Tahsin

Name : Md. Tahsin.

ID : 2020-2-60-112.

Code : CSE207

Sem : Fall 21

Scanned with CamScanner

What is Data

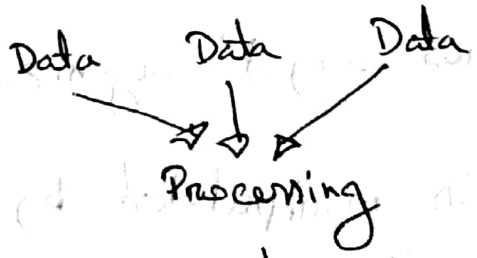
→ Values or set of values.

→ ex: name, quantity, nature of a person etc.

→ needs to represent in machine code to operate computer.
(0,1)

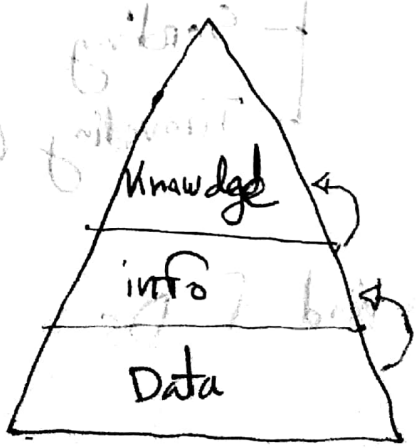
→ Data plurel form 'Detam'.

☆ Information :



অনেক গুলো Data সমষ্টি থাকলে info meaningful হয়।

☆ Knowledge



vol বেশি info বেশি।

Scanned with CamScanner

info process cycle

input → Process → output
↑↓
storage (থাকতে পারে না/ও পারে)

DS

→ set of variable(s).
→ Various way of organize data in a computer.
→ info is manipulated by systemic procedure called Algorithm.
→ Manipulation
- adding
- Searching
- Deleting
- Sorting
- Traversing (visiting on elements)

→ Array is a kind of Ds.

Scanned with CamScanner

ADT

group of 'DS, Algo interface function.' that used to solve a problem.

Data types [int, float, string]

→ Primitive and composite

(storage size) (range)

→ Char → 1 byte → 128 to 127 or 0 to 255

- int → 2 or 4 bytes → -32,768 to 32,767 or 2147,483,648 to 2,147,483,647.
- long → 4
- unsigned long.

Scanned with CamScanner

DS ও Datatype 2 types হতে → linear (Array, Stack ...)
→ non-linear (Tree type)

---

composite Data type → array, string, vectors.

DS — Primi
— compos
— relation

Basic Operations: Traversing, searching, Insertion, Deletion, Sorting, Merging.
(একটি DS কে একাধিক Merge করা যায়।)

Advantages of ADT:

- Some advantages over the convetional data structure.
- ADT is reusable and robust and is based on OOP and SE.
- Can be reused at several places.
- Encapsulation ensures the data can not be corrupted.

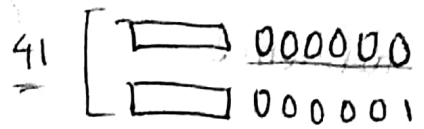
Scanned with CamScanner

27 Oct 21

Class-2

Pointer

✓ every variable has an address - কোনো variable define করলে computer memory তে store করে।



ex-

```
int a = 41;
```

✓ & (amporsion) যদি variable আগে দিই তাহলে memory address দেখাবে।

ex-

Printf(a) => 41;

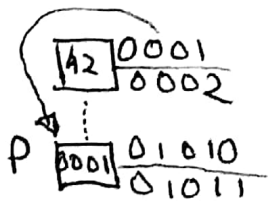
Printf(&a) => 000001.

✓ Pointer variable → a pointer is a kind of variable that hold another variable's address instade of regular value.

ex- {

```
int a = 42;
int *p;
p = &a;
```

[nb. *p দিলে 0001 এর পরের মান আসবে, 0002]



✓ cout << *p;
=> 42 (output).

✓ cout << &p
=> 0001

}

Nb. → int types এর Pointer এ শুধু int types এর variable. p=&c; char c;

↑

*P = 5; → [5] 0009 0001 হয়ে যাবে।

[0001 5]

Pointer vs Reference (55)

✓ Call by value : Value send করে।

✓ When you pass a variable by ref. to a fun what really happens ?

a ref. is just a simpler notation for passing by a pointer.

✓ Call by Reference : Value send না করে address send করা হয়।

Scanned with CamScanner

don'ts: (X)

```
int *i;
*i = 123456;   #mistakes.
```

→ Void swap(int *pa, int*pb){ //swaping.

```
int temp;
temp = *pa;
*pa = *pb;
*pb = temp; }
```

or,

```
int *temp ;
temp = pa ;
pa = pb;
pb = temp;
```

```
int main(){
  int a=5, b=6;
swap (&a &b);
cout << a;
cout << b; }
```

Scanned with CamScanner

No array নিয়ে কাজ করলে ' & ' চিহ্ন না pointer এ ।

Arrays, Address & Pointers (গুরুত্ব)

```
int nums[3]={1,2,3};
```

index নিয়ে।

```
cout << nums ; // Prints 01001 (example)
```

→ মানে nums[0]

0 তম index.

```
// & nums;
int *ptr = nums ; //pointer to array.
```

→ [01001 নিয়ে]

```
cout << *ptr;
```

[1st address থাকে অর্থাৎ 1st value Print করবে, অর্থাৎ, output = 1.]

```
cout << *(ptr+2)
```

[output = 3]

Scanned with CamScanner

Pointers on structure: use '*' to get to the structures and dot (.) to access the fields.

```
struct Nerd {
    int numzits;
    int hoursofStar;
};
int main() {
    Nerd carey;
    Nerd * ptr;

    ptr = &carey;

    (*ptr).numzits = 140;
}
```

Nerd is structure এর একটা variable যাকে আমরা carry এর মনে করতে পারি।

numzits	00001000
	1001
	1002
	1003

hoursOfstructm	00001004
	00001005
	00001006
	00001007

(ptr -> hoursofStar = 42;)

Scanned with CamScanner

✓ malloc

✓ Dynamic → যখন memory দরকার হয়।
memory.

✓ static → আগে থেকে memory ফিক্সড থাকে।

Dynamic Variable:

a = (int*) malloc (numbers* sizeof (int)); //memory allocation.

ex- Such as, a[5] = [ ][ ][ ][ ][ ] 5টি করে element হতে পারে, যদি 5টির বেশি দরকার হয়?

- এর জন্য malloc() function দরকার হবে, / calloc().

malloc():

ex ⇒ [SS → CSE 207]

```
int *a = NULL;
int num;
scanf ("%d", &num);
/* memory allocation */
a = (int*) malloc (numbers * sizeof(int));
ptr = (int*) "
if (ptr == NULL) {   // if memory = 0 / যদি
    Printf ("out of Memory");
}

for (i=0; i<num; i++){
    scanf ("%d", ptr+i);
```

Scanned with CamScanner

```
for (i=0, i<num , i++) {
    Printf("%d %d \n", Ptr+i, *(Ptr+i));

  free (Ptr); //Release allocated memory.
  }
}
```

অর্থাৎ যে adress storage থাকলো তার যেগুলো used adress free হয়ে গেছে এবং অন্য আবার use হবে।

Scanned with CamScanner

Time Complexity

Approach - assignment operation [a=10, b=20 etc]
Comparison
Mathematical
Func. call
Tasks inside the func.

Big-O :- a technique that lets you quickly understand an algorithm efficiently.
And compare it to others - so u can pick the best one!

- Worst case scenario → size of N.
  ↳ খারাপ সময়ে যে result দিতে পারে তাই উদ্দেশ্য।

ex:
```
a=10 ; — 1
b=20 ; — 1
```
[assignment operation = 1]

$f(n)=2$ [nb. শুধুমাত্র constant value হলে তাহলে $O(1)$ হবে, a=1, b=1, c=1 ∴ $O(1)$]

Scanned with CamScanner

ex -

✓ int arr [n][n]

compute f(n);

```
for (int i=0; i<n; i++)
    for (int j=0; j<n; j++)
        arr[i][j]=0;
```

✓ for (i=0, i<n, i++) { // for loop এর time এর মধ্যে,

assignment

1   n+1   n

a=a+i — n

}

✓ Assume n=3.

