Tahsin

Name : Md. Tahsin.
ID : 2020-2-60-112
Code : CSE207
Sem : Fall 21

What is Data

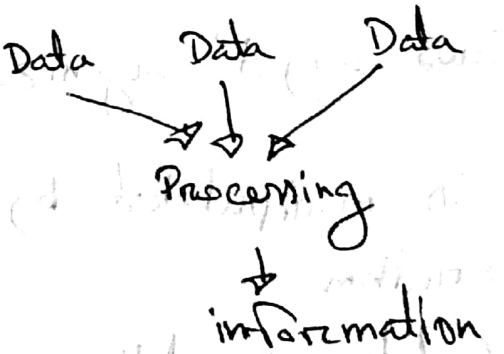
→ Values or set of values.

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

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

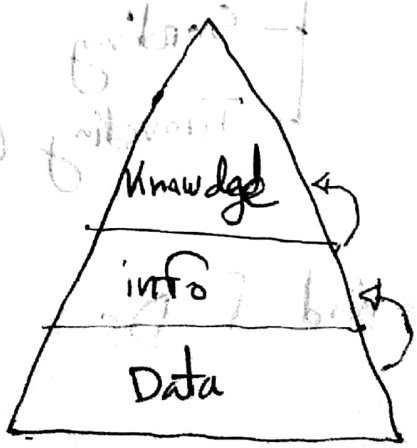
→ Data plural form 'Detam'.

✻ Information :



অনেক গুলো Data সমষ্টি থাকলে info meaningful হবে।

✻ Knowledge



vol বেশি info বেশি।

info process cycle

input → Process → output

↑↓

Storage (থাকতে পারে নাও 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
  - Traveling (visiting on elements)

- Array is a kind of DS.

ADT

group of 'DS, Algos 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.		

◦ DS ও Data type ২ types আছে → 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.
(যেকোন DS এ যেকোনো Merge করা যায়।)

✱ Advantages of ADT :

◦ Some advantages over the conventional data structure.

◦ ADT is reusable and robust and is based on OOP and SE.

◦ Can be reused at several places.

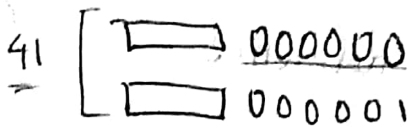
◦ Encapsulation ensures the data can not be corrupted.

27 Oct 21

Class-2

Pointer

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



ex-

```
int a = 41;
```

✓ & (ampersion) যদি variable আগে দিই তাহলে memory address দেখায়।

ex-

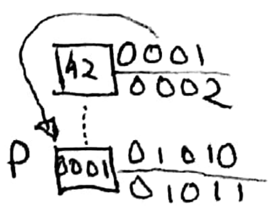
```
Printf(a) => 41;
Printf(&a) => 000001.
```

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

ex-

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

[Nb. *P দিলে 0001 [এর ভেতরের মান?] 0002 [আসে?]।]



✓ 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 হবে তাহলে।

[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।

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; }
```

No array দিয়ে কাজ করলে '&' লাগে না pointer এ।

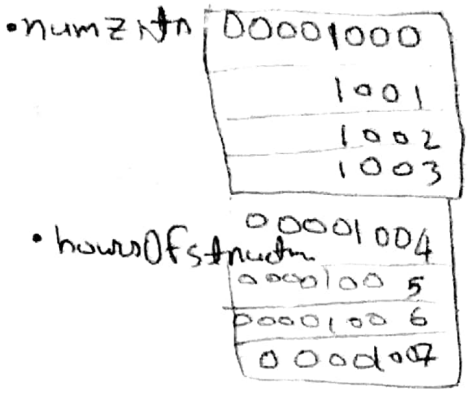
Arrays, Address & Pointers (rec)

```
{ int nums[3]={1,2,3};
✓ cout << nums;     // Prints 0100 1 (example)   ▷ অথবা nums[0] (index দিয়ে।)
                                                  0 তম index.
       /&nums;
✓ int *ptr = nums;  // pointer to array.
                    ▷ [0100 1 দিবে]
✓ cout << *ptr;  [1st address আছে এবং 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 numberOfs;
    int hoursOfStar;
};
int main(){
    Nerd carey;
    Nerd * ptr;

    ptr = &carey;

