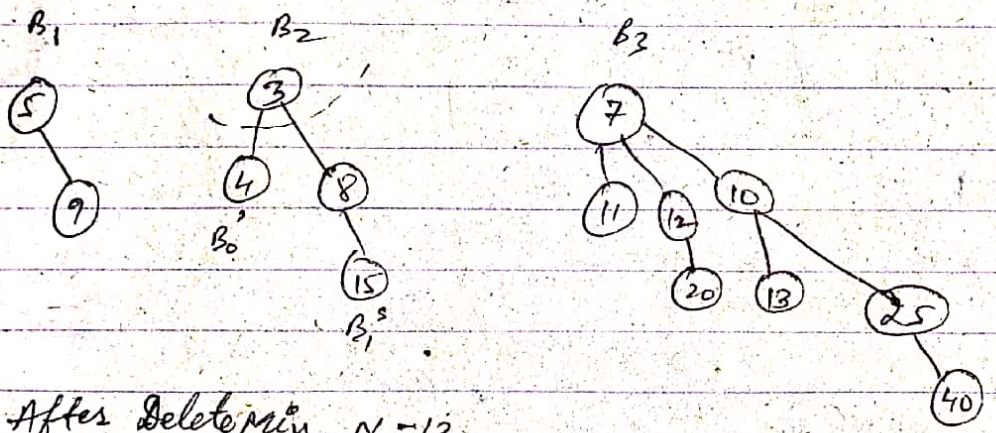
N = 13

30/09/19

Binomial Heap :

Delete Min

Eg-



After DeleteMin, N = 13

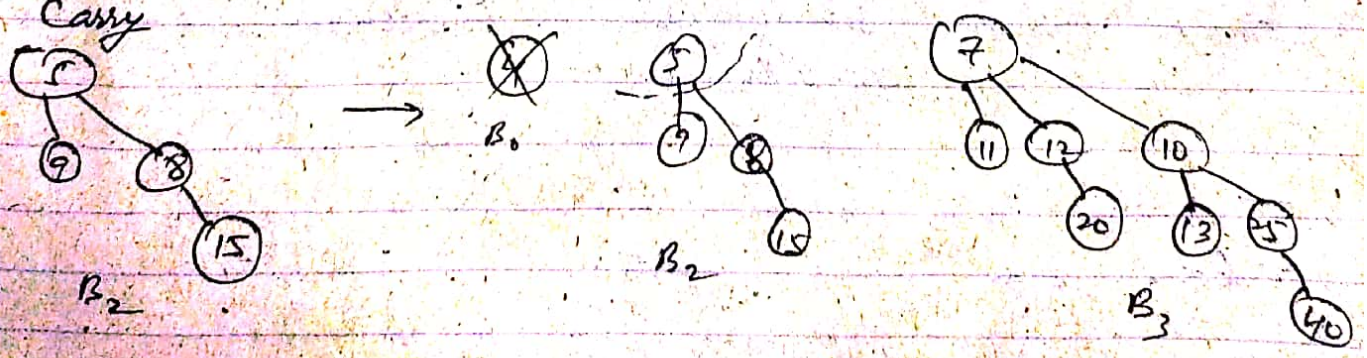
BH — B_1, B_3
B_0', B_1'

10
10
100

N = 13 ⇒ B_0, B_2, B_3

1101

$B_3 B_2 B_1 B_0$



Stack *s;
{ s.push( )
  s.pop( )
  s.top( )

Graphs

$G = (V, E)$

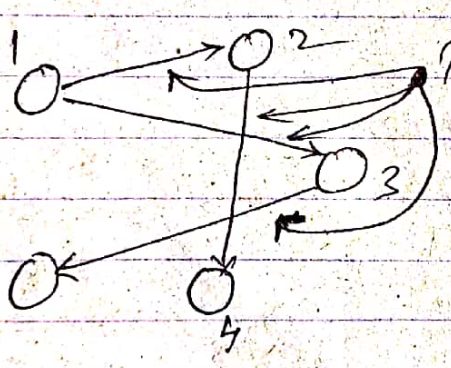
$V \rightarrow$ set of vertices/nodes

$E \rightarrow$ set of edges

$E \subseteq V \times V$

$V = \{1, 2, 3, 4, 5\}$

↳ 5 vertices/nodes



These are the edges

$E = \{(1,2), (1,3), (2,4), (3,5)\}$ 4 edges

→ A graph is called a "Directed Graph" (or Digraph) if $(u,v) \neq (v,u)$

→ Undirected graph :- $(u,v) = (v,u)$

→ Edges can have weights

Path:

from u to v.

Path is a sequence of vertices

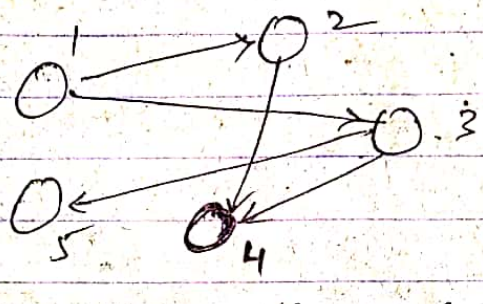
$w_1, w_2, \cdots, w_k$

such that $w_1 = u$, $w_k = v$, $(w_i, w_{i+1}) \in E$

$u = 1, v = 5$

$1, 3, 5$

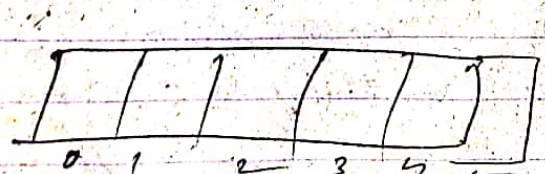
$1, 2, 4, 3, 5$

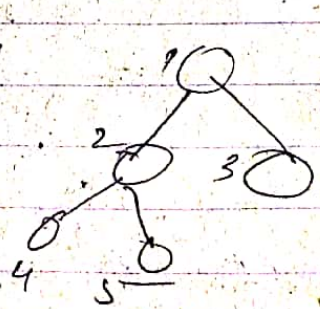


- A path is called simple if all the nodes are distinct.

Lab:

- Binary Heap
  - "Build Heap", "Insert" & "DeleteMin" operations
- Binomial Heap
  - A collection of binary heaps





$i \to 2i+1$
$2i$

04/10/19

Path :

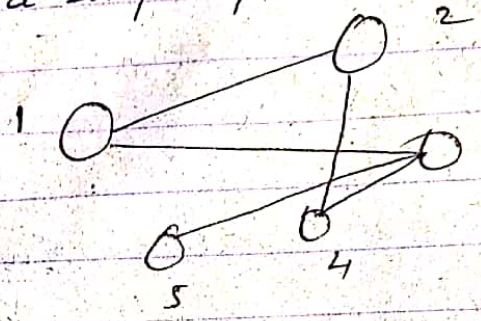
from u to v

$w_1, w_2 \dots w_k$ (a sequence of vertices)

such that $u = w_1, w_k = v$, $(w_i, w_{i+1}) \in E$

- Path is called "simple" if all nodes are distinct
- If there is some simple path from "u" to "u", then it is called "cycle". Ex - 3, 1, 2, 4, 3.

→ Ex - 1, 3, 4, 3, 5 is not a simple path.



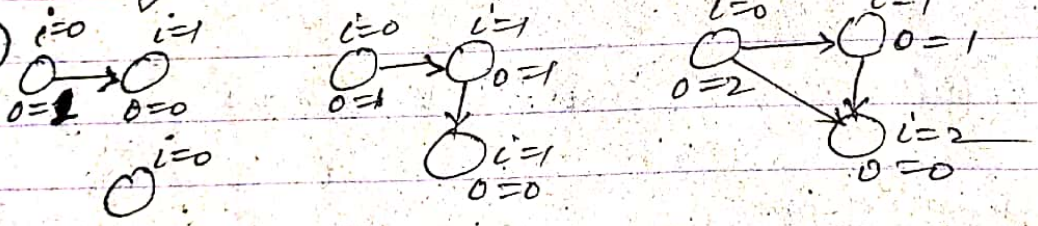
1, 2, 4

1, 3, 4

Directed Acyclic Graph (DAG)

- Has no cycle
- Each edge has a direction

(i → indegree, o → outdegree)



Q. Suppose $|V| = 3$, then what is the maximum $|E|$ such that $\exists$ a DAG $G(V, E)$.

"Indegree" of a node = # Edges going in

"Outdegree" " " = # Edges going out

Fact: In a DAG, there has to be atleast one node whose indegree is 0.

Ex - $E = \{(1,2), (1,3), (2,4), (3,5)\}$

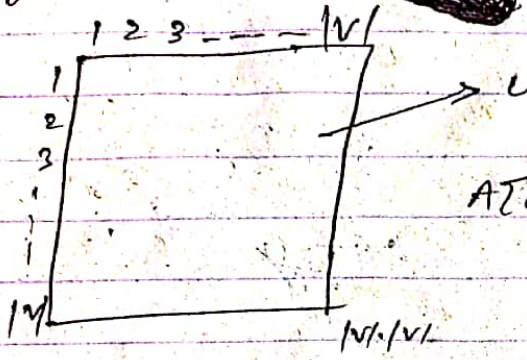
$I(1) = 0$ | 2 is a neighbour of 1.

Representing Graphs:

Naive $G=(V, E)$

store V & E sets separately

Adjacent Matrix



→ undirected matrix.

$$A[i][j] = \begin{cases} 1, & \text{if } (i,j) \in E \\ 0, & \text{if } (i,j) \notin E \end{cases}$$

fact:

If $A^k[i][j] = x$;

$\exists$ 'x' paths of length 'k' from i to j

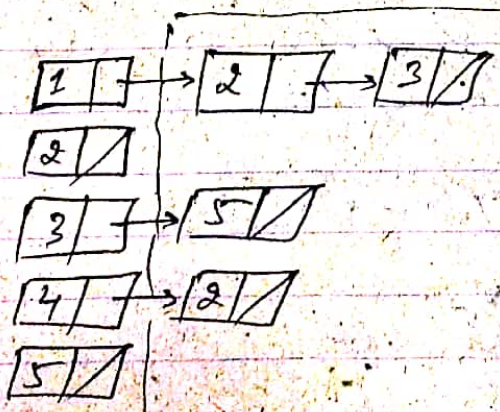
Memory used : $O(|V|^2)$

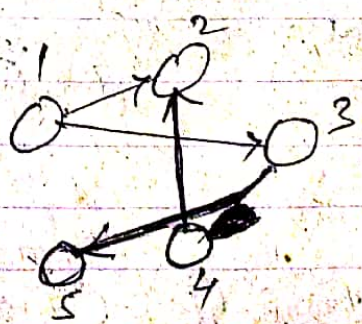
- Adjacent matrix is not recommended when the number of edges is small.

Adjacent List:

A collection of $|V|$ linked lists where the i^{th} list will have all the nodes connected to the i^{th} node.

Eg-



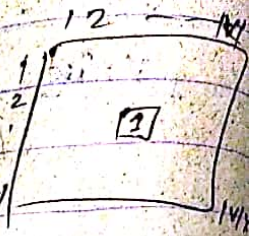


$O(|V|+|E|)$

In Adjacent Matrix:

indegree (i) = # 1's in column i

Outdegree (i) = # 1's in row i



In Adjacent list:

Outdegree (i) = length of list i

indegree (i) = [redacted] search completely

Problem: Topological sort

Given a DAG, give ordering of vertices such that if $(v_i, v_j) \in E$ then v_i appears before v_j in the ordering.

$DAG = (V, E)$

$V = \{v_1, v_2 \cdots v_{|V|}\}$

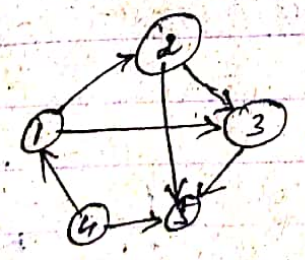
$v_{i1}, v_{i2}, \cdots v_{i|V|}$

9/10/19

Topological sort:

DAG:

Input →



Steps (Algo-1)

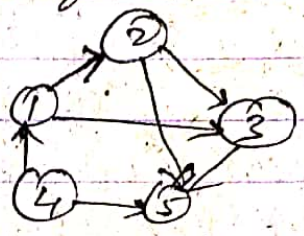
① Identify a node with indegree
② Delete that node (and corresponding edges), and recurse.

Time: $O(|V|^2)$

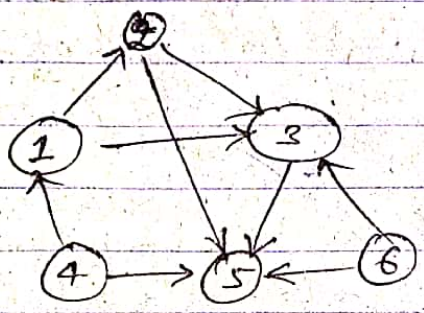
Output → 4 → 1 → 2 → 3 → 5

Steps (Algo 2: Using Queue ADT)

① Find all nodes of indegree zero & enqueue them
② v = Dequeue( )
③ Reduce the indegree of all out-neighbours by 1;



Time: $O(|V|+|E|)$



4, 6, 1, 2, 3, 5
6, 4, 1, 2, 3, 5
4, 1, 2, 6, 3, 5

} Each one of these is a permutation of the input vertices

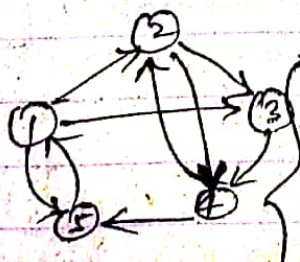
Time: $|V| + \text{outdegree}(v_1) + \text{outdegree}(v_2) + \cdots + \text{outdegree}(v_{|V|})$

$= |V| + \sum_{u \in V} \text{outdegree}(u)$

$= |V| + |E|$

Shortest path:

Given a graph $G=(V,E)$ and a designated node $s \in V$, compute the shortest path from s to all other nodes.



$s = 3$

Identify all the nodes at dist 0 from s
" " " " " 1 " "
" " " " " 2 " "
Till we have covered all the nodes
"Breadth first search"

BFS

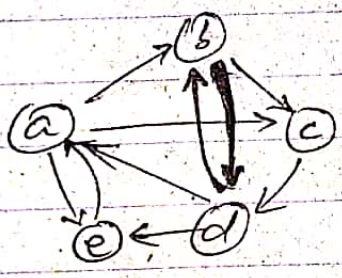
Initial state:

- All nodes are "unmasked".
- Dist. of s from s (s→s) is 0, others are ∞.