0<3	1 ~~বার check করল~~,
1<3	2
2<3	3
3<3	3=n বার বার করল
4 বার condition check করল, অর্থাৎ (n+1) বার	

∴ f(n) = n + n + 1

= 2n [1 constant value তাই ignore]

= n [" "]

✓ for (i=0; i<n; i++){ — n+1

for (j=0; j<n; j++) { — n × (n+1)

a [i][j]=0; — n × n

}

}

f(n) = 2n² + 2n + 1 ⇒ 2n² + 2n

Scanned with CamScanner

(rec)

$= 2n^2 + 2n$ [degree of polynomial – worst case so, $2n^2$]

$= 2n^2$ $\quad = 2n(n+1)$

$= n^2$ $\quad = 2n^2 = n^2$.

✓ for (int i=0; i<n; i+=2)

Sum++;

i++

i+=2 অর্থাৎ 2গুন বাড়ছে

তাই n/2 হারে হচ্ছে।

$\Rightarrow f(n) = n/2$

$= n$

$= O(n)$

✓ for (int i=0; i<n; i++) — n

for (int j=0; j<n*n; j++) — n*n

Sum++

$= n*(n*n)$

$\Rightarrow O(n^3)$

~~nested~~ nested Condition same n.

ex- for(i=0; i<a; i++){

}

for (i=0, i<b, i++){

$\Rightarrow a+b$.

Scanned with CamScanner

```
✓ K = n;
  while (K > 1) {
      sum++;
      K = K/2; }
```

$\Rightarrow$ $\log_2(n)$.

Example,

~~K = 30~~	K > 1
K = 15	
K = 8	
K = 4	
K = 2	
~~K = 1~~	

Ans:- $\log_2 16$ ~~হবে~~ (4) আসবে।

```
✓ for (int i=0; i<n; i++) — n
  { int K = n;
    while (K > 1) {
        sum++
        K = K/3
    }
```

$\Rightarrow n * \log_3(n)$

multiply এর → সাথে একটি polynomial (n) একটি logarithmic ($\log_3(n)$), তাই হবে।

```
✓ void foo () {
      int i; sum = 0;

  for (i=0; i<n*n; i++) — n^2
      sum += i;

  for (i=0; i<n*n*n; i++) — n^3
      sum += i;
```

$\Rightarrow$

$\Rightarrow n^2 + n^3$

$\rightarrow n^3 \rightarrow$ height degree

Scanned with CamScanner

nested হলে (*) → No. condition matter করে। (rec)

```
for (i=0; i<n; i++) {
    for (j=0; j<i; j++) {
        cout <<j;
    }
}
```

$\Rightarrow O(n^2)$

```
for (i=0; i<q*q; i++)
    for (j=0; j<i; j++)
        sum++;
```

q^2

q^2 বার Run হবে।

$\Rightarrow$

$\therefore q^4$

Scanned with CamScanner

lower order omit ⇒ $c^2 + c$
$\Rightarrow c^2$

Book exercise Practice

ex →

(i)
```
for (i=0; i<p; i++) {
    for (j=0; j<f; j++) {
    } }
```
$\Rightarrow p * f$ [nested]

(ii)
```
for (i=0; i<c; i++) {
    for (j=0; j<c; j++) {
    } }
    for (k=0; k<e; k++) { }
```
$\Rightarrow c^2 + e$.

(iii)
```
for (int i=0; i<n; i++) {
    if (i==n/2) {
        for (int k=0; k<q; k++)
    }
```

Scanned with CamScanner

Class - 6

Linked Lists

- used ~~to~~ in everything from video games to search engine. [all most সব জায়গায় কাজ লাগে।]

✓ Array's are great... But: ⊙ Fixed number of items..

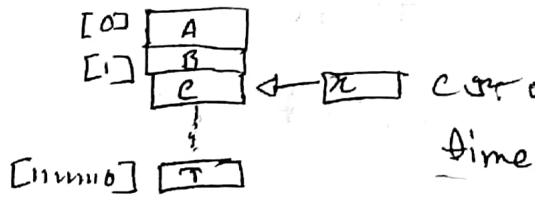
⊙ reserve enough slots....

⊙ wastage of memory.....

// So, in that case we can use Linked lists.

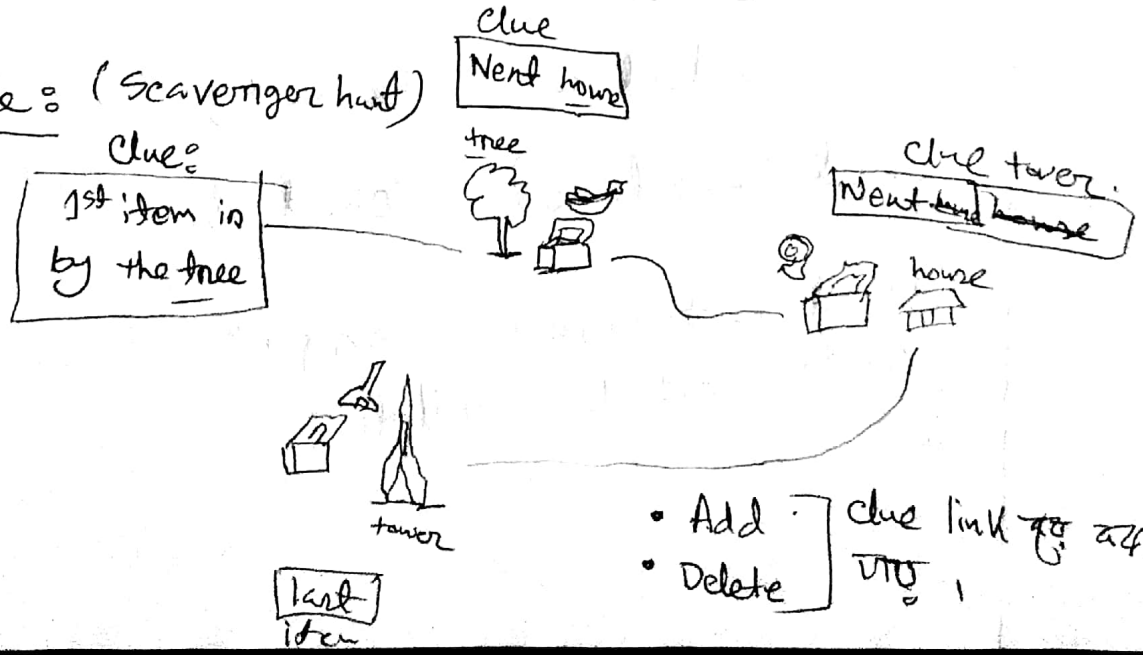
⊙ Values move করলে এবং Array size বড় হলে problem হবে।

ex-



C এর জায়গায় x বসালে (insert করলে অনেক time waste হবে।

example: (Scavenger hunt)



- Add
- Delete

] clue link হয়ে যাবে।

Scanned with CamScanner

linklist example: 1/11/21

Code (Scavenger hunt):

```
Struct Chest {
    String treasure;
    Chest *nextChest;
}

int main () {
    Chest *first;
    Chest chest1, chest2, chest3;
    first = & chest1;
    chest1.treasure = "toast";
    chest1.nextChest = & chest2;

    chest2.treasure = "bacon";
    chest2.nextChest = &chest3;

    chest3.treasure = "eggs";
    chest3.next Chest = nullptr;
}
```

// let's use a string variable to hold our treasure; eg: "Plunger".