    (*ptr).numberOfs = 140;
    (ptr->hoursOfStar=42;
}
```



Nerd is structure type variable carry is created.

✓ malloc

✓ Dynamic → memory

memory.

✓ static → memory

Dynamic Variable:

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

ex- Such as, a[5] =  5টি কোন element হতে পারে, যদি 5টি থেকে বাড়ানো হয়?

- এজন্য malloc() function ব্যবহার হয়,

/calloc().

malloc():

(SS → CSE 207)

ex=>

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

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

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

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

অর্থাৎ ঐ adress storage থাকতো যত সেই used adress free হয়ে যায় এবং অন্য কাজে use হবে।

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 অবস্থা যে result দিতে পারে তাই উদ্দেশ্য.

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

f(n)=2 [nb. যখনই constant value হবে তখনই O(1) হবে, a=1, b=1, c=1
∴ O(1)

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 (assignment), i<n, i++) {
1 , n+1 , n
a = a + i — n
}

∴ f(n) = n + n + 1

= 2n [1 constant value ignore]

= n

= [2 "]

// for loop কত time চলে তা।

✓ 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

✓ for (i=0; i<n; i++) { — n+1

for (j=0; j<n; j++) { — n × (n+1)

a[i][j]=0; — n × n

}

}

$f(n) = 2n^2 + 2n + 1 \Rightarrow 2n^2 + 2n$

(rec)

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

$= 2n^2$ | $= 2n(n+1)$

$= n^2$ | $= 2n^2 = n^2$.

✓ for (int i=0; i<n; i+=2)
    sum++;

i++
i+=2 অর্থাৎ 2গুন বাড়ছে
তাই n/2 বার হবে।

⇒ $f(n) = n/2$

$= n$

$= O(n)$

✓ for (int i=0; i<n; i++) — n
    for (int j=0; j<n*n; j++) — n*n
        sum++

$= n*(n*n)$

⇒ $O(n^3)$

~~nested~~ nested Condition same n.

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

⇒ a+b.

✓ `k = n;`

```
while (k > 1) {
    sum++;
    k = k/2; }
```

=>

$\log_2(n)$.

Assume,	
~~k = 16~~	k > 1
k = 15	
k = 8	✓
k = 4	✓
k = 2	✓
~~k = 1~~	✗

ans: $\log_2 16$ ~~= 3~~

④ বার।

✓ `for (int i=0; i<n; i++)` — n

```
{ int k = n;
  while (k > 1) {
      sum++
      k = k/3
  }
```

=> $n * \log_3(n)$

multiply এর

→ সর্বোচ্চ power polynomial

(n) এরটি logarithmic ($\log_3(n)$), তাই দ্রুত।

✓ `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;
```

=> $n^2 + n^3$

-> n^3 → height degree

* nested loop (*) → No. condition matter करते, (rec)

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

⇒ $O(n^2)$

```
for(i=0; i< q*q; i++)        q^2
    for (j=0; j<i; j++)      q^2 बार Run करेगा,
        sum++;
```

⇒ $\therefore q^4$

lower order omit ⇒ $O(c^2 + c)$
$= O(c^2)$

Book example Practice

ex →

① for (i=0; i<p; i++) {
for (j=0; j<f; j++) {
} }

⇒ p * f [nested]

② for (i=0; i<c; i++) {
for (j=0; j<c; j++) {
} }

for (k=0; k<e; k++) { }

$= O(c^2 + e)$.

③ for (int i=0; i<n; i++) {
if (i == n/2) {
for (int k=0; k<q; k++)

}

Class - 6

Linked Lists

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

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

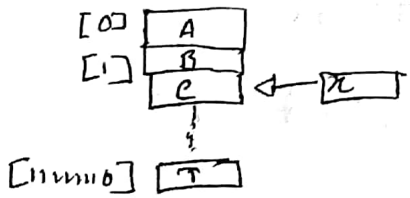
⊙ reserve enough slots....

⊙ Wastage of memory.....