---

while (unmasked node exists)

{ - v = choose node with least 'd' (among unmasked)
- Mark v
- for all out neighbours w of v
  if (d(w) = inf.)
    d(w) ← d(v) + 1
    path(w) ← v



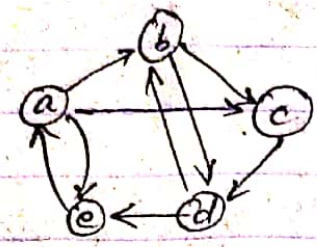
s = e* d(e) = 0
d(e) = 0
Mark(e)
d(a) = 1
path(a) = e

v = a
Mark(a)
d(b) = 2
path(b) = a
d(c) = 2
path(c) = a

Node	Mask	d	Path
a			
b			
c			
d			
e			

10/10/19

BFS



source = e

	Mask	dist	path
a	0	∞	–
b	0	∞	–
c	0	∞	–
d	0	∞	–
e	0	0	–

e —d=1→ a —d=2→ b —d=3→ d

a —d=2→ c

	Mask	dist	Path
a	1	1	e
b	1	2	a
c	1	2	a
d	1	3	b
e	1	0	

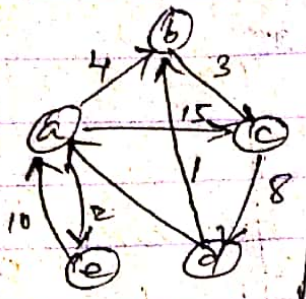
Time: $O(|V|^2 + |E|)$

v=e, w=a | v=a, w={b,c} | v=b, w={d} | v=c, w={} | v=d, w={}

Queue Method

```
Enqueue (s, Q)
while (Q is not empty)
{
    v = Dequeue(Q)
    Mask v
    for (all outneighbours w of v)
    { if (d(w) = inf && w is not masked)
      { d[w] = d(v) + 1
        path(w) = v
        enqueue (w, Q) }
    }
}
```

Dijkstra's Algo:



$c(a,b)=4$

when all weights are non-negative, we use "Dijkstra's" Algorithm.

① Among all the unmarked nodes, choose the one with minimum d.

② Mark 'v'

③ For all outneighbours w of v if $(d(w) > d(v)+c(v,w)$ & w is unmarked

$d(w) = d(v)+c(v,w)$

Mark dis path path(w)=v

a	0	∞	–
b	0	∞	–
c	0	∞	–
d	0	∞	–
e	1	0	–

	mark	dist	Path
a	ϕ1	ϕ10	e
b	ϕ1	ϕ14	a
c	ϕ1	ϕ2517	ϕb
d	ϕ1	ϕ25	c
e	ϕ1	0	

e → a → b → c → d

v=e

w={a}

w={b,c,e}

v=b

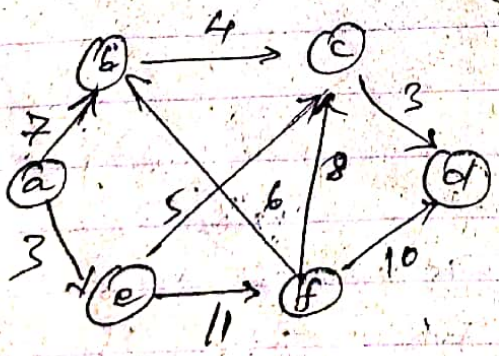
w={c}

25 > 17

v=c

w={d}

Q.



source = 'a'

Solve using table

11/10/19

Dijkstra's Algo:

Time (Naive) : $O(|V|^2 + |E|)$

Using Priority Queue : $O(|V| \log|V| + |E| \log|V|)$

Using fibonacci Heap : $O(|V| \log|V| + |E|)$

↳ fastest known implementation

- In non-negative weighted graphs : $O(|V| \log M + |E|)$ (using fibonacci Heap)

- In DAG : $O(|V|+|E|)$

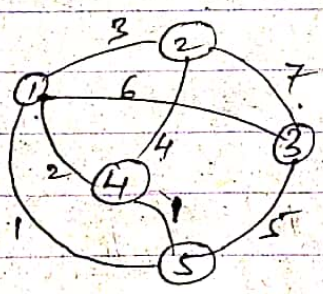
In DAG : Do Topological sort

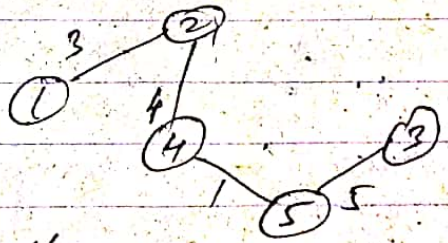
for any V_k, $d_k = \min(d_i + C_{ik})$

$\forall$ in-neighbours i's

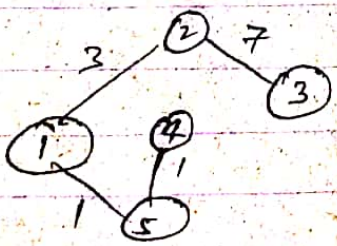
Minimum Spanning Tree (MST):

Given a weighted undirected graph G, find a spanning (i.e. cover all vertices) acyclic connected sub-graphs of G with the least cost.

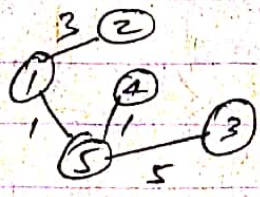




cost = 3+4+1+5 = 13



cost = 1+1+3+7 = 12



cost = 3+1+1+5 = 10

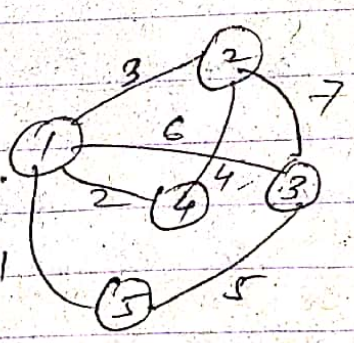
If the graph is undirected, an acyclic connected sub-graph is called tree

PRIM's Algorithm:

- choose a node (say "1") & ~~mark it~~ set its $d_v = 0$
- choose an unmarked node v with minimum d_v.
- Mask v
- Add the edge (v, p_v)
- Update distances of all unmarked neighbours w of v

$$\begin{cases} d_w = \min(d_w, c_{vw}) \\ p_v = v \end{cases}$$

Mask	d_v	p_v → "Previous"
1	1	0	1
2	+1	∞ 3	+1
3	+1	∞ 5	+5
4	+1	∞ 1	+5
5	+1	∞ 1	+1



v=1 | v=5 | v=4 | v=2 | v=3

w= 2 d_2 = | w=4 d_4 | w=2, d_2 = | w=3 d_3

w=3
w=4
w=5
w=3

18/10/19

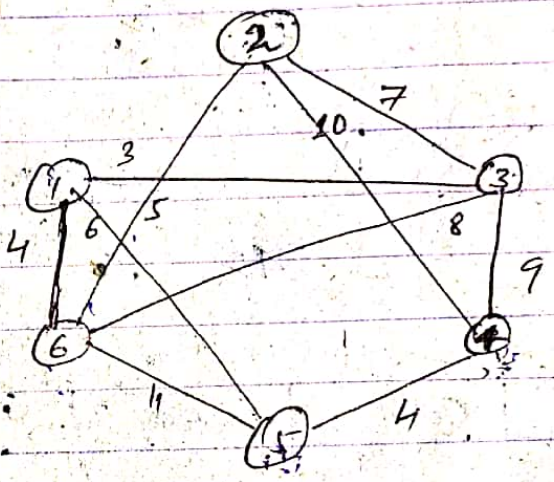
PRM's Algorithm

- choose a node & set its $d_v = 0$
- choose an unmarked node v with min d_v
- Mask v
- Add the edge (v, P_v)
- Update dist of all unmasked neighbours w of v

$$d_w = \min(d_w, C_{vw})$$

$$P_v = v$$

Repeat



	Mark	d_v	P_v
1	∅ 1	∅ 0	—
2	∅ 1	∅ ~~7~~ 5	+ ~~6~~
3	∅ 1	∅ 3	+ 1
4	● ●	∅ ~~9~~ 4	+ ~~5~~
5	∅ 1	∅ 6	+ 4
6	∅ 1	∅ 4	+ 1

$v=1$
$w=3$ $d_w=3$
$w=5$ $d_w=6$
$w=6$ $d_w=4$

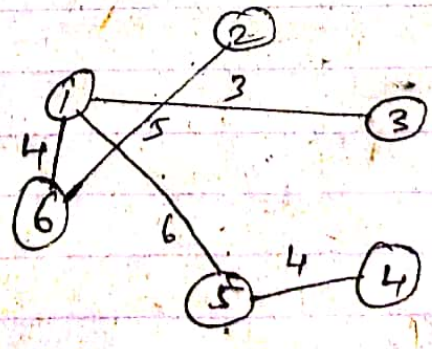
$v=3$
$w=2$ $d_2=7$
$w=4$ $d_4=9$
$w=6$ $d_6=4$

$v=6$
$w=2$ $d_w=5$
$w=5$ $d_w=6$

$v=2$
$w=4$ $d_4=9$

$v=5$
$w=4, d_4=4$

$v=4$

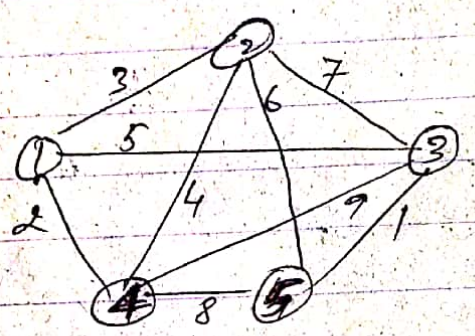


cost = 22

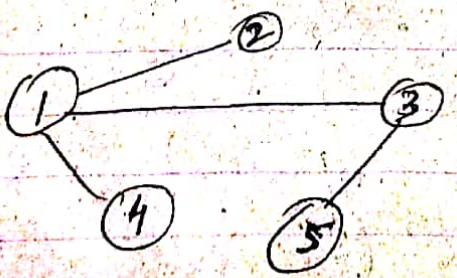
KRUSKAL's Algorithm:

- First sort the edges as per their respective costs.
- Initially assume that each vertex is a tree.
- Test if : u & v belong to different trees.
- then merge the trees.

Repeat

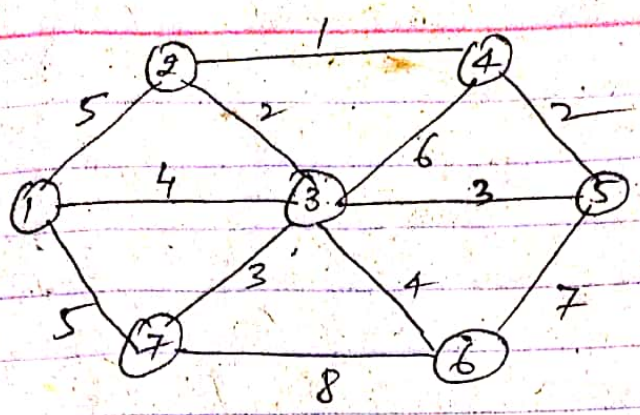


Edge	Cost	Add ?	(1), (2), (3), (4), (5) ← forest
(3,5)	1	Yes	(1), (2), (3,5), (4)
(1,4)	2	✓	(1,4), (2), (3,5)
(1,2)	3	✓	(1,2,4), (3,5)
(2,4)	4	X	(1,2,4), (3,5)
(1,3)	5	✓	(1,2,3,4,5) Halt
(2,5)	6		
(2,3)	7		
(4,5)	8		
(3,4)	9		



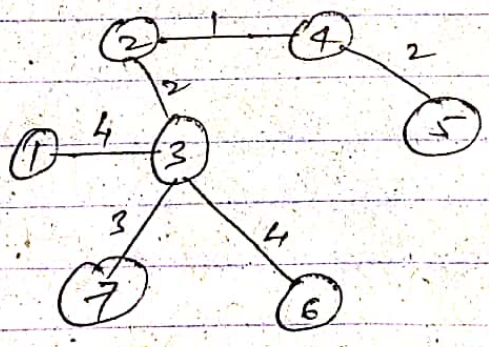
Syllabus for Mid-II (upto this from 29th July)

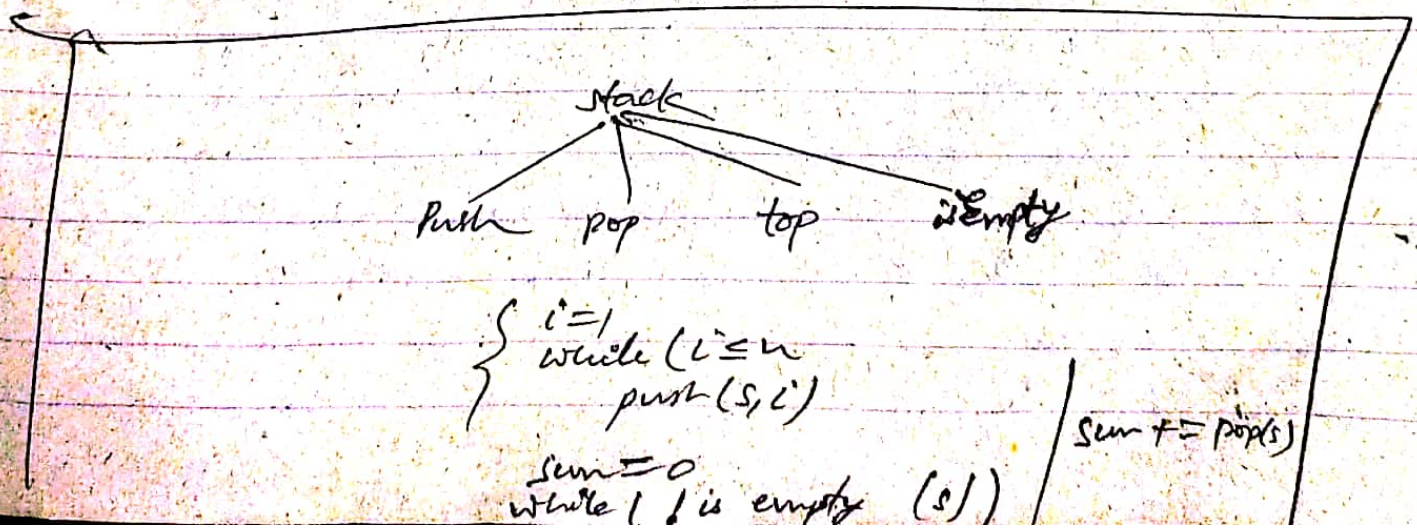
Q.



Edge	cost	Add	(1), (2), (3), (4), (5), (6), (7),
(2,4)	1	✓	(1), (2,4), (3), (5), (6), (7)
(4,5)	2	✓	(1), (2,4,5), (3), (6), (7),
(2,3)	2	—	(1) (2,3,4,5), (6), (7)
(3,7)	3	—	(1) (2,3,4,5,7), (6)
(3,5)	3	✗	(1), (2,3,4,5,7), (6)
(3,6)	4	—	(1), (2,3,4,5,6,7)
(3,1)	4	—	(1,2,3,4,5,6,7)

Ans





14/10/19

Lab

- Topological Sort
- Shortest path (BFS)
  - Naive
  - Using Queue
- Dijkstra's Algorithm
  - Naive
  - Using Min Heap
- Minimum Spanning Tree
  - PRIM's
  - KRUSKAL's

21/10/19

Sorting :

Problem Given an array of N numbers, sort them in ascending order

Solution

- Easy solution (comparison based)
- Tough solution (local comparison / divide & conq
- Non comparison based (under certain assump

Sorting :

Insertion sort

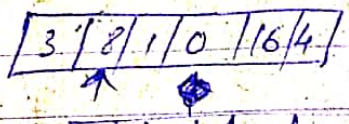
- N-1 passes over the array
- At the end of the p^{th} pass, the first (p+1) elements will be in a sorted order.