Scanned with CamScanner

Why linked list: each element in the list in 'linked' by a pointer to the next element.

how it works (code)

called 'head'

first → 1000

Chest 1 (1000)
- treasure: "toast"
- nextChest: 1020

Chest 2 (1020)
- treasure: "bacon"
- nextChest: 1040

Chest 3 (1040)
- treasure: "eggs"
- nextChest: "nullptr"

[NB: Normally we dont use local variable to create linked list.] 41. min rec

↳ +

[Instead we use dynamically-allocated variable (and pointers!)]

first = &Chest 1;

(✗) Chest1.treasure = "toast"

(✓) first = new chest;
first → treasure = "toast"

Scanned with CamScanner

(2) struct Node {

```
int StdID;
string name;
int phoneNumb;
float GPA;
Node *next; }
```

Do it ---- ✓

⇐ struct Node {

```
int data;
struct Node *next; }; OR } *head, *second;

int main () {
struct Node* head = NULL;
struct Node* second = NULL;

head = (struct Node*) malloc (sizeof (struc
second = "
;

head -> data = 3;
Printf(" ,! ");
```

Scanned with CamScanner

```
head -> next = second;
second -> data = '6';
Printf(" ");
```

3/11/21

Class-7

insert in linked list:

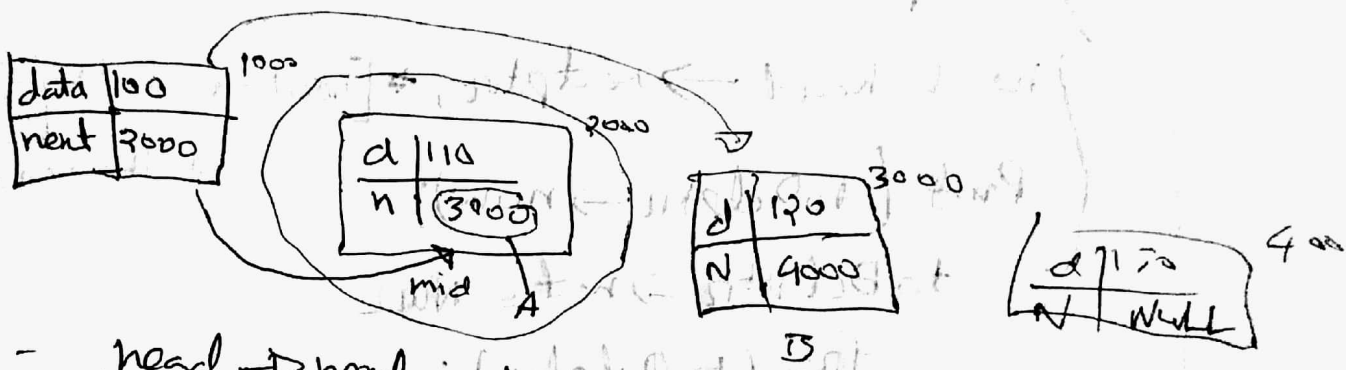
56 min

Delete in linked list:

/কখন কখন এসে থাকে।

3 ভাবে একটা যায়- head, mid, last.

(mid node delete)



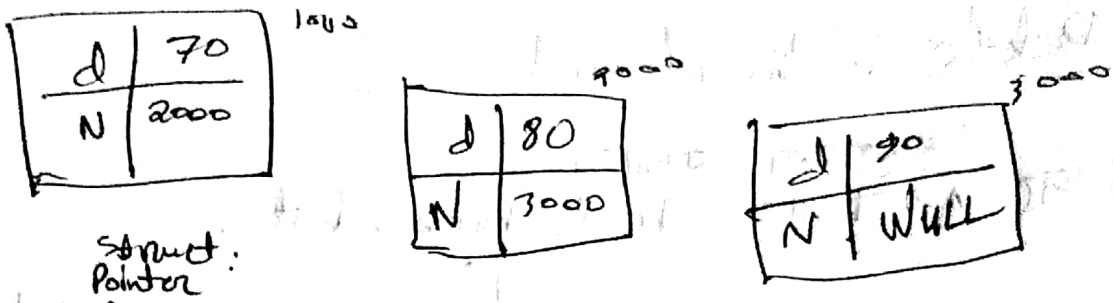
- head -> next = A -> Next
  free(A);
  তাহলে, head -> next = head -> next -> next.

[Rev code]

Rev
temp1
temp2
head

```
temp2 = head -> link;
head -> link = temp1;
temp1 = head;
head = temp2;
```

1st Node Delete:



struct Pointer

```
toDelptr = head;
head = head -> nextptr;   * [head = 2000 ... so ... d 80 N 3000 in the head.]
Print (toDelptr -> num);
toDelptr -> next = NULL;
free (toDelptr);
```

Last Value Deletion:

Last এর আগের Node এর Next কে NULL করতে হবে,

1.5 hrs.

(4-C) code.

Insert: (i) head এর আগে একটি Node add (New head)
(ii) মাঝখানে, (iii) last এ add করা যাবে.

(i) =>

```
fnNode -> Num = num;
fnNode -> Nextptr = head; //
head = fnNode; // makes it head
```

(iii) =>

Class-9

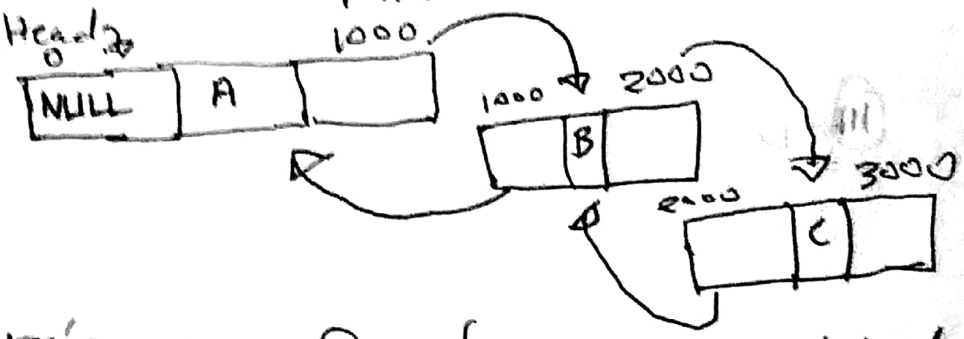
Doubly Linked list

• Pointer দিয়ে হবে,

Ex: Linked list — [data | next ptr]

doubly ... — [Prev. ptr | Next ptr]

Head → 1000



তাহলে, এ লেখা দিকে যাওয়া যায়, আবার Linked List এ last দিয়ে শুরু করা যায়,

Prob - (i) delete - C -> Cprev -> next = C -> next;

free (C);

Note: Linked list এ বাদ হলে Loop লাগবে।

→ Prev.ptr আসা লাগবে, so; C -> next -> Prev = C -> Prev;

(ii) insert = Last -> C;

C -> Next = D;

D -> Prev = C;

F -> Next = C;

F -> Prev = B;

B -> next = F;

C -> Prev = F;

[Last এ D Node insert করলে,

[B এবং এর মাঝখানে F আসছে,

Stack / LIFO (Last in first out)

- Stack is a linear list which all additions and deletion are restricted to one end, called top.

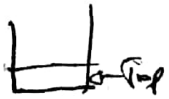
[top ও value থাকে।]

- If you insert data series as {5, 10, 15, 20} in removed as {20, 15, 10, 5}. // like inverts / reverse.

- Basic operation – ◦ Push (Value insert) ; Pop (Delete),
  ◦ StackTop (Top Value).

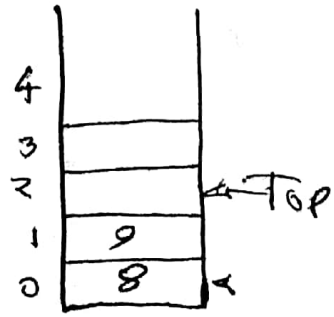
ex-

① Pop :

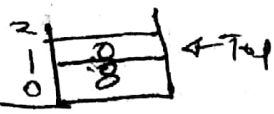


② Push : 8

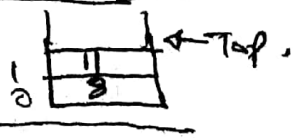
③ Push : 9



④ Pop



⑤ Push : 11



Scanned with CamScanner

Common Use of stacks -

- Storing undo items for your word processor.

  ex → 'ctrl + z' দিলে শেষ দিকের কাজগুলো পাওয়া যায়, //typing

- Evaluating mathematical expressions -

  5 + 6*3 → 23 // Sum এর আগে পড়ে, multiply এর আগে আসে এর উল্টো হলে,

- Converting infix to postfix expression. -

  A + B → AB +

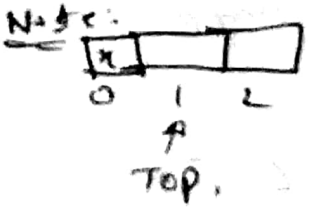
- Solving maze.