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

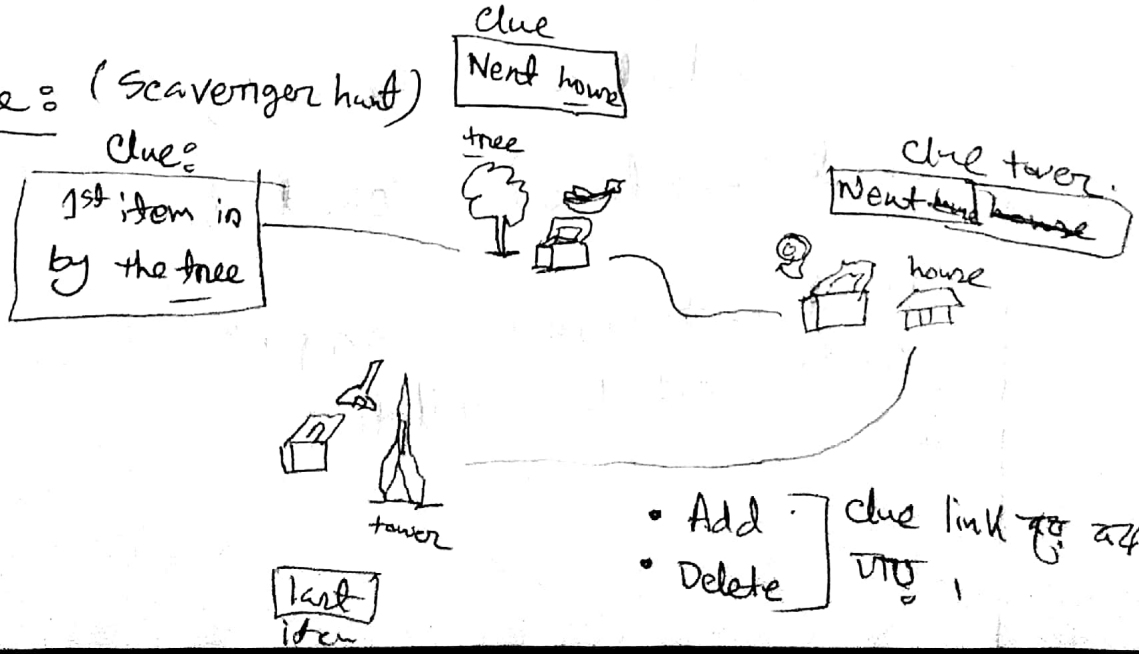
⊙ Values move করালে হয়- Array size বড় হলে problem হবে।

ex-



C এর জায়গায় R বসাতে (insert করতে গেলে) time waste হবে।

example: (Scavenger hunt)