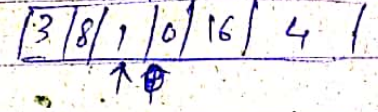
```
for p = 1 to N-1
    k ← A[p]
    j ← p-1
```

```
while (j>=0 && x<A[j])
{
    A[j+1] = A[j]
    j ← j-1
}
A[j+1] = x
```

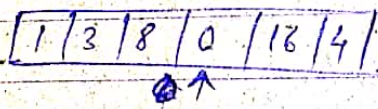
Example

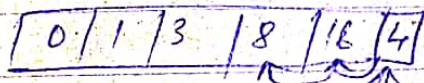
N=6

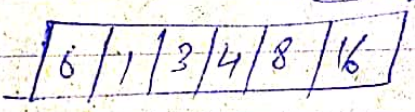


P=1 

P=1 



P=2 



21/09/19

Lab

- Breadth first Search
  - Naive
  - Using Queue
- Dijkstra's Algorithm
  - Naive
  - Using Queue
- PRIM's Algorithm
- KRUSKAL's Algorithm
- Insertion sort

22/10/19

Insertion Sort:

Analysis

Worst case: When the input is in descending order.

Pass	Number of comparisons
1st →	1
2nd →	2
⋮	⋮
$(N-1)^{th}$ →	$N-1$

$1+2+\cdots+N-1 = \frac{(N-1)N}{2} \Rightarrow O(N^2)$

Best case: Input is already sorted

Pass	No. of comparisons
1st →	1
2nd →	1
⋮	⋮
$(N-1)^{th}$ →	1

$1+1+\cdots+1 = N-1 \Rightarrow O(N)$

Average case:

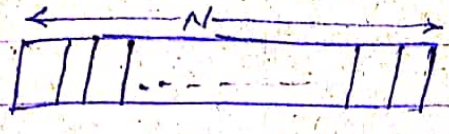
$$O(N) + O\left(\frac{N(N-1)}{4}\right) = O(N^2)$$

fact: Any sorting algorithm that compares only adjacent cells of an array has to do $\geq \frac{N(N-1)}{4}$ comparisons on an average.

Q. Is it possible to beat the $O(N^2)$ basics comparing only the adjacent element?

Merge sort

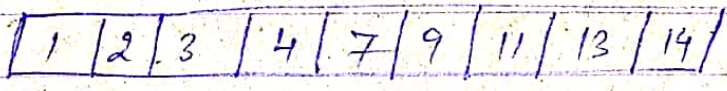
Idea: "Divide and conquer".



If N=1
    return

else
- divide the array into 2 almost equal parts.
- Sort left and right parts.
- Merge the sorted parts.

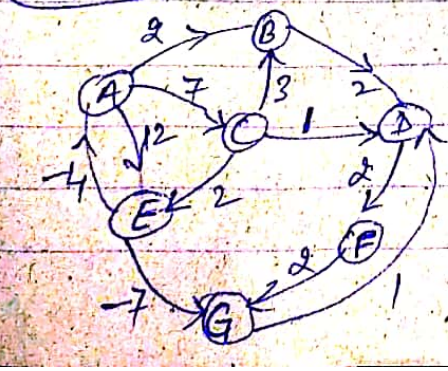
Merging two sorted arrays of $\frac{N}{2}$ size each can be done in O(N) steps (∵ each element is read only once)



If (N=1)
    return A[0]

else
    split A into A_1 and A_2
    B_1 = Merge sort (A_1)
    B_2 = Merge sort (A_2)
    return Merge (B_1, B_2)

Paper Question



	Dist	Path
A	0	—
B	2	A
C	7	A
D	4	B
E	12, 9	A, C
F	6	D
G	8	F

① A → B → D → F → G
   ↳ A → C → E → G

② A → B → D
   ↳ A → C → E → G → D

③ A → B → D → F
   ↳ A → C → E → G → D → F

Tutorial

```
if (N==1)
    return A[0]
else
    split A into A1 & A2
    B1 = Merge sort (A1)
    B2 = Merge sort (A2)
    return Merge (B1, B2)
```

Ex-

N = 8

3	10	8	4	2	0	1	9

23/10/19

Analysis of Merge Sort

$T(N)$ ⟶ for N-sized array

$T(1) = 1$

$T(N) = 2T(N/2) + N$ → for merging

$= [2T(N/2) + N/2] + N$

$= 2^2 T(N/2^2) + 2N$

$= 2^3 T(N/2^3) + 3N$

for any i

$T(N) = 2^i T(N/2^i) + iN$

Choosing $i = \log N$

$T(N) = 2^{\log N} T(1) + \log N \cdot N$

$\equiv O(N \log N)$

fact (revisited): Any comparison based sorting algorithm takes $\geq CN\log N$ steps.

Let us consider an array of N elements.

[| | . . . . . | |]

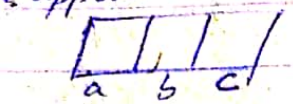
↓ after sorting

[| | . . . . . | |]

Putting the elements in ascending order is one permutations of the N elements.

Total number of permutations = N!

Suppose N=3

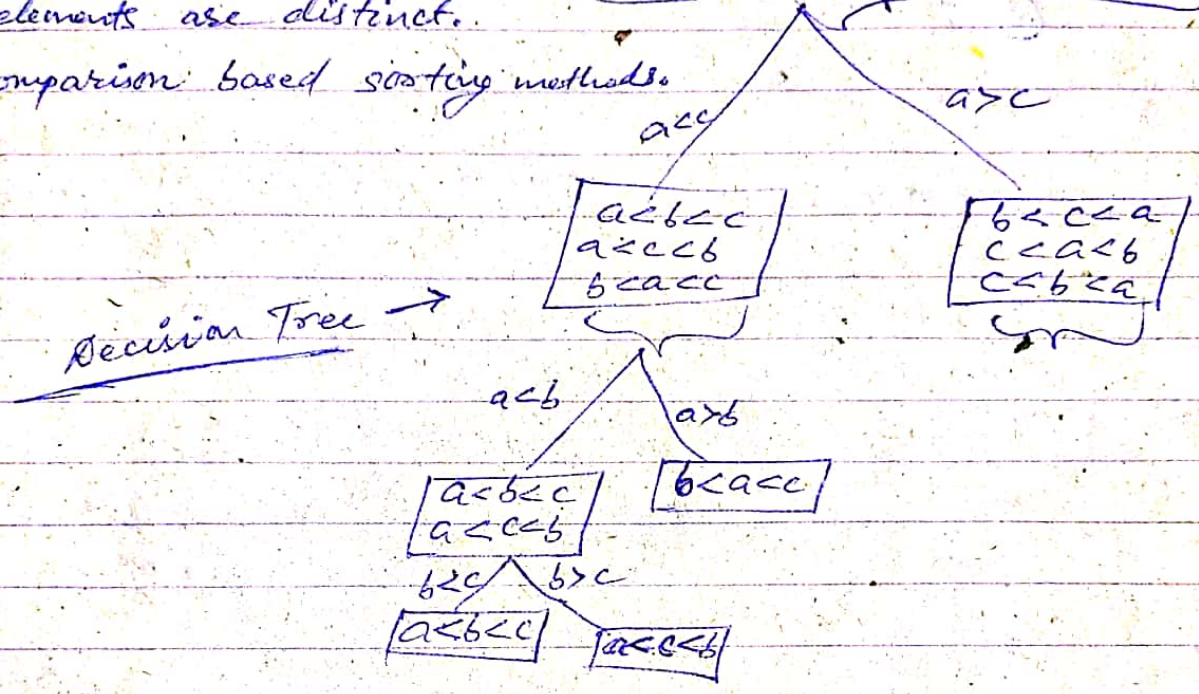


Total permutation

{ a<b<c b<a<c c<a<b
  a<c<b b<c<a c<b<a }

① All elements are distinct.

② for comparison based sorting methods.



Decision Tree →

for an array of size N, we have N! leaves.

No. of comparisons : Depth of tree (d)

Any binary tree of depth d has atmost 2^d leaves.

No. of leaves $\leq 2^d$

$N! \leq 2^d$

$d \geq \log(N!)$

⇒ No. of comparisons should be atleast $\log(N!)$.

$\log(N!) = \log(1.2.3.\cdots N)$

$= \log(1) + \log(2) + \cdots + \log N$

$\geq \log(N/2) + \log(\frac{N}{2}+1) + \cdots + \log(N)$

$\geq \log(N/2) + \log(N/2) + \cdots + \log(N/2)$

$= \frac{N}{2} \log(N/2) = \Omega(N \log N)$

[redacted]

1. $f(n) = O(g(n))$ if $\exists\ c, n_0$ such that $\forall\ n \geq n_0$, $f(n) \leq c g(n)$.
2. $f(n) = \Omega(g(n))$ if $\exists\ c, n_0$ such that $\forall\ n \geq n_0$, $f(n) \geq c g(n)$.
3. $f(n) = o(g(n))$ if $\exists\ c, n_0$ such that $f(n) < c g(n)$.
4. $f(n) = \omega(g(n))$ if $\exists\ c, n_0$ such that $f(n) > c g(n)$
5. $f(n) = \theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$
   ↳ $\exists\ c_1, c_2, n_0$ such that $\forall\ n \geq n_0$,
   $c_1 g(n) \leq f(n) \leq c_2 g(n)$

Ensures tight bound

24/10/19

Depth First Search (DFS)

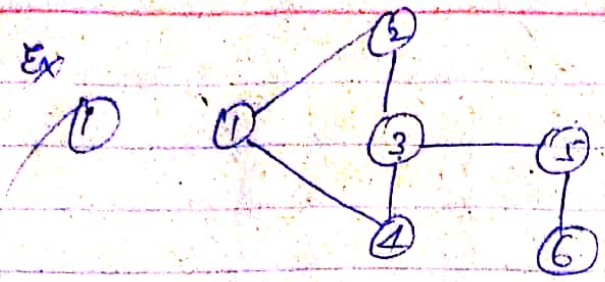
For Tree (acyclic undirected graph)

DFS ≡ Pre-order traversal

for general graph

```
DFS(u)
{
  Mark u (≡ visited / processed)
  for all unmarked neighbours 'w' of u
  DFS(w)
}
```

Ex



u=1

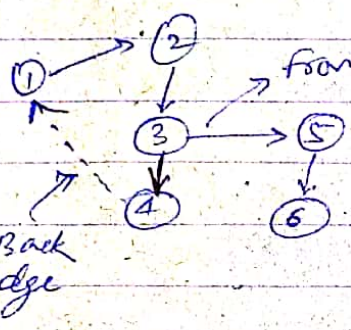
DFS(1) ⇒ 1—2—3—5—6—4

4—5—6

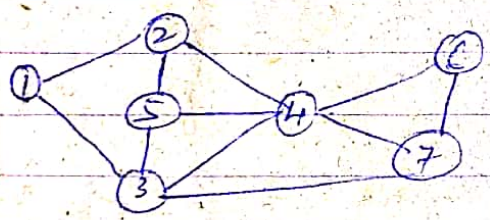
4—3 < 2—5—6 / 5—6—2

(2)

1 → 2 → 3 → 4 → 5 → 6

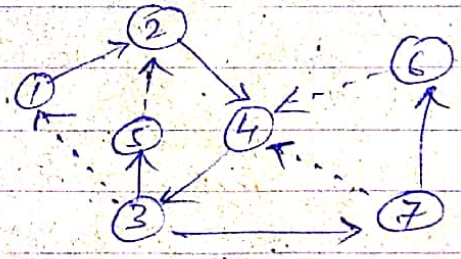


(3)



u=1

DFS(1) → 1 → 2 → 4 → 3



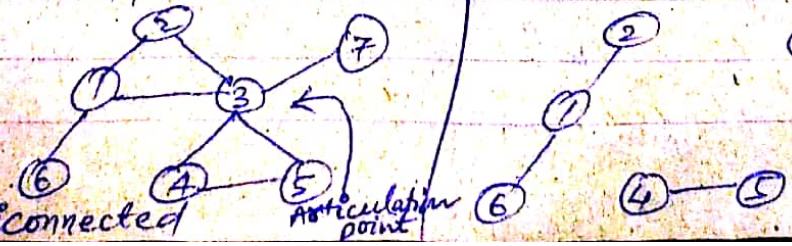
→ 1^b → 5 → 1^b → 2^b → 7 → 4^b → 6 → 4^b.

DFS = 1 → 2 → 4 → 3 → 5 → 7 → 6

Problem: "Biconnectivity"

Deletion of any one node does not make the graph disconnected.

Ex -



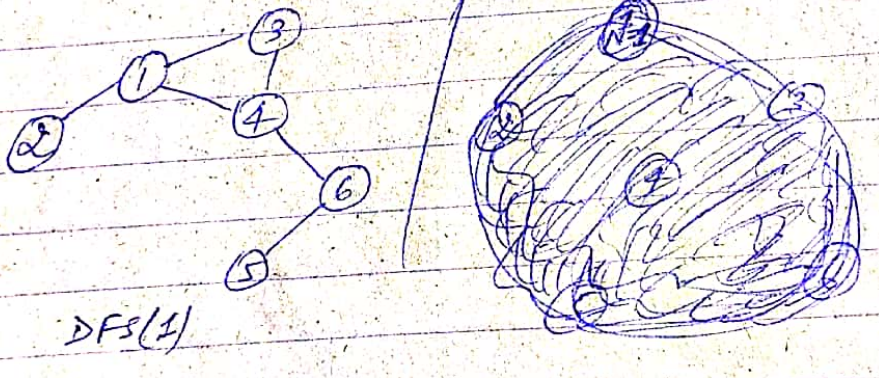
Algorithm:

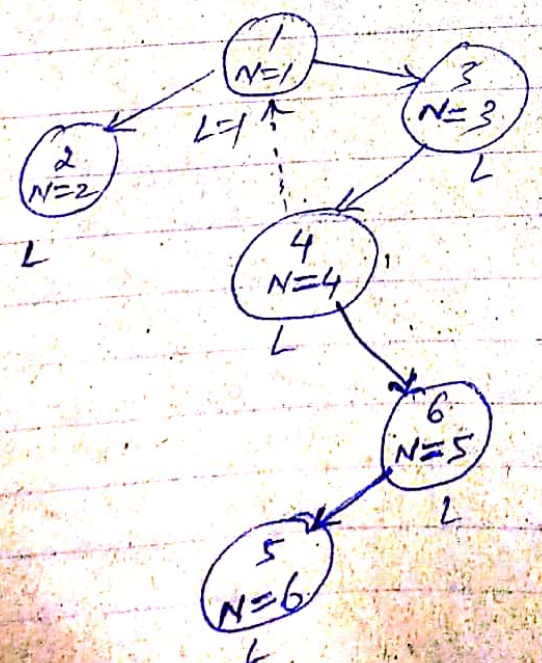
- for every vertex v_i we assign a number $N(u)$ → numbering in the DFS masking order.
- $L(u)$ → the least numbered node (in terms of $N(u)$ that is reachable from u using any number of front edges followed by at most one back edge

A node ($\neq$ root) is an articulation point iff $\exists$ neighbour w of u such that

$$L(w) \geq N(u)$$

{* root is an articulation point if it has >1 child





25/10/19

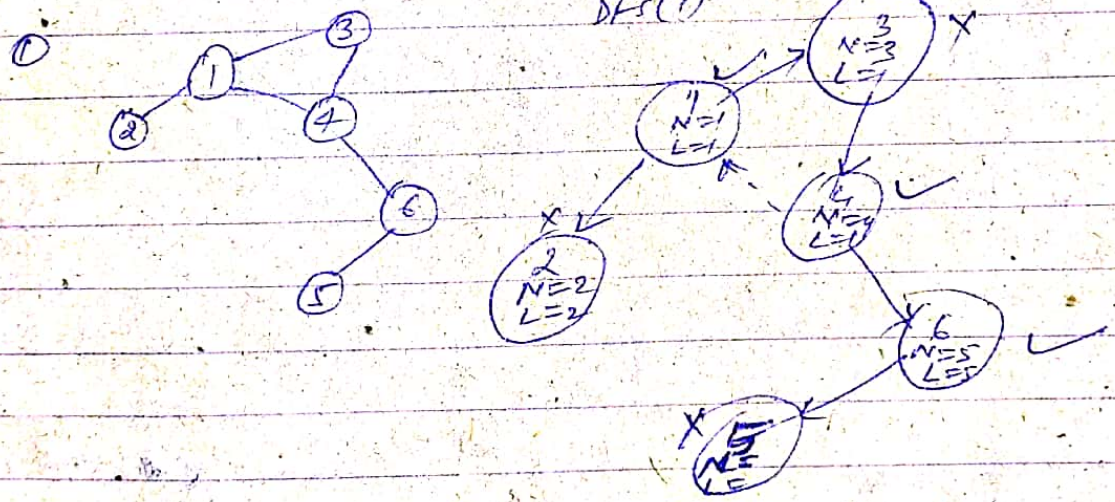
Q. Given an undirected graph G, output all its articulation points.