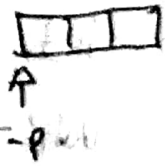
2 min

Code:
```
#include < >
int stack[100];
void push (void);          //1.Push
void pop (void);           2. Pop
void display (void);       3. Display
                           4.
int main(){
top=-1; scanf ("%d", n);
if (n==1) { push();
            break; }
if (n==2) {
```

```
Void push() {
    if (top >= n-1){
        Printf ("stack in overflow");}
    else {
        Scanf ("%d", &x);
        top++; // top=-1 হলে পরিবর্তন হয়ে 0 হবে,
        stack [top] = x; } // এ input হয়ে এ
                              show করবে।
```

Note:



the push করলে 

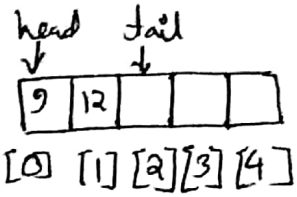

```
Void display() {
    if (top>=0) {
    for(int=top ; i>=0, i--)
        Printf("%d", stack [i]); }}
    else {"empty"}
}
```

Class-11

Queue (FIFO) → first in first Out.

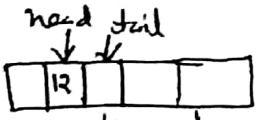
Queue		stacks	
ENQUEUE	↔	Push	(add / insert)
DEQUEUE	↔	Pop	(delete)

ex- int a[5] so,



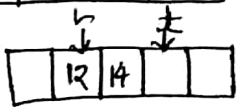
(i)

then,



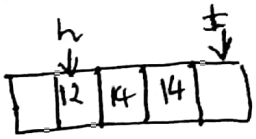
(ii) DeQUEUE (9)

then,



(iii) EnQUEUE (14)

then,

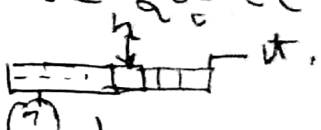


(iv) " (14)

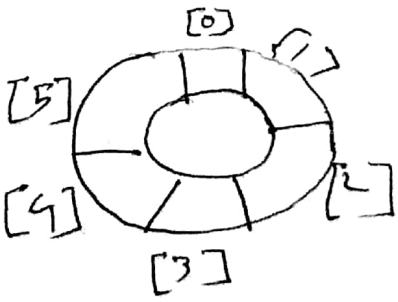
↳ আর max 1টি value রাখা যাবে,

– আর্গুমেন্ট এ শেষে head এখান থেকে Queue শুরু হবে,

✓ So, আগের Part গুলো waste যাবে ? ⇒



⇒ Circuler use করে বলা যাবে,



✓ Code: a[0] --- a[7]

a[0] a[7]

⊛ mode → array size.

Scanned with CamScanner

Search & Sort

Search: (linear search) – Sorted হওয়া লাগবে না, here,

12	12	18	20	25

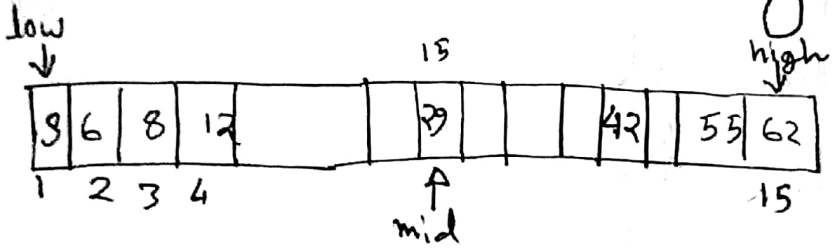
← 20 find?

```
for (i=0, i<n, i++) {
    if a[i] == 20
        Printf ("Yo! done");
}
```

– Complexity $O(n)$.

(binary search) – lists are sorted.

- why binary? ⇒ Just for reducing time complexity!

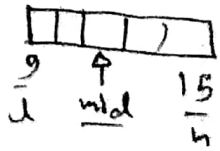


here, low = 3; Position = 1

high = 62; ; Pos = 15

$$mid = \frac{low + high(Position)}{2} \quad \left[\frac{1+15}{2} = 8\right]$$

note: find 42? → So; mid value তে 42 compare করে list এর half side নিয়ে যাব যেখানে হবে, Then



নিয়ে যাব,

– Complexity $\log_2(n)$.

Scanned with CamScanner

Code: Bin Search (A, n, Key)

```
{ L = 1, h = n;
    while(L <= h){
    mid = L+h/2 ;
  if ( Key == A [mid]){ return mid; };
  else if (Key < A[mid]){
      h = mid-1; }
        else{ L = mid+1; };
```

Sorting : types -

13 class

Rec (Prev Sem-1)

(I) Selection Sort.

(II) Bubble Sort.

(III) Insertion Sort.

(IV) Merge Sort.

(V) Quick Sort.

Scanned with CamScanner

(I) Selection Sort - (One by One করে আসাতে)

12	15	8	2	9

- temp
- max, min হবে.
- loop -
  1st (1 - n)
  2nd (2 - n)
  ⋮

⇒ 2 8 9 12 15

(I) → 2 15 8 12 9

(II) → 2 8 15 12 9
sorted | unsorted

$O(n^2)$ time complexity

(Web - Check for visualize code
Sorting — Visual Algo)

(II) Bubble Sort: (যে সংখ্যা সবচেয়ে বড় তা উপরে নিয়ে আসা- হয়; like as bubble ।)

8 12 2 4 6

8 12 2 4 6

8 2 12 4 6

8 2 4 12 6

8 2 4 6 12 ✓

8 2 4 6 12

2 8 4 6 12

4 2

⋮

2 4 6 8 12 — ✓

✓ প্রথম দুটো check করে যদি ছোট হয় তাহলে swap. নিচ্ছে,

✓ selection sorting এর প্রথমটি হবে last পর্যন্ত যেতে।

Note: সবচেয়ে বড় সংখ্যাকে শেষে last এ আসবে।

Scanned with CamScanner

Selection Sort : ~~Ist~~ Part E

```
for (c=0; c<n; c++)
        scan(        arr[c]);
for (c=0; c<(n-1), c++)// finding min element
                                  (n-1)th
{
    for (d=c+1; d<n; d+1){
        if (arr[Pos] > arr[d]) {
            Pos = d; }}
        if (Pos != c) {
            t = arr[c];
            arr[c] = arr[Pos];
            arr[Pos] = t; }}
```

Scanned with CamScanner

Bubble: - Complint (n^2)

```
//scanf (n) arr[c]
for (c=0; c<n-1; c++){
    for (d=0; d<n-c-1; d++){
        if (array[d] > array[d+1]){
            //swap
            swap = arr[d]
            arr[d] = arr[d+1];
            arr[d+1] = swap; }}}
```

individual check ⇓ height value পাবে।

সব ২টা value check

✓ **Innertion sort:** ✓ Sorted list থাকা অবস্থায় যতক্ষণ যদি।

✓ then new value assign করতে sorting এ যায়।

12	20	50	40

(40) - assign করতে হবে।

✓ 3 38 5 12

3 38 sorted 5 12
এইটুকু।

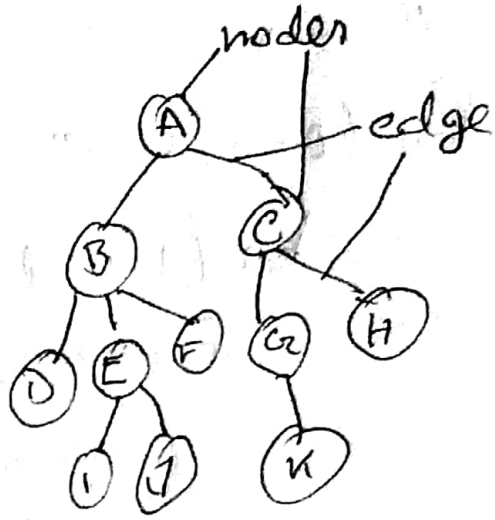
3 5 38 12
sort

3 5 12 38
sorted.

Scanned with CamScanner

BSC → site

Tree



- ✓ Parent ব্যক্তি হল।
- ✓ indivinual elements are node.
- ✓ if a tree ⇒ 'n' numbers of nodes there will be max (n-1) numbers of edge.
- ✓ (A) → Root node.
- ✓ (A) → Parent → child → (B) (C), (B) → Parent → child D, E, F
- ✓ (B) (C) child where (A) Parent.
- ✓ (B), (C) are sibilings, same as, (D, E, F), (G, H), (I J) etc.
- ✓ who'n (node) doesn't have child is called leaf node. / terminal / external.
- ✓ কমপক্ষে 1টি child থাকলে internal node হবে / non-leaf.

Scanned with CamScanner

✓ **Degree** – total numb'n of child

here, Degree of B in 3.
" " A " 2.
" " F " 0.