- Add
- Delete

clue link হবে বলে যাচ্ছে।

LinkList Example: 1/11/21

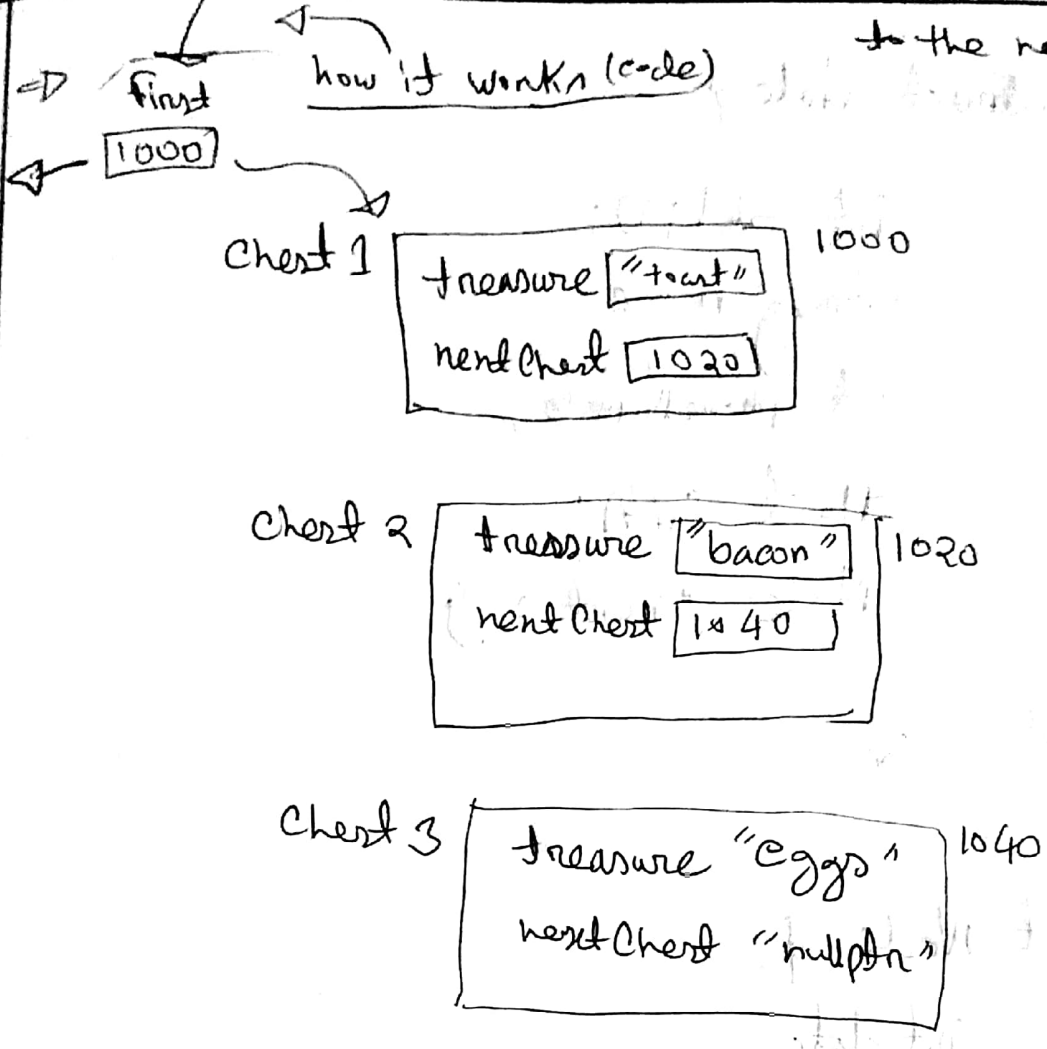
Code (Scavenger hunt):

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

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

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

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

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

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

how it works (code)



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

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

First = &Chest 1;

✗ Chest1.treasure = "tart";

✓ first = new chest;
first -> treasure = "tart";

(7) 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 (struct

second = ;

head -> data = 3;
Printf("%d");

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

3/11/21

Class-7

insert in linked list:

56 min

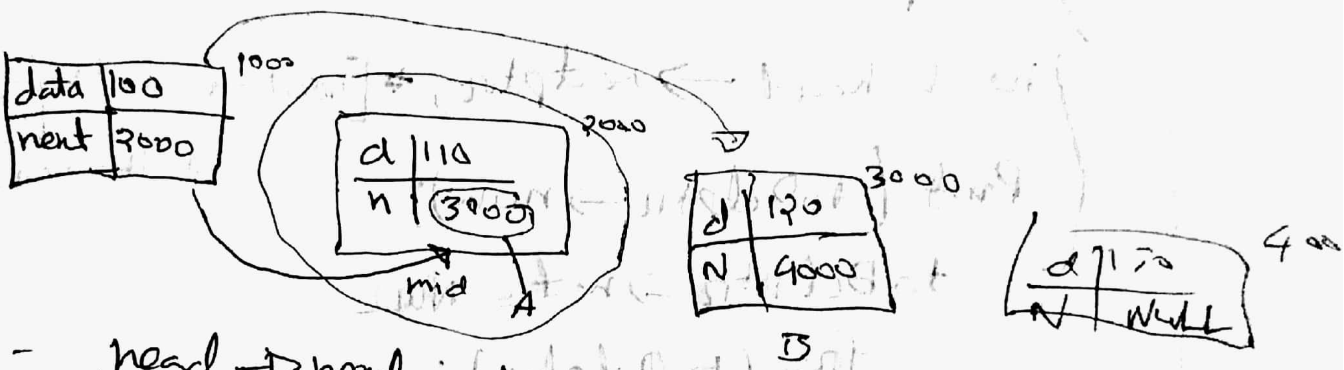
Delete in linked list:

1hr/10

/[illegible]

3 ভাবে delete হয় - head, mid, last.