Algorithm:

- For every vertex, we give a number N(u). (based on DFS marking)
- L(u) → the least number node (in terms N(u)) that is reachable from u using any no. of front edges followed by at most one back edge.
- A node u (≠ root) is an articulation point iff ∃ a neighbour w of u such that L(w) ≥ N(u)
- Root is an articulation point if it has > 1 child nodes.

Ex-

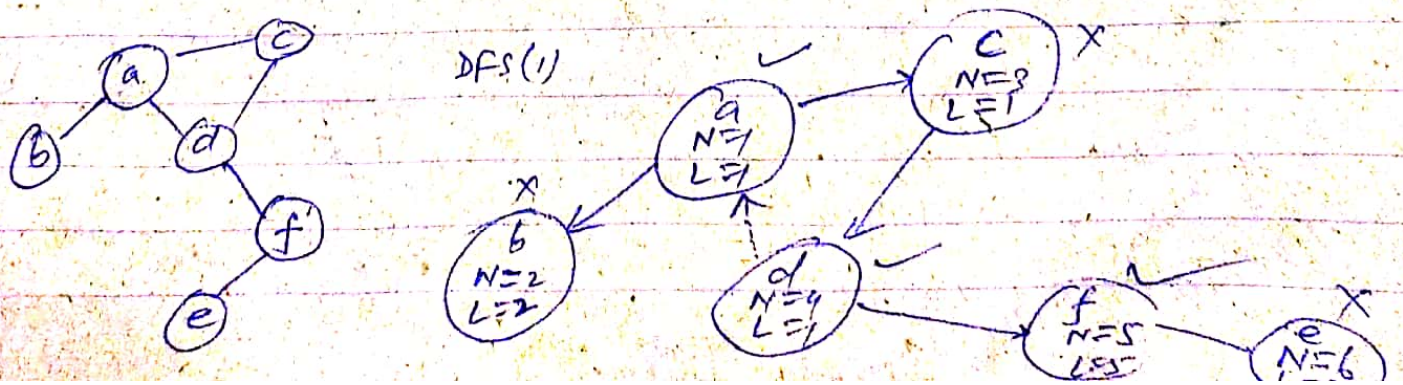
①



DFS(1) → 1, 2, 3, 4, 6, 5

Articulation point → 1, 4, 6

②



Bucket sort:

We know beforehand the upper bound of the values we are going to sort.

Step 1: Create an array B of size M (initialized to all 0s) where no element in array A is >M

Step 2: Scan A (from left to right) on reading a value v, increment B[v] by 1.

Step 3: Scan B from left to right. Print value i B[i] times.

$A = [3, 3, 7, 4, 5, 6, 0, 0, 8, 9, 9, 8, 4, 6, 5]$

N

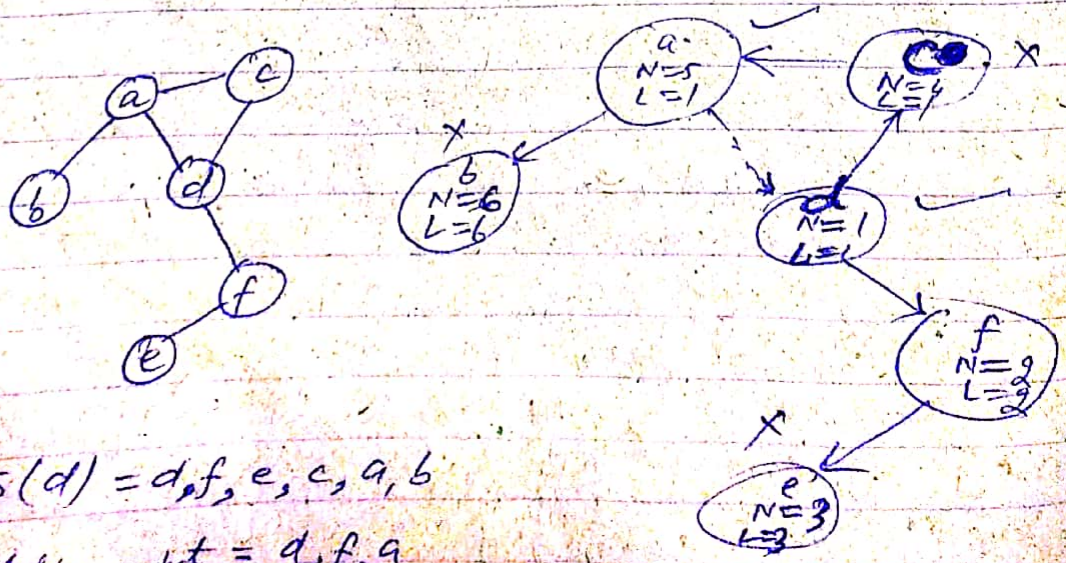
	0	1	2	3	4	5	6	7	8	9
B =	2	0	0	2	2	2	2	1	2	2
	0	0	0	0	0	0	0	0	0	0

M

O/p → 0, 0, 3, 3, 4, 4, 5, 5, 6, 6, 7, 8, 8, 9, 9

$O(M+N)$

Ex –

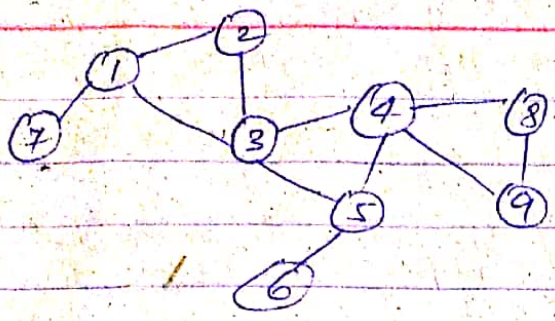


DFS(d) = d, f, e, c, a, b

Articulation point = d, f, a

Tutorial

Q.



DFS (2) = 2, 1, 3, 4, 5, 6, 8, 9, 7

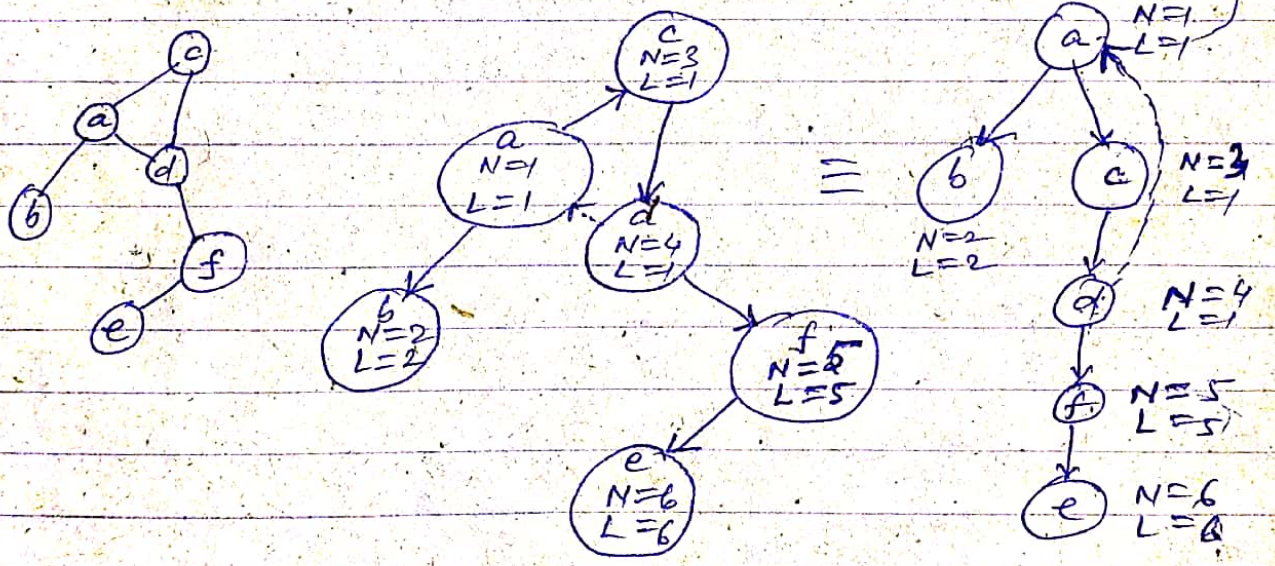
{ -L values
  -Articulation point

04/11/19

Articulation Points:

(Using DFS)

DFS(1)



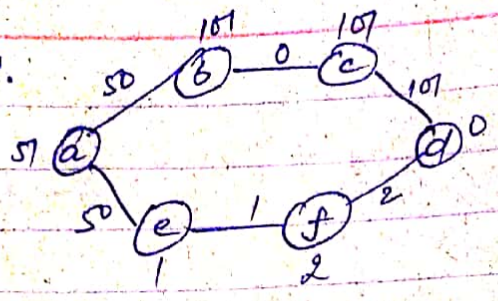
Lab Exam → Sat → 30 Nov → 3-6

L(w) ≥ N(w)

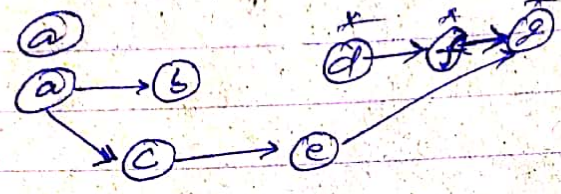
- Look only at the outgoing neighbours.
- N & L values will always increase as we move down with L ≤ N

Paper sol^{n} - Mid Sem - 2

Q.1.



Q. 2

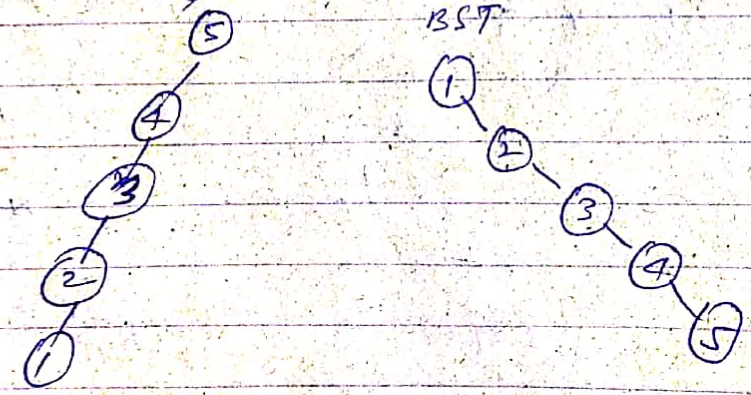


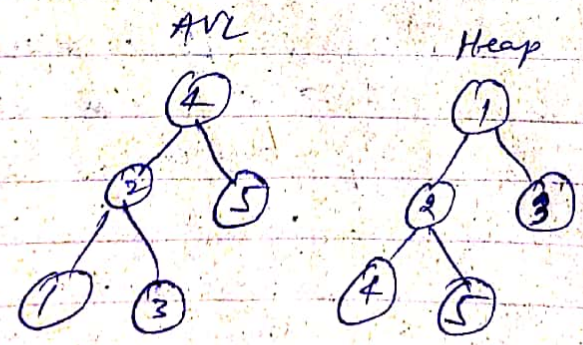
(b)

(c) EG

(3) NO, Y/N; Y, Yes

(4) Splay tree → {1, 2, 3, 4, 5}





04/11/2019

Lab:

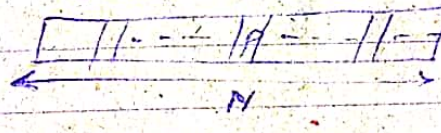
- BFS
  - Naive
  - Using Queue
- Dijkstra's Algorithm
  - Naive
  - Using Queue
- Sorting
  - Insertion sort
  - Merge sort
  - Bucket sort
- finding articulation points using DFS.

06/11/19

Sorting

- BST based
- Heap sort
- Insertion sort
- Merge sort
- Bucket sort

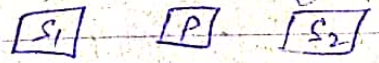
Quick sort



N

Step 1: Choose a pivot p

Step 2: Partition the array into 3 parts.



all elements < p all elements > p

Step 3: Output [Quicksort (S_1) P Quicksort (S_2)]

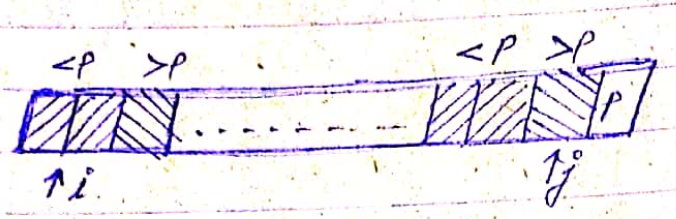
How to choose p ?

Best choice (?) — p is median

finding the Median is non-trivial

$p = \text{Median}(A[0], A[\lfloor N/2 \rfloor], A[N-1])$

How to partition ?



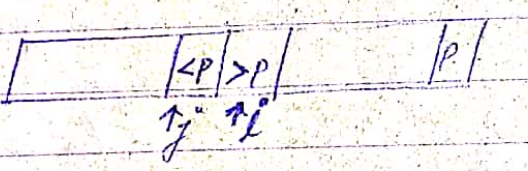
Step 1: swap p with A[N-1]

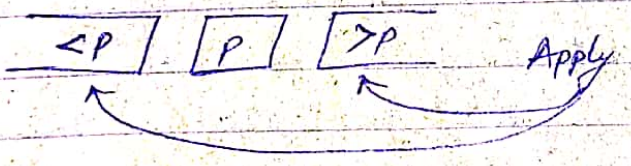
Step 2: Move i to the right as long as an element ≥p is not found.

Step 3: Move j to the left as long as an element <p is not found.