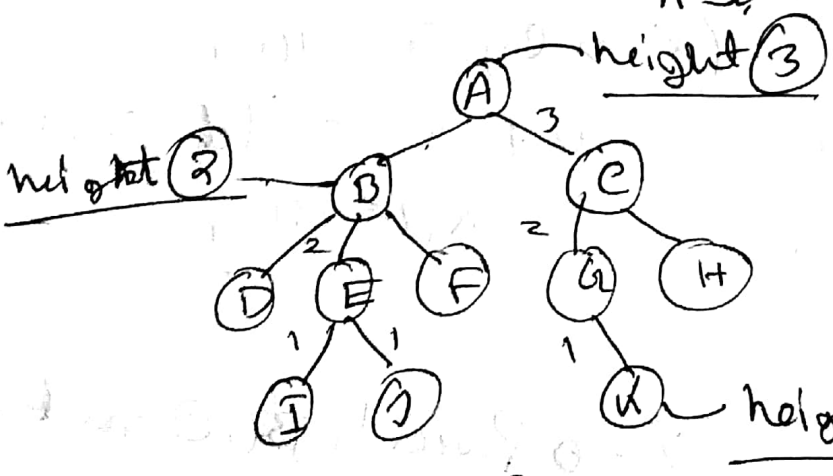
height numb of node degre in called "degree of tree"

✓ **level:**

(A) – lvl 0

(B) (C) – lvl 1

✓ **height:** leaf node থেকে Root node এ যেতে A এ.



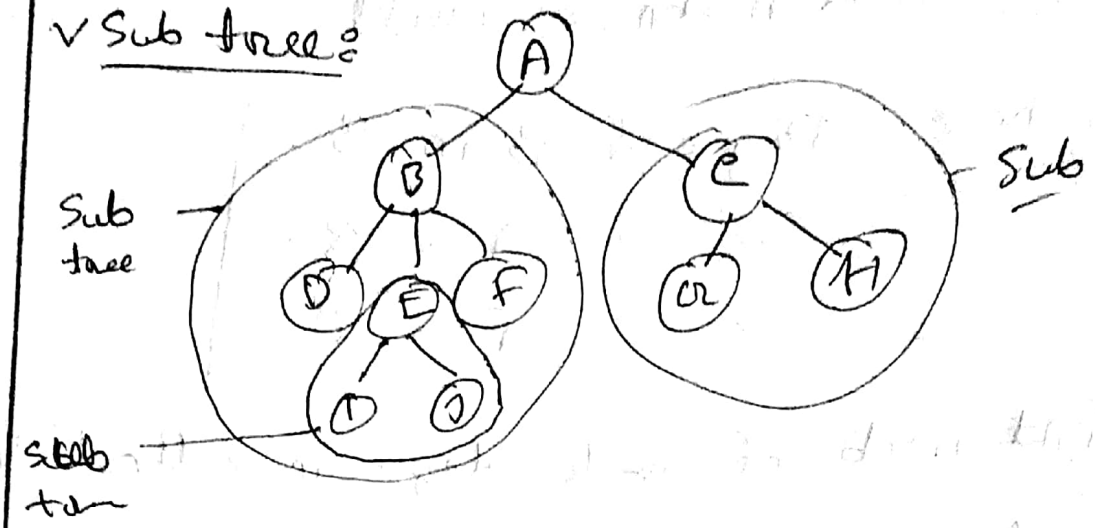
[leaf → Root]

✓ **Depth:** height এর উল্টো, [Root → leaf]

✓ **Path:** (A) একটি node থেকে অন্য node যেতে

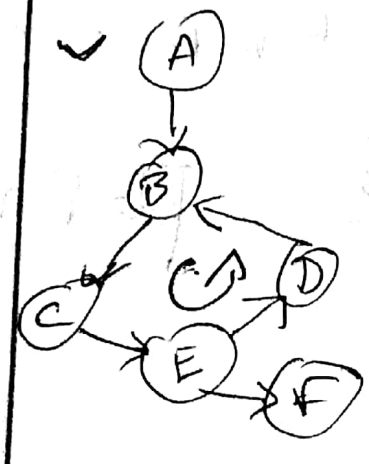
(A) & (J) ⇒ A – B – E – J
Path

Sub tree:

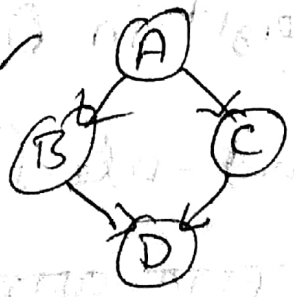


Ques:

A → B ✓ ✓ X → cycle হয় না,



X → cycle আছে,
→ E এর child B হতে পারে না।

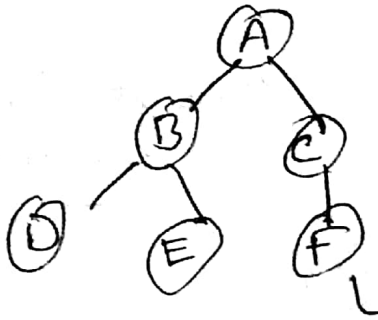


X → Parent B, C not possible.

✓ Structure for Tree Representation.

Binary Tree

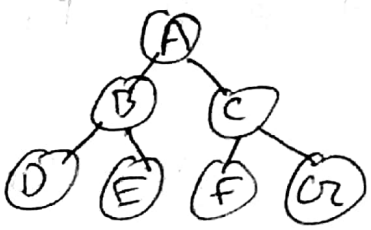
Condition: প্রতি node এর দুটি child (max).



- it's a bin tree.

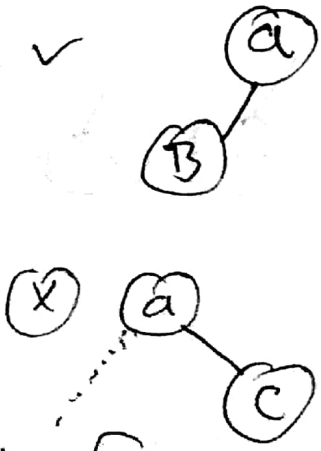
→ it's not a full bin tree..

✓ Full Bin Tree: (child max দুই হবে,)



✓ Complete bin tree: Condition: প্রতি node এর left node থাকা mandatory

✓



left নাই
so, হবে না,

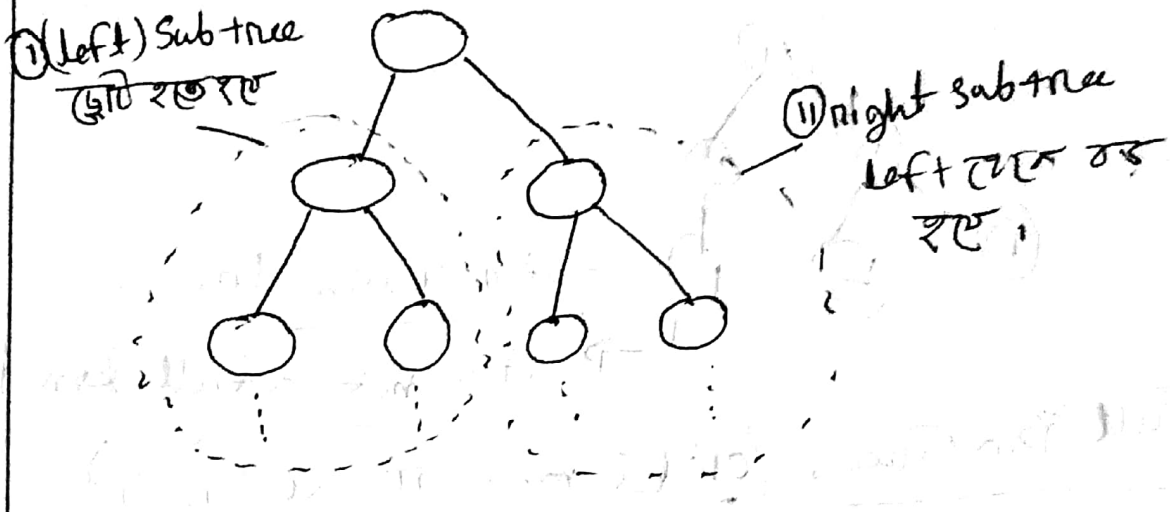
```
struct node {
    int data;
    node * left;
    node * right;
}
```