(mid node delete)



head -> next = A -> Next
free(A);

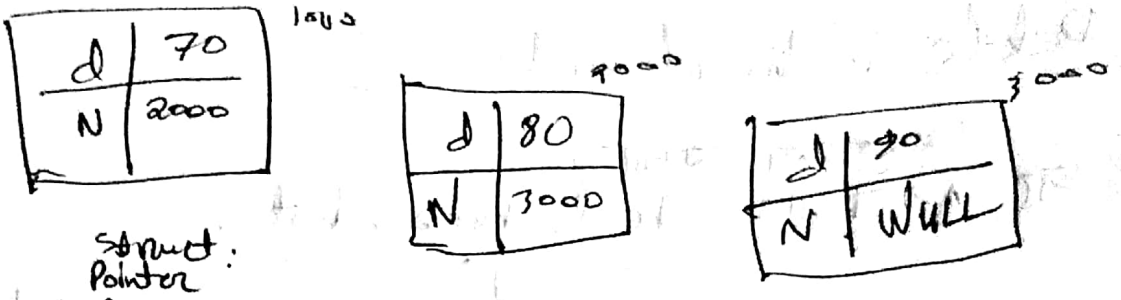
অথবা, head -> next = head -> next -> next.

[Problem] code

Rev
temp 1
temp 2
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 করে দিতে হবে,

1.5 hour.

(4-C) code.

Insert: (i) head ... Node add (NewHead)
(ii) ..., (iii) Next ... add ...

(i) =>

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

(iii) =>

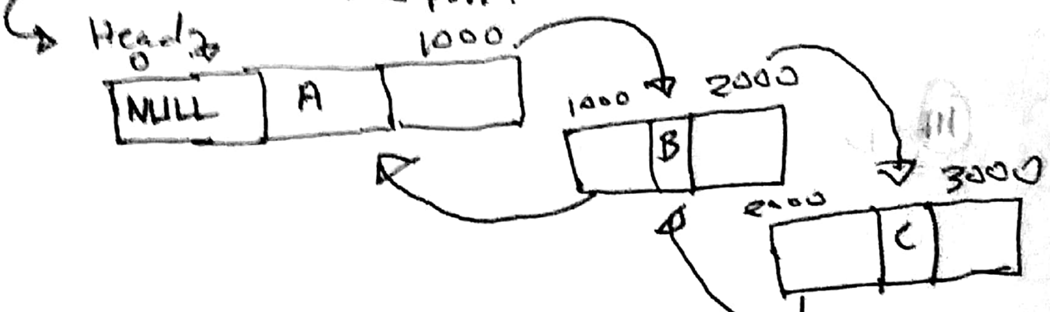
Class-9

Doubly Linked List

• Pointer দুদিকে থাকে,

১. Linked list – [data | next ptr]

doubly " – [Prev. ptr | Next | ptr]



যেমন, এ থেকে দিকে যাওয়া যায়। আবার Linked List এ Last থেকে First এ যাওয়া যায়,

Prob – ① delete – C –> Prev –> next = C –> next;
free (C) ;

Note: Linked list এ করতে হবে Loop লাগবে।
–> Prev.ptr আগে লাগবে, SO; C –> next –> Prev = C –> Prev;

(ii) insert = Last –> C ;
C –> Next = D ;
D – Prev = C ;

[Last এ D Node insert করুন,]

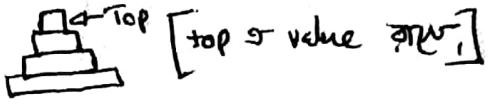
F –> Next = C ;
F –> Prev = B ;
B –> next = F ;
C –> Prev = F ;

[B এবং C এর মাঝখানে F আনতে হবে,]

Class-10

Stack / LIFO (Last in first out)

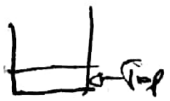
- Stack is a linear list which all additions and deletion are restricted to one end, called top.