Step 4: If i and j have not crossed over (i<j), swap A[i] & A[j]. Then go to step 2

Step 5: If crossed over, swap p and A[i]
↳ A[N-1]





Ex -

0	1	2	3	4	5	6	7	8	9	10
87	42	22	81	17	3	19	28	33	56	91

p = Median (87, 3, 91) = 87

S1 A[0] = 91 A[10] = 87

56	42	22	81	17	3	19	28	33	91	87

↑i ↑j

swap (91 & 56)

Increament i unless cross over

56	.....	28	33	87	91

<87 >87

→ Applying Quicksort individually

56	42	22	81	17	3	19	28	33	...

↑i ↑j

P = Median (56, 17, 33) = 33

swap (56, 28) | swap (81, 3)
swap (42, 19) | swap ( )

28	19	22	3	17	33	42	56	81

swap (33, 81)

(a) | 28 | 42 | 22 | 81 | 17 | 3 | 19 | 56 | 33 |

(b) | 28 | 19 | 22 | 81 | 17 | 3 | 42 | 56 | 33 |

(c) | 28 | 19 | 22 | 3 | 17 | 81 | 42 | 56 | 33 |

(d) | 28 | 19 | 22 | 3 | 17 | 33 | 42 | 56 | 81 |

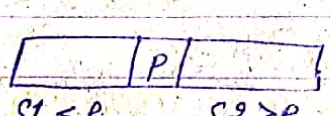
<P >P

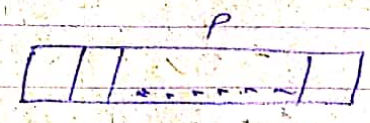
07 | 11 | 19

Quick sort:

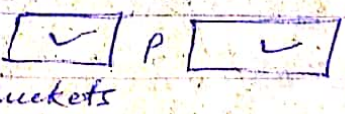
Analysis

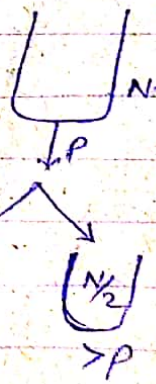
Worst case:





- Already sorted in descending order.
- 'P' is the first/last element.

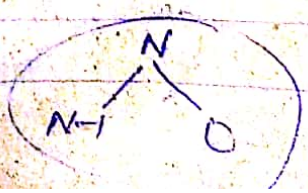




→ 'P' is always the smallest

$T(N) = T(N-1) + O(N)$

$= T(N-2) + O(N-1) + O(N)$



$T(N) = T(N-i) + O(N-i+1) + O(N-i+2) + \cdots + O(N)$

$T(1) = 1$

substitute $i = N-1$

$T(N) = T(1) + O(2) + O(3) + \cdots + O(N)$

$= 1$

$\Rightarrow T(N) = O(N^2)$

Best case: P is the median of all the elements

$T(N) = 2T\left(\frac{N-1}{2}\right) + O(N)$

$= O(N \log N)$

Tutorial Quest.

Ex-

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	10	8	4	2	0	1	9	6	5	11	100	-8	12	-10	107

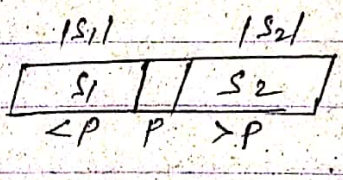
Show the steps

- Mention the swaps
- No need to make the full array at every step

08/11/19

Quick Sort:

Average case



N elements

$T(N) = T(|S_1|) + T(|S_2|) + O(N)$

$= T(|S_1|) + T(|S_2|) + c'N$

$= T(|S_1|) + T(N - |S_1| - 1) + c'N$

Average case

N = 5

$\|S_1\|$	$N - \|S_1\| - 1$
0	4
1	3
2	2
3	1

avg ($\|S_1\|$)

$T(N) = 2(\text{avg } T(|s_i|) + c'N$

$\text{avg } T(|s_i|) = \sum_{i=0}^{N-1} T(i)$

$T(N) = \frac{2}{N}\left[\sum_{i=0}^{N-1} T(i)\right] + c'N$

$NT(N) = 2\sum_{i=0}^{N-1} T(i) + c'N^2$

$(N-1)T(N-1) = 2\sum_{i=0}^{N-2} T(i) + c'(N-1)^2$

Subtract

$NT(N) - (N-1)T(N-1) = 2T(N-1) + cN$

$NT(N) = (N+1)T(N-1) + cN$

Divide by $N(N+1)$

$\frac{T(N)}{N+1} = \frac{T(N-1)}{N} + \frac{c}{N+1}$

$\frac{T(N-1)}{N} = \frac{T(N-2)}{N-1} + \frac{c}{N}$

$\frac{T(2)}{3} = \frac{T(1)}{2} + \frac{c}{2}$

$\frac{T(N)}{N+1} = \frac{T(1)}{2} + \frac{c}{N+1} + \frac{c}{N} + \cdots + \frac{c}{3}$

$\frac{T(N)}{N+1} = c\underbrace{\left(\sum_{i=3}^{N+1} \frac{1}{i}\right)}_{\text{Harmonic Series}} = O(\log N)$

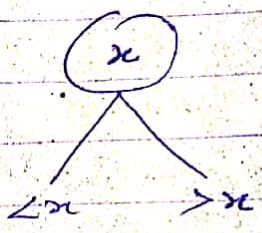
$\Rightarrow T(N) = O(N \log N)$

$\Rightarrow$ Avg. case matches the best case.

11/10/19

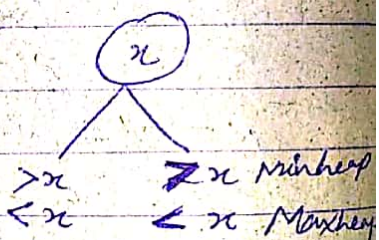
BST :

- Each node has a 'key' value.
- follows BST property.



Binary Heap :

- Each node contains 'priority' value
- follows ordering & structure property



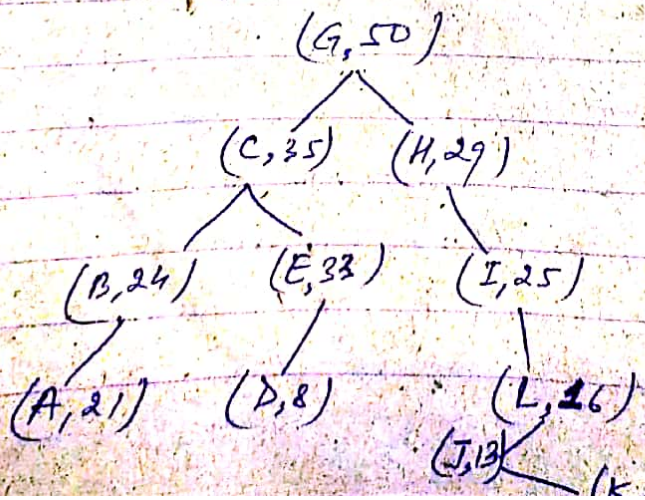
A Treap is a binary tree :

- Each node contains both 'key' and 'priority'.
- BST ordering property w.r.t. keys and heap ordering property w.r.t priorities.
- NO structure property as in heap.

Ex-

Keys - alphabets

priorities - numbers.



Uniqueness of Treaps:

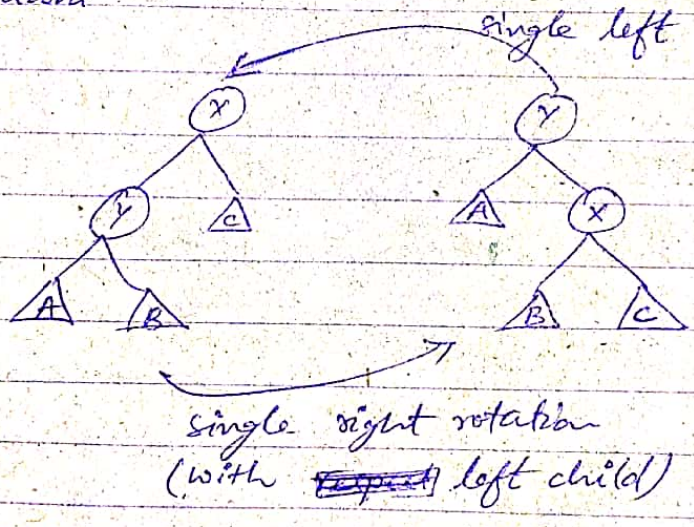
* With all the key values unique.
  - Start with an empty treap.
  - Insert the (key, priority) pairs in decreasing order of priority, using the usual BST insert operation that considers only the key values.

* If the priority as well as the key values are unique, the treap is unique.

(G, 50), (C, 35), (E, 33), (H, 29), (I, 25), (B, 24), (A, 21), (L, 16), (J, 13), (K, 9), (D, 8)

Operations on Treaps

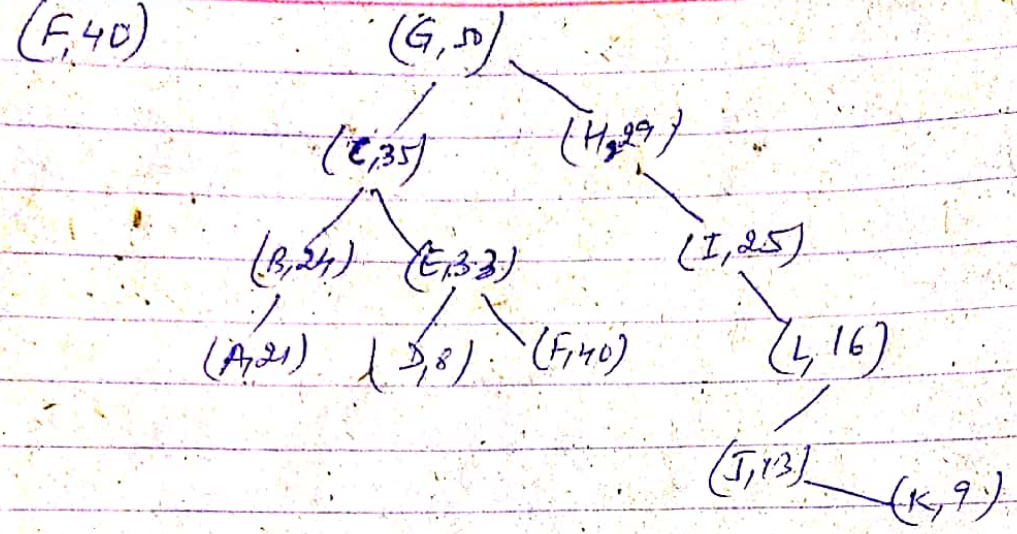
- Find: Use the usual BST find algorithm for keys.
- Insert & delete: Respect both BST & Heap ordering properties
- Use AVL relations

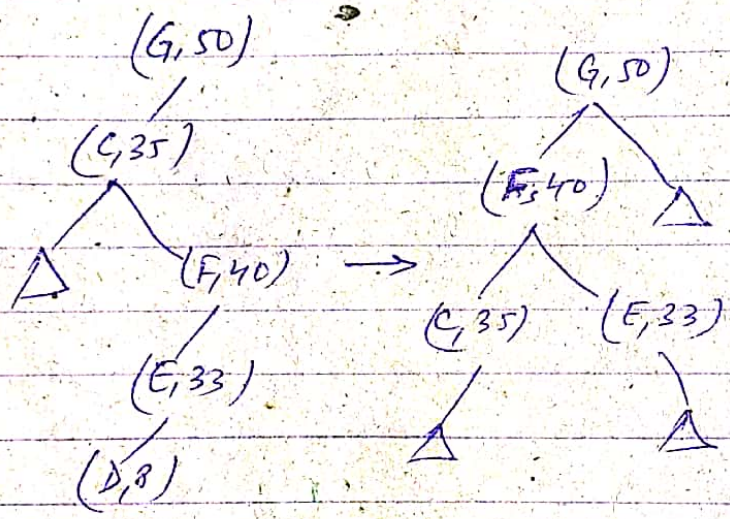


- An AVL relation always preserves BST ordering; can be used to move nodes up/down to maintain heap ordering.

Insert (K, P):

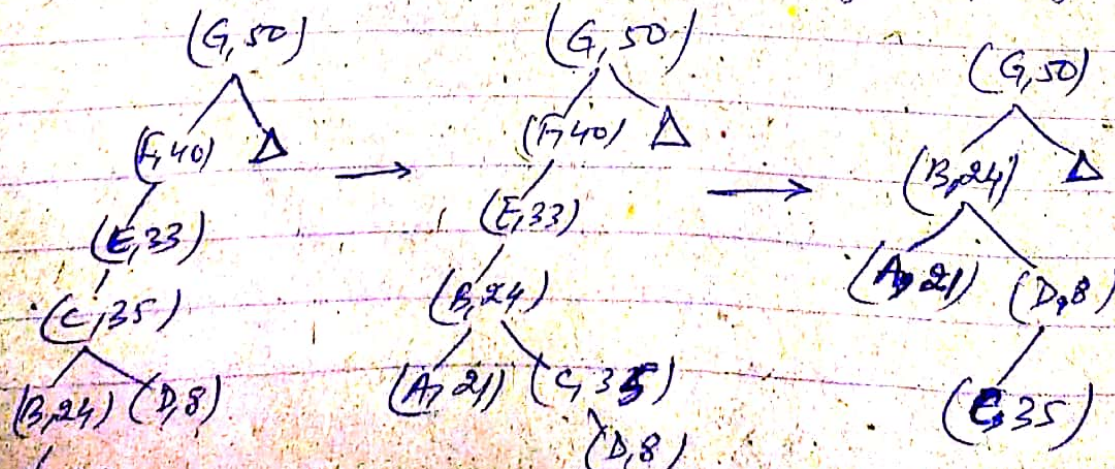
- Insert as a new leaf node using the usual BST insert method w.r.t K.
- Rotate the nodes up using AVL rotations as necessary, until the priority of its parent is $\geq P$





Delete (K):

- Search the node x with key k using BST finding algo
- If x is a leaf node, delete it.
- Otherwise, use AVL rotations to rotate the node down until, it becomes a leaf, then delete it. (rotate the node with its larger priority child)



11/10/19

Lab:

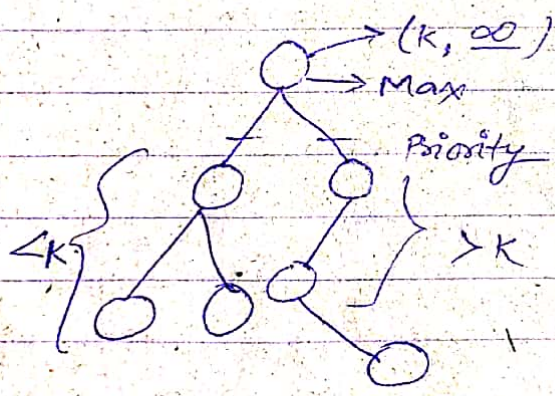
- Merge sort
- Quick sort
- Compare the execution time of your implementation with the inbuilt functions in c++ by varying the no. of elements in $\{10, 100, 10^3, \dots\dots 10^6, 10^7\}$
- Treap.