Scanned with CamScanner

6/12/21

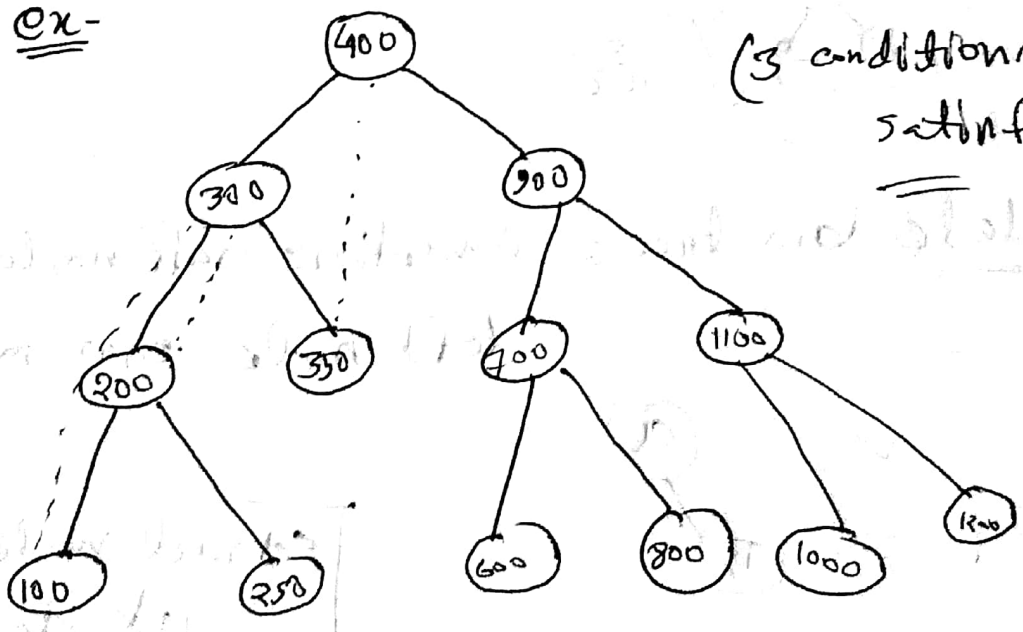
Binary Search Tree: (BST)

[difference - values থেকে যাবে]



(iii) কোনো duplicate node হতে পারবে না।

Ex-



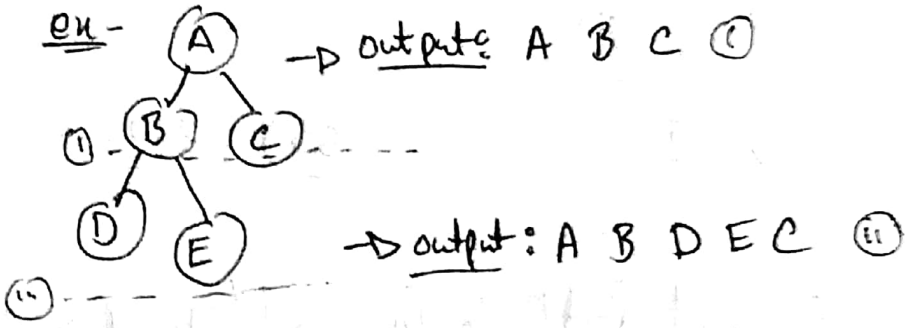
(3 conditions satisfied)

Scanned with CamScanner

BST Traversal : ◦ Pre Order ◦ in-Order ◦ Post Order.

✓ **Pre Order:** Root -> Left -> Right

ex-



-> output: A B C (i)

-> output: A B D E C (ii)

Code:

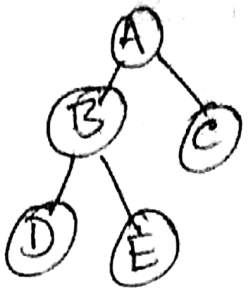
```
Void PreOrderTraversal (Struct node * root){
    if (root == NULL) return;
    Printf ("%d -> ", root -> item);
    PreOrderTraversal (root -> left); //Recursion.
    PreOrderTraversal (root -> right); }
```

Scanned with CamScanner

Post Order:

Left - Right -> Root

ex -



out: D E B C A

(SS) -> 207.6

Code:
```
Void PostOrderTraversal (struct node *root) {
    if (root == NULL) return;
    Print PostOrder (Root -> left);
    "        (Root -> Right);
    Printf("%d", root -> item); }
```

InOrder: Left -> Root -> Right

Tree

100 200 250
300 350 400
600 700 800
500 1000 1100
1200

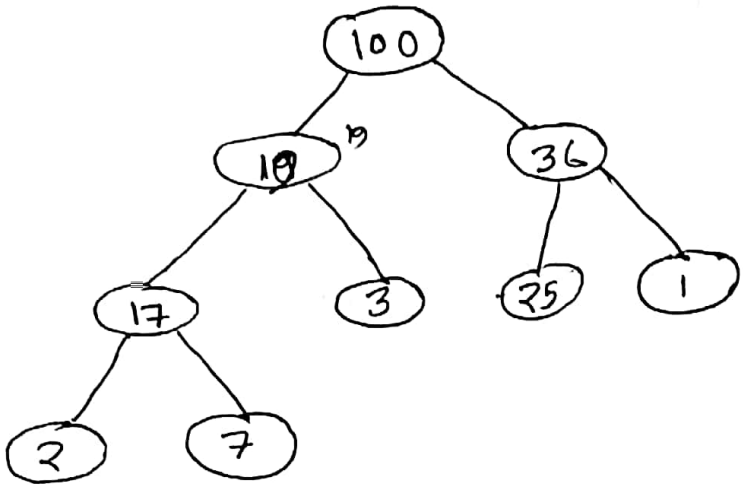
(SS) -> 207.6

```
inOrder (root -> left)
Print ("%d ->", root -> item);
inOrder (root -> Right) }
```

Scanned with CamScanner

Heap:

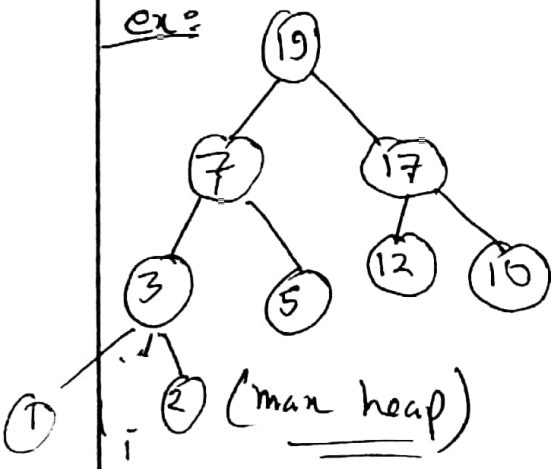
Condition: (i) Bintree হতে হবে,
(ii) Complete Bintree,



- Max heap: Root node এর Val বড় হবে, (Every Root)
- Min heap: Root node এর Value ছোট হবে,

why 0? - 1 2

ex:



(max heap)

max heap creation: tip: Serially left-Right maintain করতে হবে,

int heap = 9

int heap[ ] = | 0 | 19 | 7 | 17 | 3 | 5 | 12 | 10 | 1 | 2 |
0 1 2 3 4 5 6 7 8 9

heap[1] = 19
heap[2] = 7
heap[3] = 19

left এর ক্ষেত্রে logic → heap[i * 2] — যেটা বড় হবে,
Right " " logic → heap[i * 2 + 1]

Scanned with CamScanner

ex =>
Idea

(4) এর child -> n = 4;

left child = (n * 2)

Right child = (n * 2 + 1)

(node এর left & Right child বের করে)

ex =>

index / 2

(Root Parent বের করে)

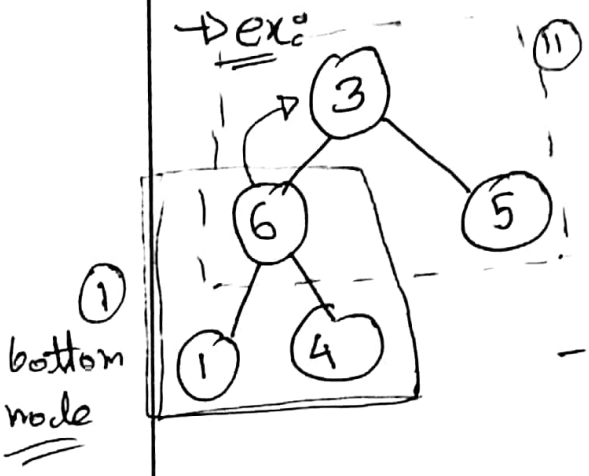
Code:

```
int left(int i){ return i*2; }
int right(int i){ return i*2+1; }
int Parent(int i){ return i/2 ; }
```

Scanned with CamScanner

- How to Convert a tree to Max heap?