- If you insert data series as {5, 10, 15, 20} in reversed as {20, 15, 10, 5}. // like inverse / reverse.
- Basic operation — ◦ Push (Value insert) ; ◦ Pop (Delete),
  ◦ StackTop (Top Value).

ex-

① Pop :

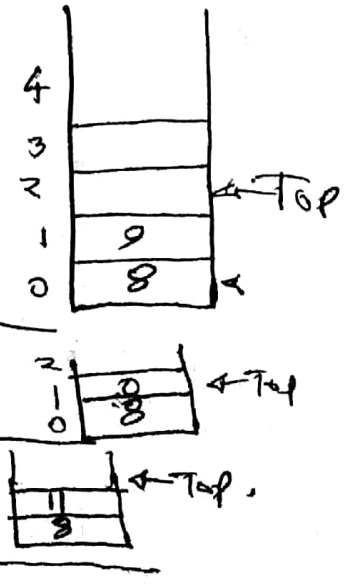


② Push : 8

③ Push : 9

④ Pop

⑤ Push : 11



Common use of stacks -

- Storing undo items for your word processor.

  ex -> ctrl + z দিয়ে সব জিনিস আগের মতো , //typing

- Evaluating mathematical expressions -

  5 + 6*3 -> 23 // sum এর পর যদি, multiply এর আগে তা হবে নয়,

- Converting infix to postfix expression. -

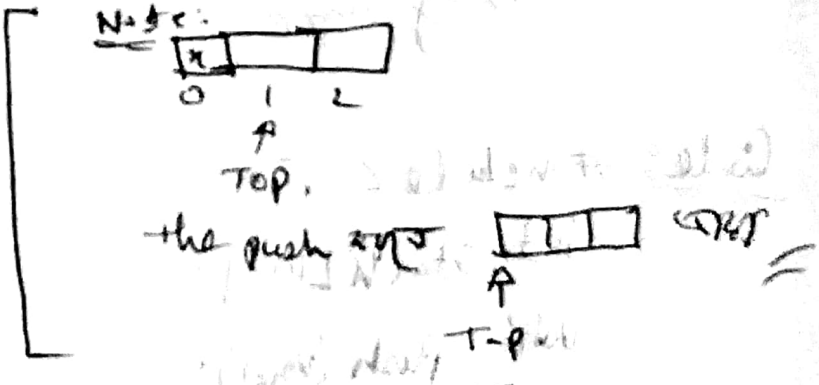
  A + B -> AB+

- Solving maze.

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 থেকে বাড়িয়ে হবে 0 হবে,
        stack [top] = x; } // যা input হবে তা
                              show করাবে।
```



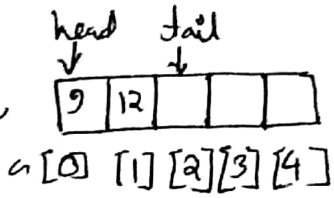

```
Void display() { if (top >= 0) {
    for(int = top ; i >= 0, i--)
        Printf("%d", stack [i]); }}
    else { empty; }
}
```

Class-11

Queue (FIFO) → First in First Out.

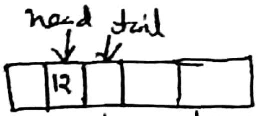
Queue	stacks	
ENQUEUE ⟷	Push	(add / insert)
DEQUEUE ⟷	Pop	(delete)

ex- int a[5] so,



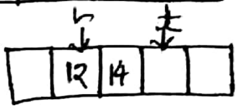
(i)

then,



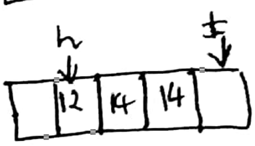
(ii) DeQUEUE (9)

then,



(iii) EnQUEUE (14)

then,

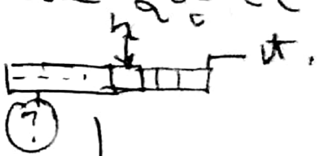


(iv) " (14)

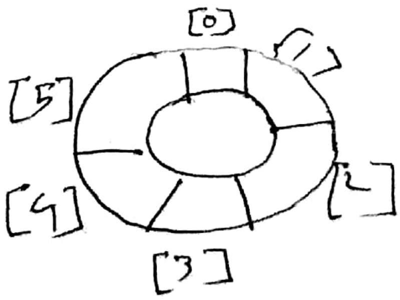
↳ আর max ১টা value রাখা যাবে,

- আবার- এ মনে head ওপর থেকে Queue শুরু হবে,

✓ So, আগের Part গুলো waste যাবে ? ⇒ (head)



⇒ Circular use করে করা যাবে,



✓ Code: a[0] - - - a[7]   ⊕ mode → array size.

a[0]   a[7]

Scanned with CamScanner

Search & Sort

Search: (linear search) — Sorted থাকে না,

here,

(*) 12 | 12 | 18 | 20 | 25 ← 20 find?

⇒ for (i=0, i<n, i++) {

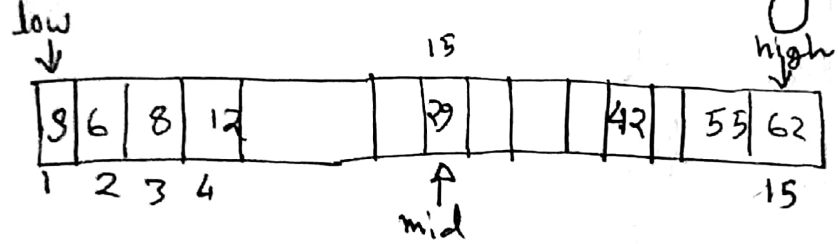
    if a[i] == 20

        Printf ("%d done", i); }

— Complixity $O(n)$.

(binary search) — lists are sorted.

∘ why binary? ⇒ Just for reducing time complixity!

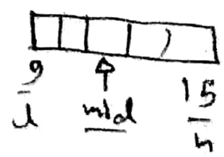


here, low = 3 ; Position = 1

high = 62 ; ; Pos = 15

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

note: find 42? → So; mid value এর 42 compare করে list এ half side নিয়ে কাজ করতে হবে, then



নিতে হবে।

— complixity $\log_2(n)$.

Code: BinSearch (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; };
```