18/11/19

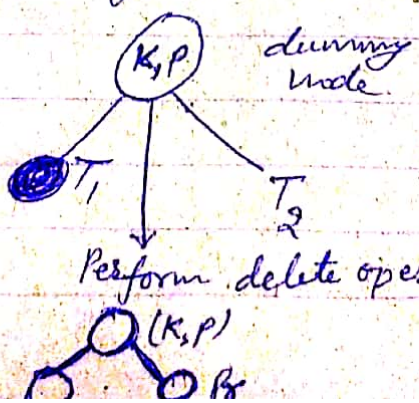
Treap:

Problem: "Tree Splitting" - Given a tree and a key value K not in the tree, create two trees: one with keys less than K, and one with keys greater than K.

(Key, Priority)



Problem: "Tree joining" - Given two trees T_1 & T_2 such that each key in T_1 is less than all the keys in T_2, create a new tree T that contains all & only the keys from T_1 & T_2.

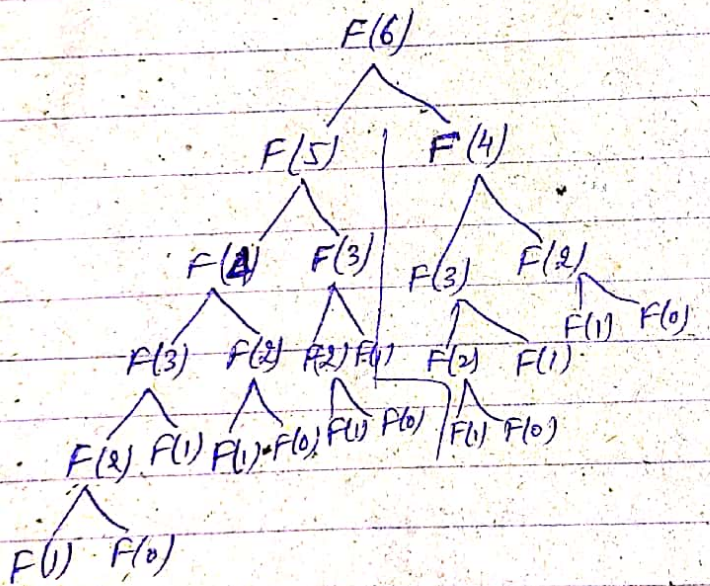


1, 1, 2, 3, 5, 8, 13, 21 - - - - - -

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

$F(0) = F(1) = 1$

```
int fib (int n)
{
  if (n>=1)
    return 1,
  else
    return (fib(n-1) + fib(n-2))
}
```




```
int fib (int n)
{
  int i, next_to_last, last, answer,
  if (n<=1) return 1;
    last = next_to_last = 1,
  for (i=2, i<=n ; i++)
  {
    answer = last + next_to_last;
    next_to_last = last;
    last = answer;
  }
  return answer;
```

fib(n)

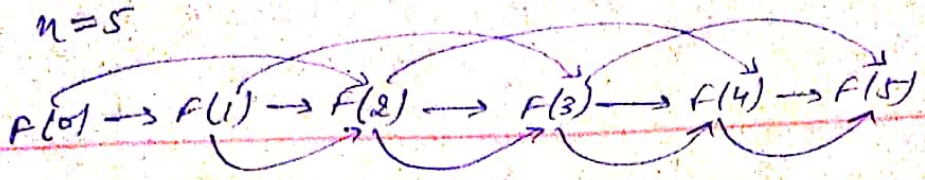
n=6

F(0), F(1)

F(2) = F(0) + F(1)

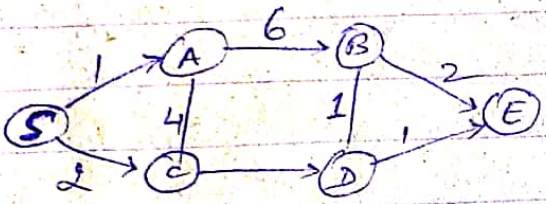
F(3) = F(2) + F(1)

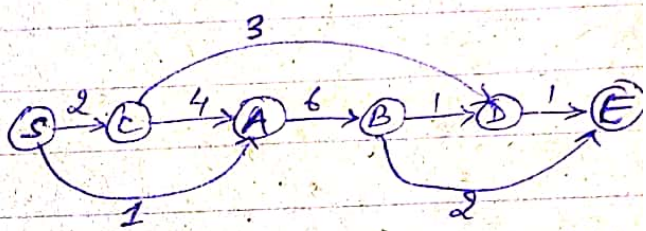
$n = 5$



- DAG
- Subproblems arranged in increasing order of difficulty.
- Directed dependences (cum-directional)

Shortest path in DAG:





Source = S

$dist(D) = \min(dist(C)+3, dist(B)$

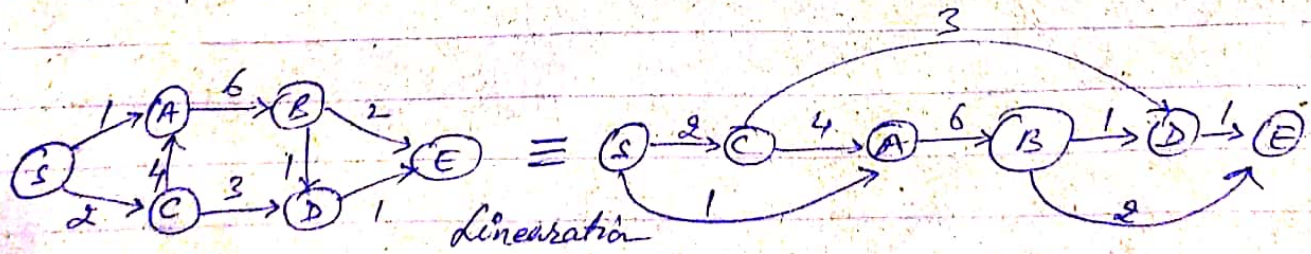
$dist(S) = 0$

- initialize all dist() value to ∞
- $dist(S) = 0$
- for each $v \in V \setminus \{S\}$, in linearized order

$dist(v) =$ _____

14/11/19

Shortest paths in DAGs



Linearization (for topological sorting)

Source = S

$d(S) = 0$

$d(C) = \min\{0+2, \infty\} = 2$

$(u, C) \in E$

u

S

$d(A) = \min\{\infty+1, 2+4\} = 1$

$(u,A) \in E \quad u=5 \quad u=6$

$\{5, 6\}$

$d(B) = 1+6 = 7$

$d(D) = \{5, 8\} = 5$

$d(E) = \{6, 9\} = 6$

- Initialize all dist.( ) values to ∞.
- dist $(S) = 0$
- for each $v \in V \setminus \{S\}$ in linearized order
  dist $(v) =$

- Fibonacci series
- Shortest path

Algorithmic Approach

- Dynamic Prog — ① Prob → sub-problems, multiple
  ② Linear/sequential dependency
  ↳ simple to more difficult sub-problems
  ③ Some solve to subproblems.

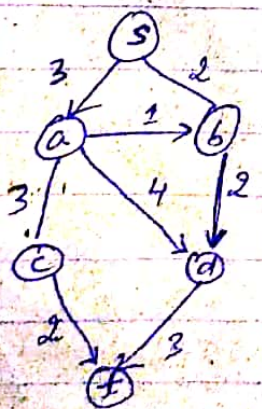
$$C(n) = \begin{cases} \sum_{i=0}^{n-1} C(i) + n & ; C(0) = 1 \end{cases}$$

$C(0) \rightarrow C(1) \rightarrow C(2) \rightarrow C(3)$

Prog. for Linearized version

15/15/19

Network Flow:



S → source

t → Sink

edge weights ≡ flow capacity

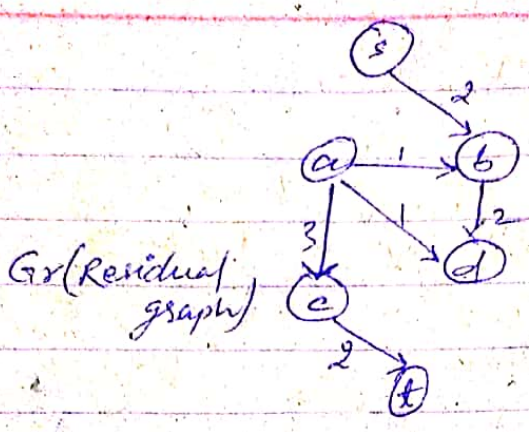
Ex - water, Bandwidth in MBPS

No. of lanes

Algo-1

① s, a, d, t

Max flow = 3



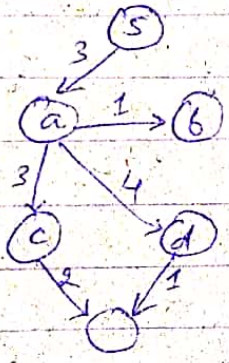
Gr (Residual graph)

Total Max flow = 3

Algo-1

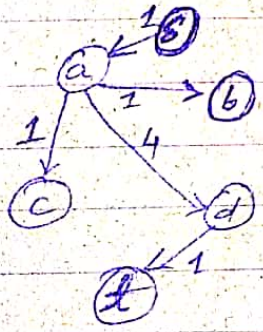
① s, b, d, t

Max flow = 2



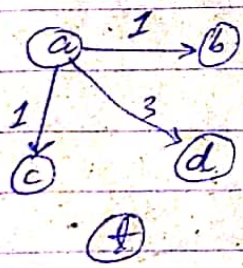
② s, a, c, t

Flow = 2



③ s, a, d, t

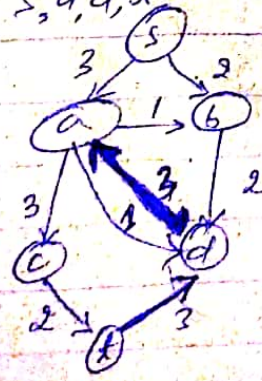
Flow = 1



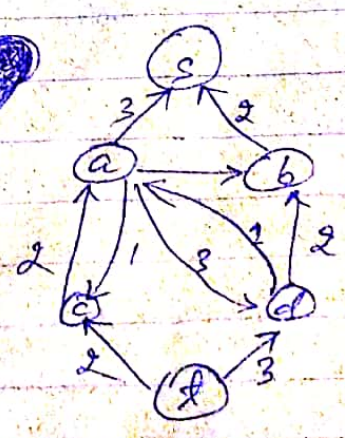
Max flow = 2 + 2 + 1 = 5

Greedy Approach

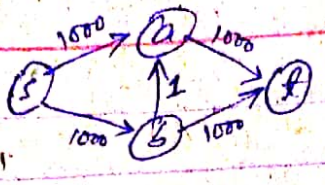
① s, a, d, t flow = 3



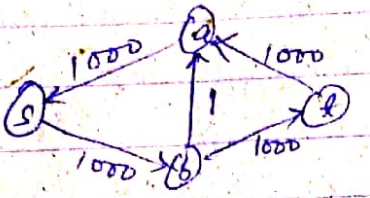
②



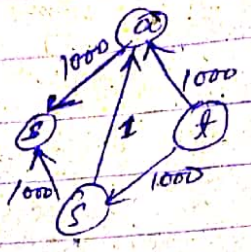
Total Max flow = 3 + 2 = 5



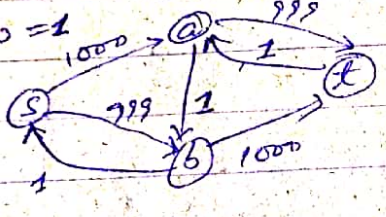
① s, a, t , flow = 1000



② s, b, t
flow = 1000



① s, b, a, t
flow = 1



② s, a, b, t
flow = 1
⋮
2000 steps

If capacities are all +ve integers and the max flow is 'f', since each path increases the flow value by atleast 1, 'f' steps suffice, & the total running time is $O(f|E|)$

Mincost - Max flow problem:
Each edge has not only a capacity but also a cost per unit of flow. The problem is to find among all max flow solutions, the one with the min cost.

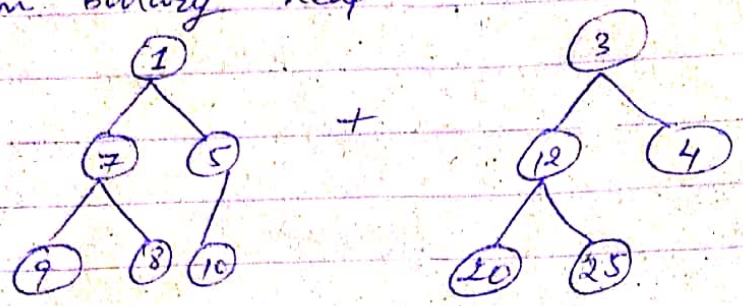
18/11/19

Leftist Heap:

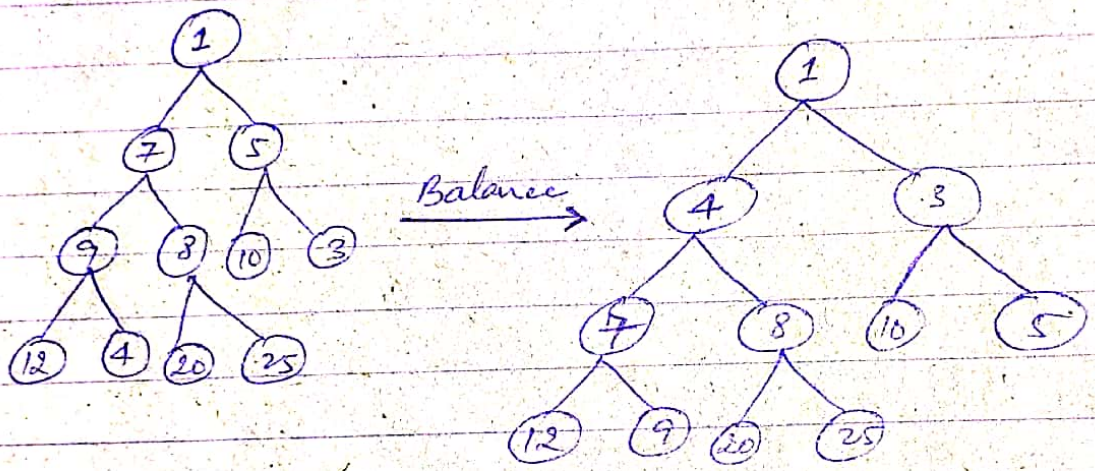
- Binary heaps with merge operation
- Properties
  - similar to Binary Heap except the bias

go deep towards the left.

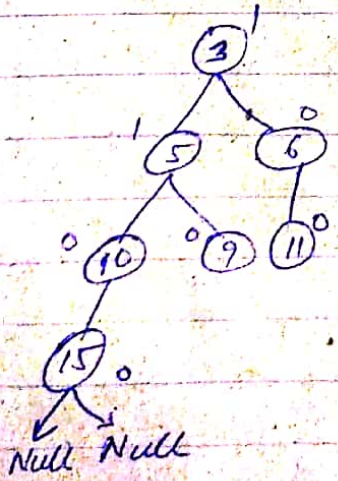
Ex - from binary heap



↓ Merge (Build new heap)



Null Path Length: The shortest dist we need to take from a given node to reach NULL pointer.



NPL of NULL pointer is −1.