→ ex:



[it's a tree but not a Max heap]

- So, এটিকে heapify করতে হলে - - -

(i) • bottom node কে দিয়ে করতে হবে।
(replace) ✓ সবচেয়ে বড় val Root এ আসবে।
(i), (ii) - - ।

(ii) Top nod কে ধরে নিচে নেমে হবে আবার, যাতে Parent গুলো বড় থাকে।

Scanned with CamScanner

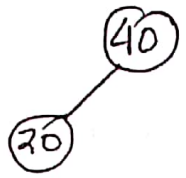
AVL Tree

Note: 1st we have to learn about -

- BST
- Drawbacks of BST.
- How to improve the drawbacks of BST.

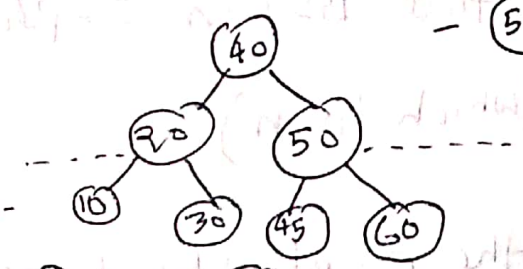
BST → ~~Root~~ Right বড় হলে, Root থেকে।

(i)



- 20 insert করার সময় check করে Root থেকে ছোট কিনা, ছোট তাই left এ গেল

(ii) 50 insert করলে।



- 50 " .... বড় তাই Right হল।

ex-

(iii) find 30 ?

- 1st এ Root এর সাথে check করতে হবে।
  - → 30 < 40 ; so 40 হবে না.
  - → 30 > 20 ; so, Root node 20 এ Right এ check করবে।

∴ ২টি height এর মাধ্যমে complete হলো।
3 Steps

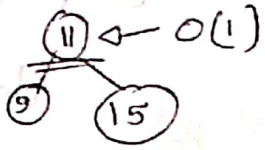
- Complexity - $\log n$. [কারণ যতগুলো node থাকে এর half-half করতে থাকে।]

Scanned with CamScanner

• We always try to find the worst case complexity

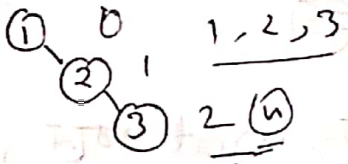
• For BST :

(i) Best case : $O(1)$ [কারণ যদি First Root node টি এর বদলে হয় তাহলে অ-তাহলে শেষ পাবে।]



(ii) Avg Case : $\log n$

(iii) worst case : n

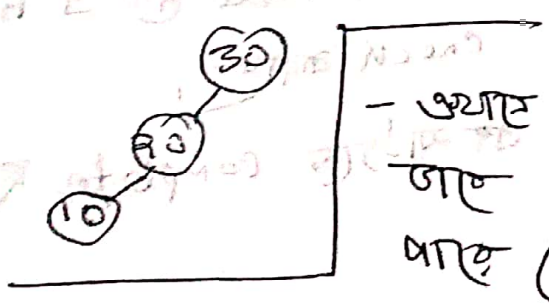


→ So, here we know that BST's complexity is not good. (which is n).

→ So, To make the complexity better we have to use AVL tree.

— o —

ex:
30, 20, 10 – make a BST →



– এখানে 3টি order এ আসতে পারে combination করা যেতে পারে (n!) অর্থাৎ- 3! => 6টি

– So, BST অনেক types হতে পারে,

⊛ Complinity তোমরা always height তোমরা নাগবে।
(Count করা)

ex- Value (nodes) থাকা height তোমরা বুঝা যায়।
↳ Value & height same হলে better.

AVL Tree : একটি Tree কে logically change করি হলো AVL এর নাম।

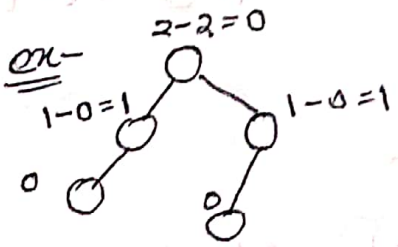
∘ balance factor : height of left sub tree − height of right sub-tree.

$$\boxed{bf = hl - hr} = \{-1, 0, 1\}$$

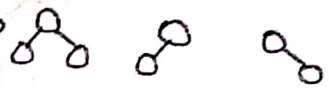
↓

এই উক্তি গুলোর Value থাকলে bf হবে।

ex-



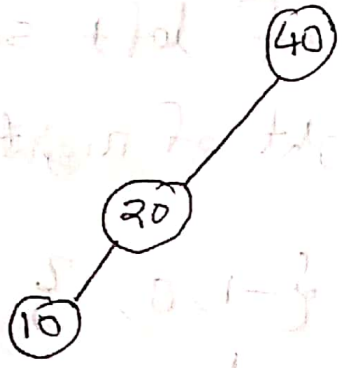
; so এই Tree কে convert করার পরে করে যাবে $\{0, -1, +1\}$ range এ আছে।

∘ why $\{+1, 0, -1\}$ এ থাকা লাগবে? ⇒  ← best form. 0, 1, -1.

AVL Tree Key

ex: → 40, 20, 10, 25, 30, 22, 50

Nb. BST এর logic এ হবে।

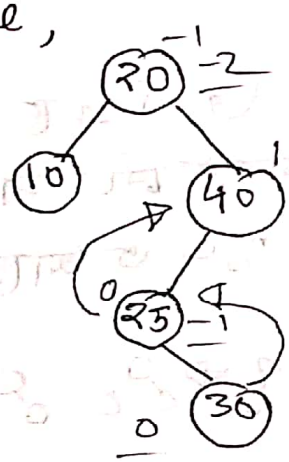


→ (i) 40 add করার bf = 0.
always bf count করতে হবে।

→ (ii) 20 add করার পর 1−0 = 1.
so, No problem.

→ (iii) 10 add করার পর bf = 2.
∴ এখন যেহেতু 2 → bf so,
Rotation apply করতে হবে।

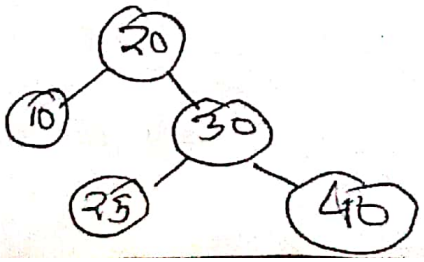
Therefore,



(iv)
→ 25 add করার
∴ bf = −1

→ (v) 30 add করার,
∴ bf = −2 (20)
2 (40)

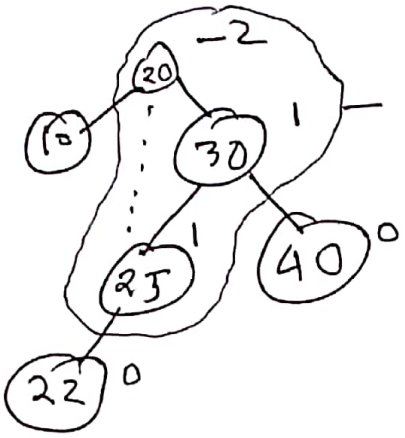
∴ এখানে দুই পাশের Problem. So, leaf থেকে এনে হবে।



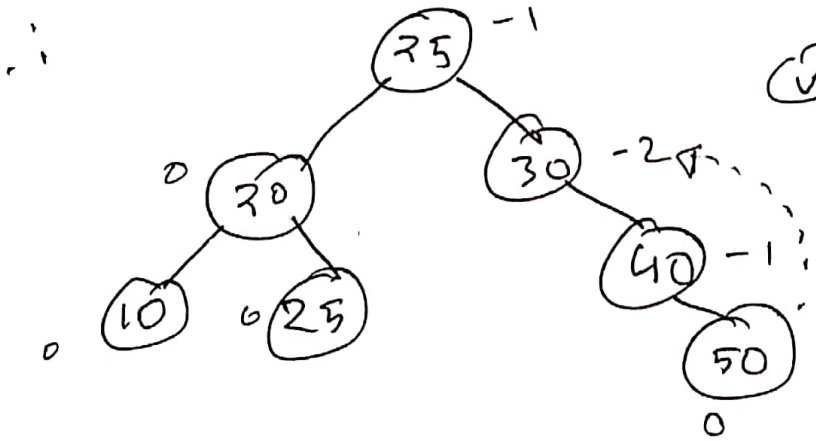
(এখানে check :)