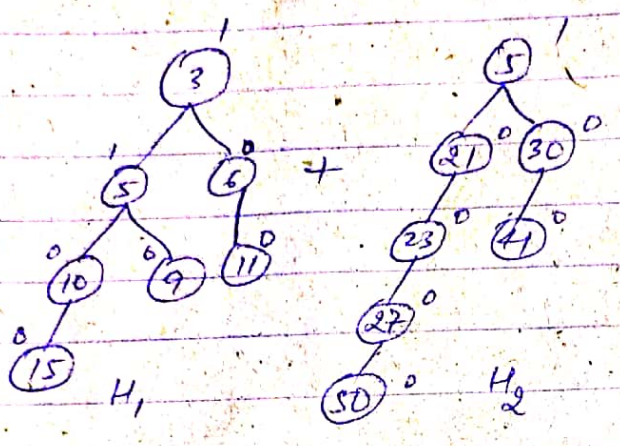
* { In a leftist heap, NPL of left child ≥ NPL of right child for all nodes.

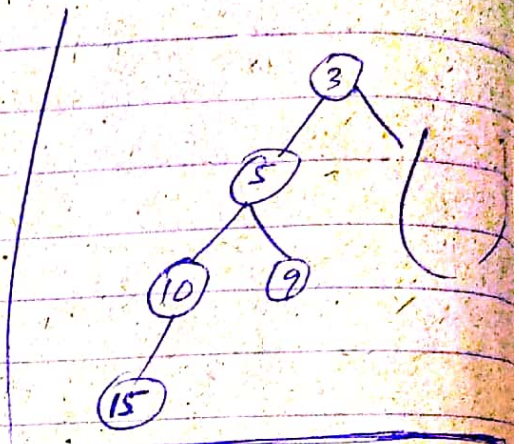
fact: In any leftist heap with r nodes in its rightmost path, there will be atleast $2^r - 1$ nodes.

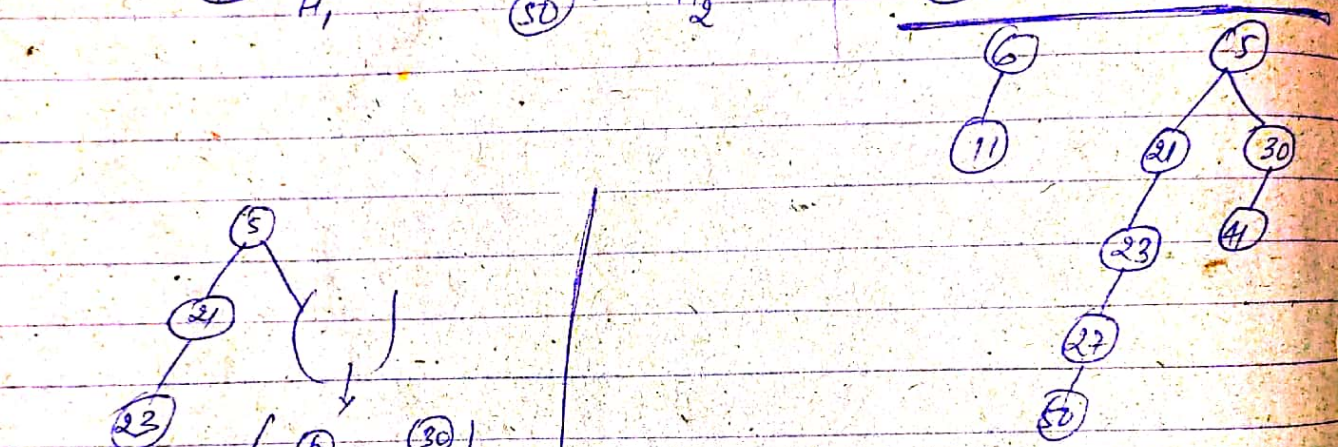
(Derivation ← Homework)

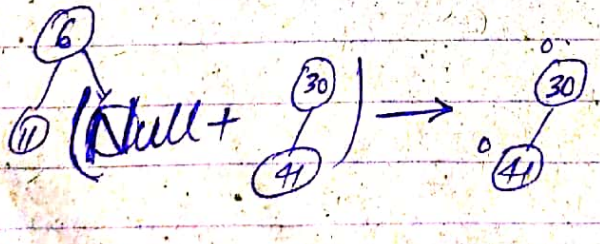
Merging two LHs:

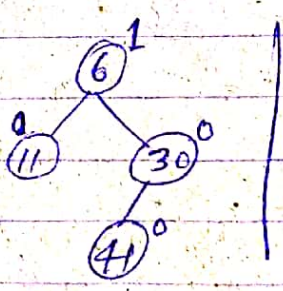
Recursive method – See the value stored in root, take the one with smaller value. Recursively merge right subtree with another one.

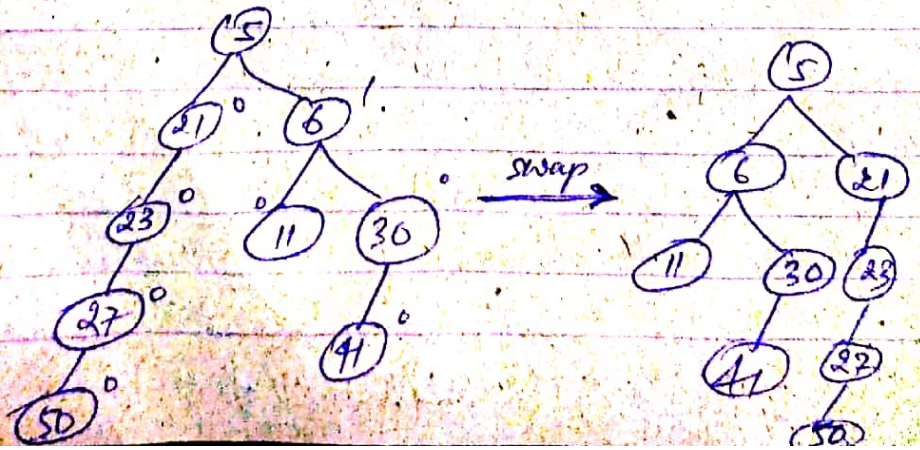












18/11/19

Lab:

- Treap
- DP examples
  - Fibonacci series
  - shortest path
- Network flow
  - The two methods discussed in class.

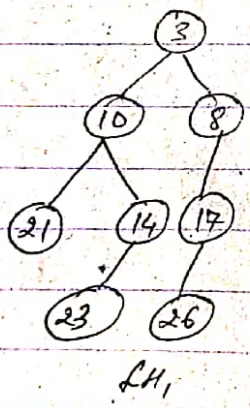
20/11/19

Leftist heap

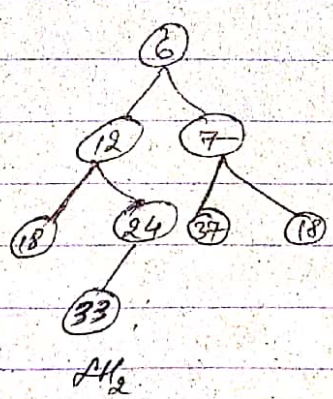
Merge → Non-recursive soln

Take all nodes in the rightmost paths of LH_1 and LH_2. Merge them in ascending order. Keep adding their left subheaps. Swap if not leftist at any node.

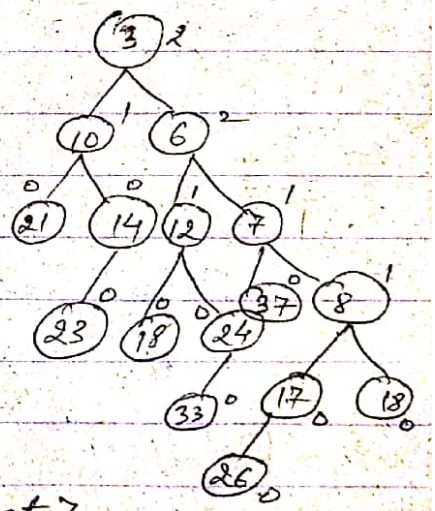
Ex -



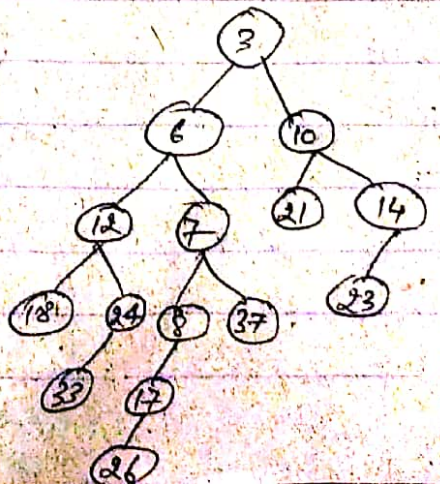
LH_1



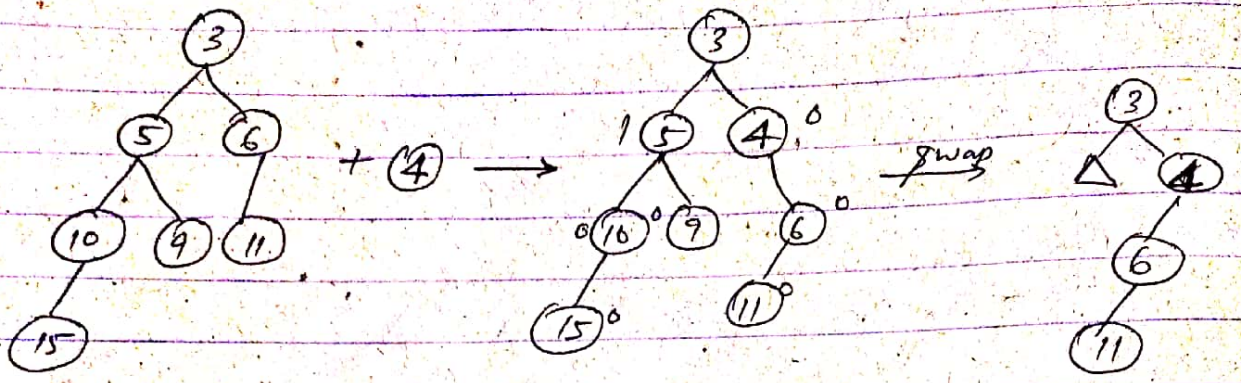
LH_2



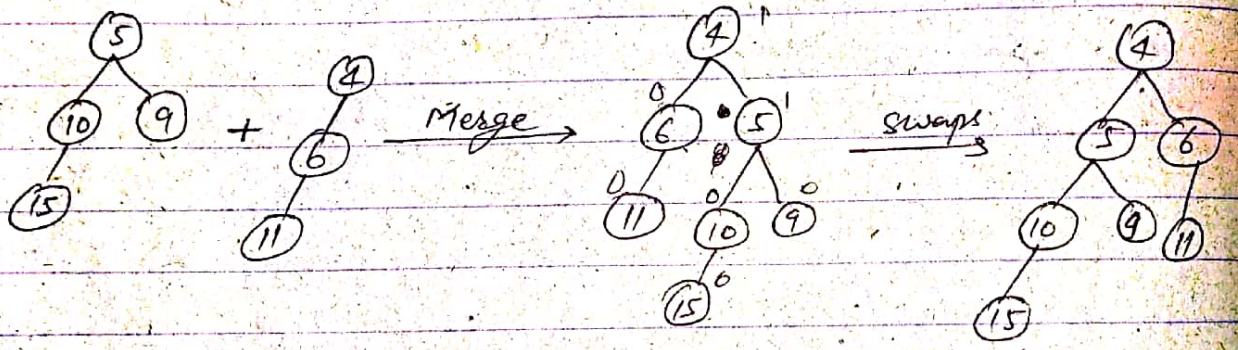
NPL
- Swap at 7
- Swap at 3



Insert :



Delete Min :



Fact: Right paths contains at most $\lfloor \log(n+1) \rfloor$ nodes.

→ $O(\log n)$

Binary Heap :

- Insert & delete.
- Merge

Skew Heap :

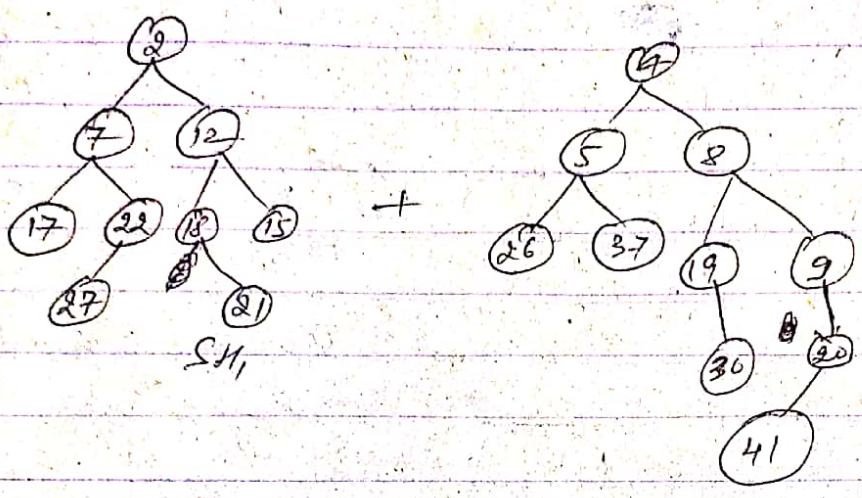
- No structure property
- Order property is same.
- No NPL property

Merging two SHs

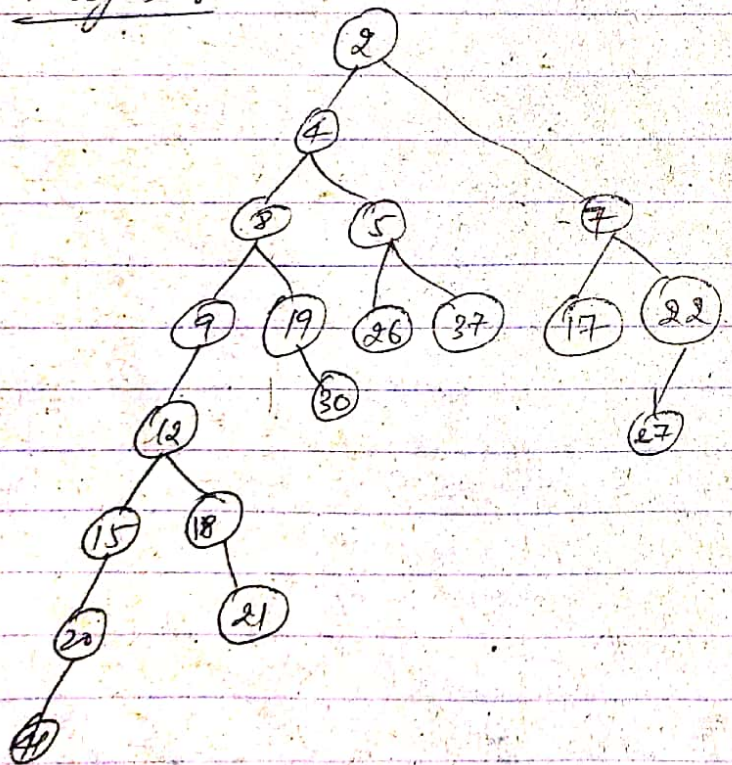
Take elements in right most path of first & second merge them in ascending order and that chain

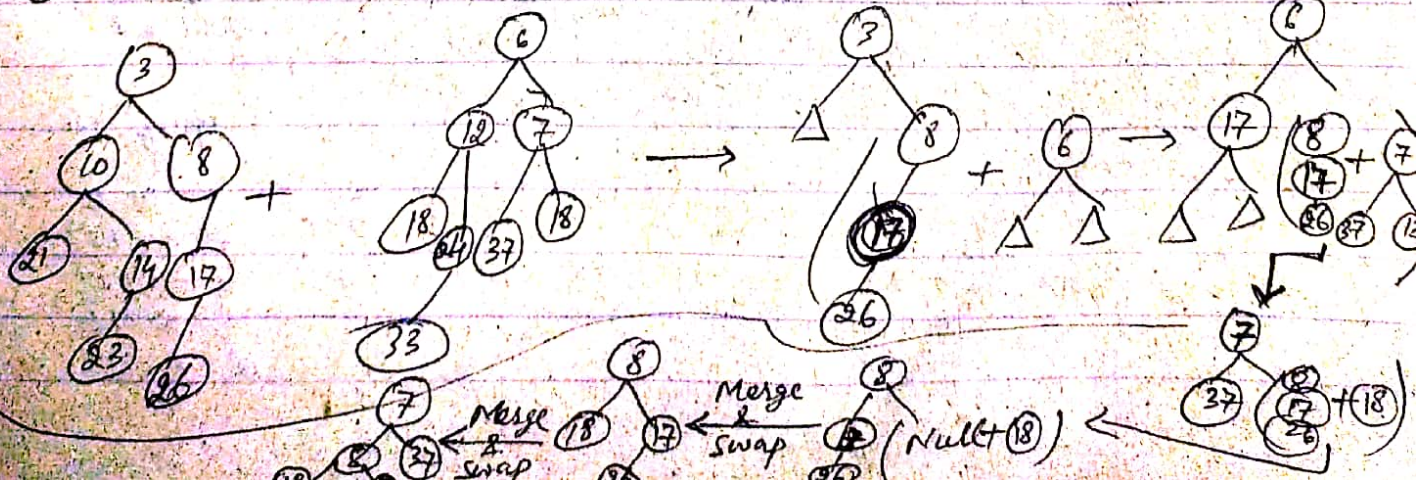
is drawn as left. for the last node, atleast left itself.

Ex-



Merges :





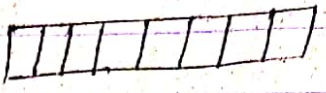
21/11/19

␣ → space

Encoding :

- Fixed length – Encode characters using bytes.

0 – 8 bits
1 – 8 bits
2 – "
"
9
a
b
c
"
"
"
z
A
B
C
"
"
Z
"



$2^8 = 256$

1 byte = 8 bits

I live in Jodhpur.

␣ → 3
i → 3
l → 1
v → 1
e → 1
|
. → 1

abaacdba

a → 4 → 0
b → 2 → 01
c → 1 → 11
d → 1 → 001

} Encoding

↑ code

a b a a c d b a
0 01 0 0 11 001 01 0

↓ Decode

Multiple outputs

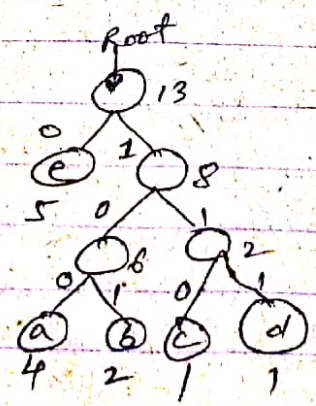
Corrected

e → 5
a → 4
b → 2
c → 1
d → 1

abaacdbaeeeee

e → 5
a → 4
b → 2
c → 1
d → 1

e a b c d
5 4 2 1 1



Left → 0
right → 1

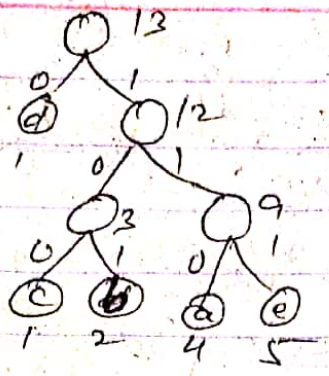
"Prefix"

e → 0
a → 100
b → 101
c → 110
d → 111

5+8×3 = 29 → descending

1+12×3 = 37 → Ascending

d → 0
c → 100
b → 101
a → 110
e → 111



No encoding (code) is a prefix for any other encoding (code).

22/11/19

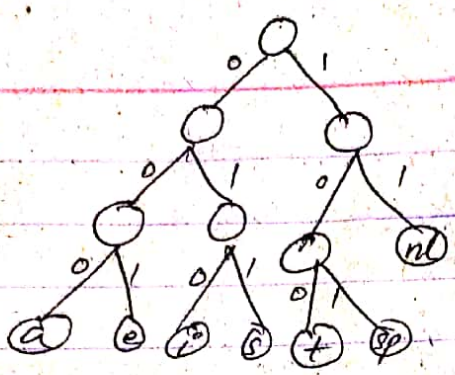
Heaps

Binary Heap	Leftist Heap	Skew Heaps
Insert and Delete → O(log N)	Insert, Delete and Merge → O(log N)	Insert and Delete and Merge → Amortized O(log N)
Merge → O(N)	(NPL)	for a seq. of m operations we will get O(m log N)

Huffman Coding :

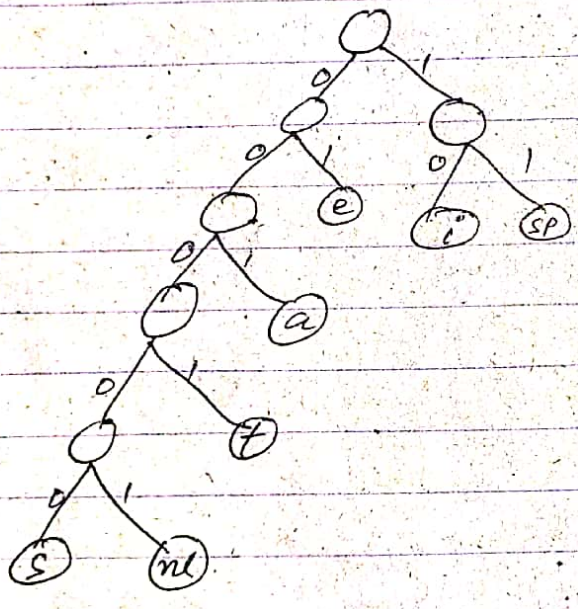
character	code	freq.	#Bits
a	000	10	30
e	001	15	45
i	010	12	36
s	011	3	9
t	100	4	12
sp	101	3	9
nl	110	1	3 2

Total = 144/143



full tree

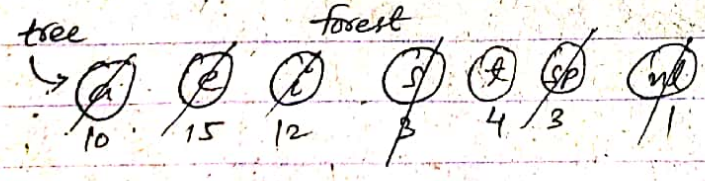
Our goal is to find the full binary tree with min. cost, where all chars are contained in leaf
(cost ≡ total no. of bits)

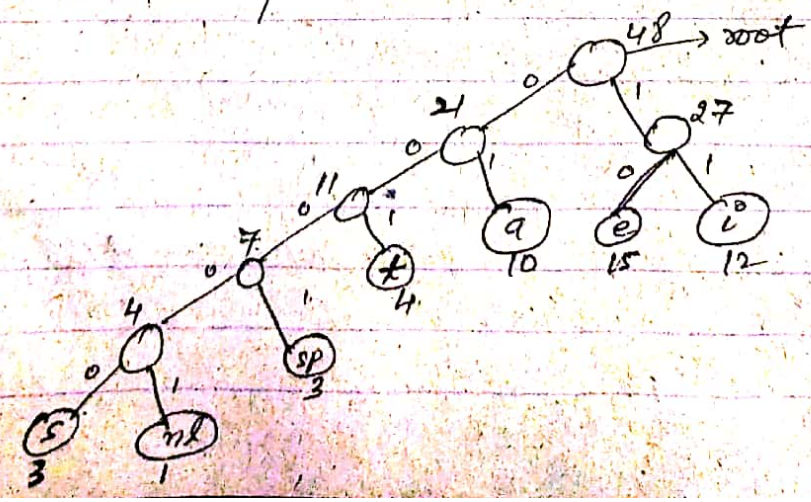


a → 001	30
e → 01	30
i → 10	24
s → 00000	15
t → 0001	16
sp → 11	6
nl → 00001	5
	126

$\left(\frac{18}{144}\right) \times 100 = 12.5\%$

tree → forest





Binary Tree

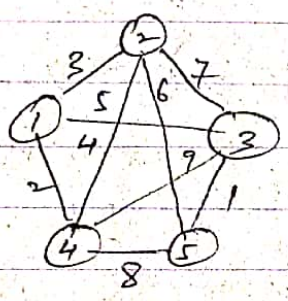
Priority Queue → List
→ 2 Delete Min
→ I Insert
→ 1 Merge

8 bits per char
2^8 $2^7 = 128$
1 bit → parity bit

25/11/19

KRUSKAL's Algorithm:

MST is an undirected graph



Edge	cost	Add(?)	1,2,3,4,5
(3,5)	1	✓	1,2{3,5},4
(1,4)	2	✓	{1,4},2,{3,5}
(1,2)	3	✓	(1,2,4),(3,5)
(2,4)	4	X	—
(1,3)	5	✓	(1,2,3,4,5)
(2,5)	6		Halt
(2,3)	7		
(4,5)	8		
(3,4)	9		

Union/find ADT:

Each set is represented by a tree.

find(x) – Return the root of the tree to which x belongs

$O(\log^* N)$

Union(x,y) – Make the root of Y as child of the root of X, (If the depth of Y's tree is less than that of X's tree)

(0) 1 2 3 4 5

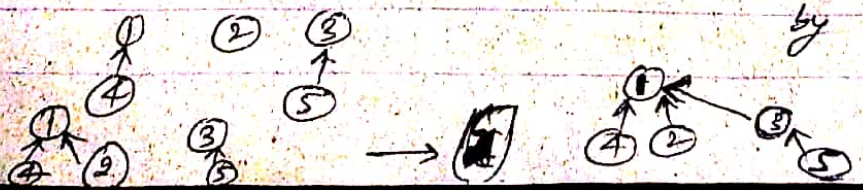
(1) 1 2 3 4

5

(2) 1 2 3

4 5

(3)



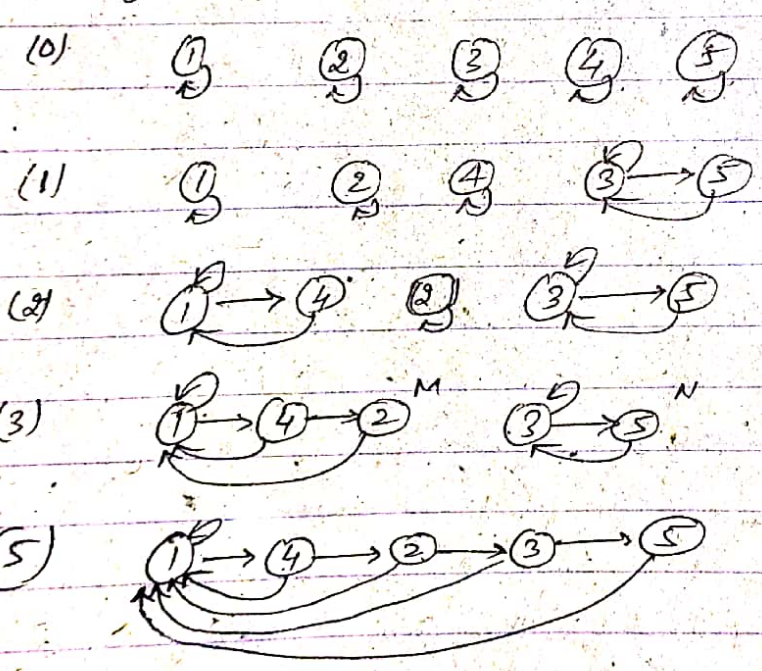
Each set is represented by its root

$\log^* N$ → smallest i such that after taking i logarithm, the value is ≤ 1.

$\log^*(512) = 4$
$\log(512) = 9$
$\log(9) = 3.\ldots$
$\log(3.\ldots) = 1.\ldots$
$\log(1.\ldots) < 1$

$\log^*(2^{1024}) = 5$
$\log(2^{1024}) = 1024$
$\log(1024) = 10$
$\log(10) = 3.\ldots$
$\log(3.\ldots) = 1.\ldots$
$\log(1.\ldots) \leq 1$

Using List:



The header node represents the set.

(M+N)
min(M,N)

Union/Find ADT:

Best known algorithm:
Union – O(1)
Find – Amortized $O(\alpha(M,N))$
$\alpha(M,N)$ – inverse Ackermann's function.

Ackermann's function

$A(i,j) = A(i-1, A(i, j-1)), \quad i, j \geq 2$

$A(i,j) = 2^i, \quad \forall j > 1$

$A(i,1) = A(i-1, 2), \quad \forall i \geq 1$

$\alpha(M,N) = \{ \text{Min } i \geq 1 \mid A(i, \lfloor M/N \rfloor) > \log N \}$

Lab

25/11/19

- Leftist Heap
- Skew heap
- Huffman Coding
- Union find
  - Using list
  - Using tree

28/11/19

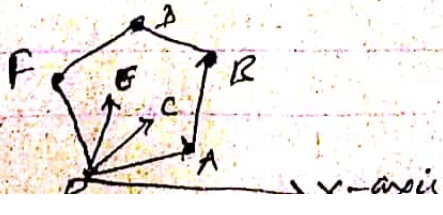
Convex Hull

convex Vs concave → Ex –

A Line joining any two points inside it lies inside

Ex – 

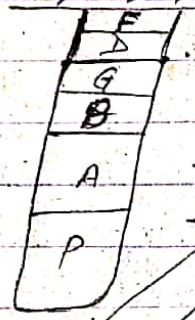
Given a set of points, find a convex-hull with the min area that covers all the points.



A, B, C, D, E, F

- first, identify the point with min-y-coord. value
If more than one such points, pick the one with min x-coord
- Let this point be denoted by P.
- Sort the remaining (2) points in increasing order of angles between PZ and x-axis

P, A, C, B, G, E, D, F



- Push (P)
- Push (A)
- P_2 = pop( ) → (x_2, y_2)
- P_1 = pop( ) → (x_1, y_1)
- P_3 = C → (x_3, y_3)

A P

Push (P)
Push (A)

Left Right

PAC → Left turn
PA AC
⇒ Push (C)

ACB → right turn
↓ AC & CB
Remove (top of stack)
Push (B)

ABG
BGE

P_2 = C
P_1 = A
P_3 = B

P_1P_2 P_2P_3

GED
pop( )
Push (D)

G, D, F
Push (F)

Cross product of P_1P_2 & P_1P_3

$$\overrightarrow{P_1P_2} = \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \end{bmatrix} \quad P_1P_3 = \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \end{bmatrix} = (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$$

$0 \Rightarrow$ Collinear
$+ve \Rightarrow$ Left turn
$-ve \Rightarrow$ Right turn