Scanned with CamScanner

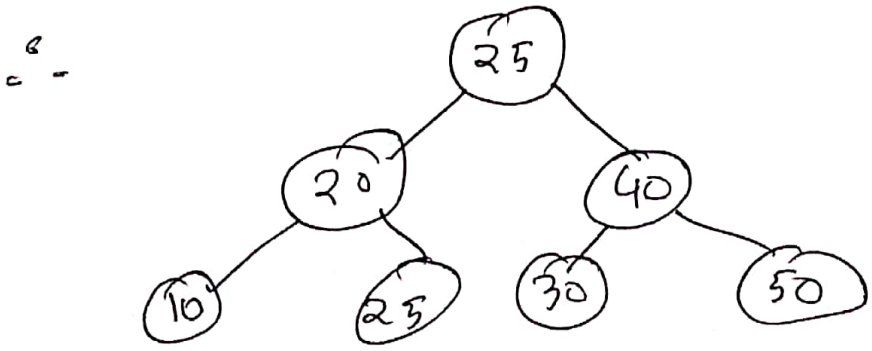
(vi) Now, adding (22) → Bf: (20) -2



যেহেতু -2 হয়েছে 30 25 এই এই নিয়ে আর,



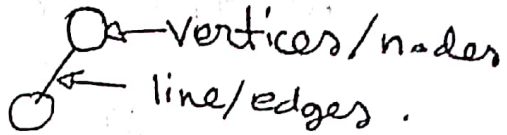
(v) (50) add



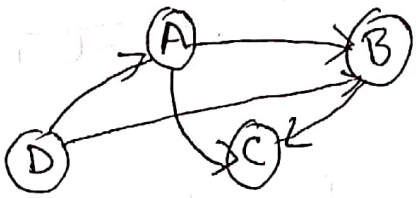
Scanned with CamScanner

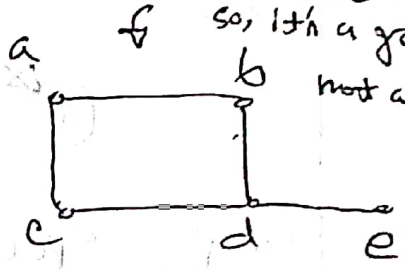
Graph

- nodes - called vertices and collection of segments called line or edges.



- graph is a pictorial representation of a set of object's are connected.





- No Root node here. So, it's a graph not a tree.

- Then what is the difference between Tree & Graph?

→ For tree → Root থেকে নিচের দিকে যাওয়া যায় অর্থাৎ - starting point Root হিসেবে ধরা হয়.

- যা Graph এর ক্ষেত্রে প্রযোজ্য নয়,

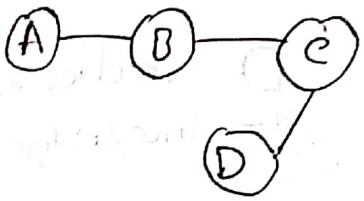
- Adjancy - A — B — C -

Here, A & C are adjanc to B.

A থেকে B, B থেকে A, অর্থাৎ যেকোনো দিকে যাওয়া যায়,

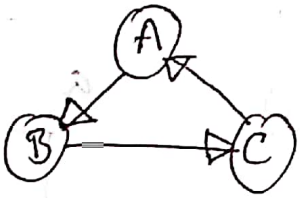
Scanned with CamScanner

Path: a sequence of edges between two vertices.



থেকে (A) থেকে (D) এর Path হল, ABCD.

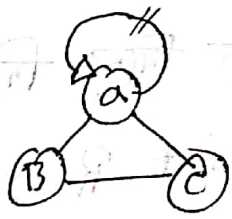
Cycle:



• 3টি node minimum হতে হবে।

Loop: special case of cycle.

diff: Cycle vs loop: a single arc in loop.



Directed graph: (a) → (b)

UnDirected graph: (A) — (b).

Connected graph: সবাই সবার সাথে Connected.

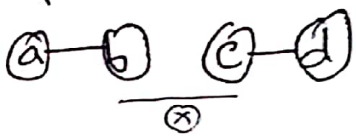
Scanned with CamScanner

Strongly Connected: A directed graph is strongly connected if there is a path from each node to.

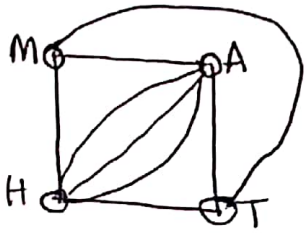
Weakly Connected: A directed graph is weakly connected if at least two nodes/vertices are not connected.

Disjoint Graph: if graph is not connected.

ex: a graph.



Degree: প্রত্যেক node's এর edges এর ending (directed) point's.



Vertex	degree
M	3
A	5
T	3
H	5

- Indegree: number of edge's entering the vertex.
- Outdegree: " " leaving " " of bigraph.

Scanned with CamScanner

Operations

Add vertex – inserting a new vertex

Delete vertex – Removing all connected ~~ve~~ edge.

Add edge

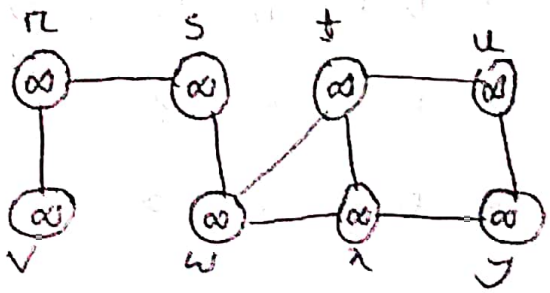
Delete edge

Find vertex

Traverse Graph –
- Depth-first Traversal / Search. (DFS) *
- Breadth-first Traversal. (BFS) *

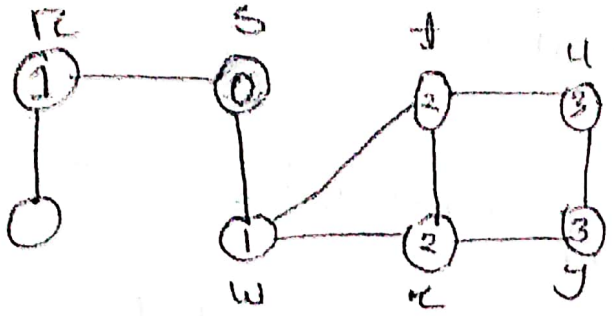
DFS: (next)

BFS: Breadth-First Search



	r	s	t	u	v	w	x	y
- Colour :	W	W	.	.	.	.	.	.
- distance(d) :	∞	∞	.	.	.	.	.	.
- Prev. :	nill	nill	.	.	.	.	.	.

Scanned with CamScanner



Notes:

- (s) থেকে start হবে
  $d = 0$, $c =$ (illegible)
  Prev = null.

- (s) এর পর (w) & (r)
  $d = 1$, Prev = (s)

Q : | s | w | r |

- (w) নিয়ে যাব বন্ধুরা (x) & (t).
  $d = 2$, Prev = (w)

- (t) থেকে (u) ; • (x) থেকে (y) ; [(u) থেকে (y) এলো কিন্তু (y) already traversed so হবে না,

[nb: colour change নাই]

Queue : s w r t x v u y.

new →

Vertex :	r	s	t	u	v	w	x	y	
Colour :	B	B	B	B	B	B	B	B	← সব শেষে সবই Black.
distance :	1	0	2	3	2	1	2	3	
Prev :	s	null	w	t	r	s	w	x	

Scanned with CamScanner

Breadth-First Search: The Code

```
Data: color[V], prev[V],d[V]

BFS(G) // starts from here
{
    for each vertex u ∈ V-{s}
    {
        color[u]=WHITE;
        prev[u]=NIL;
        d[u]=inf;
    }
    color[s]=GRAY;
    d[s]=0; prev[s]=NIL;
    Q=empty;
    ENQUEUE(Q,s);
```

```
While(Q not empty)
{
    u = DEQUEUE(Q);
    for each v ∈ adj[u]{
        if (color[v] ==
    WHITE){
            color[v] = GREY;
            d[v] = d[u] + 1;
            prev[v] = u;
            Enqueue(Q, v);
        }
    }
    color[u] = BLACK;
}
}
```


Scanned with CamScanner