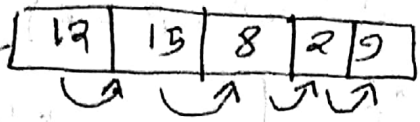
Sorting :- types -

13 class

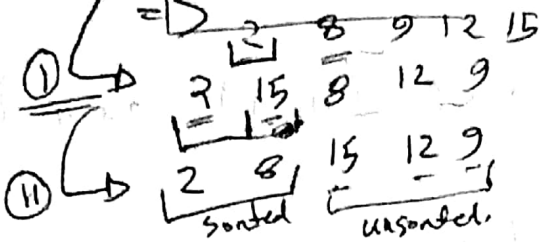
Rec (PrevSem-1)

(I) Selection Sort.
(II) Bubble Sort.
(III) Insertion Sort.
(IV) Merge Sort.
(V) Quick Sort.

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



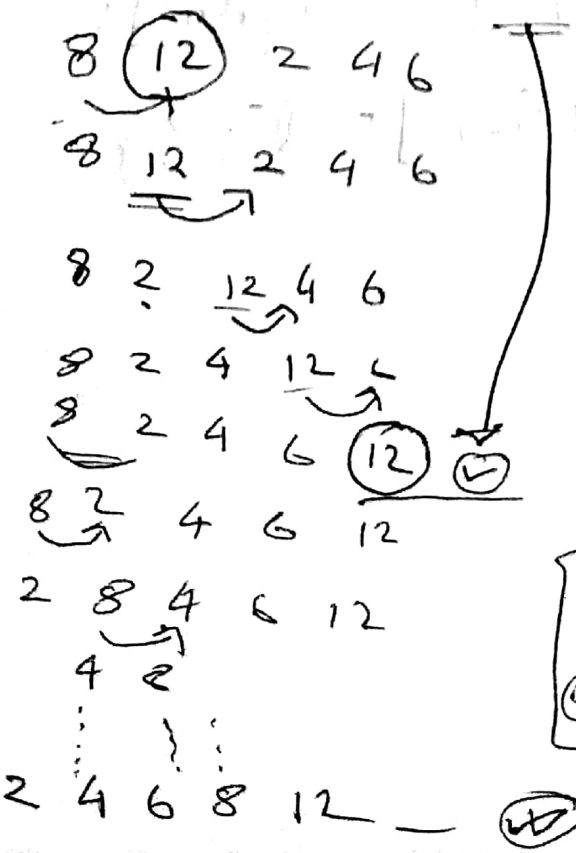
- temp.
- max_min ধরে.
- loop -
  1st (1 - n)
  2nd (2 - n)
  ⋮
  ~~n-1~~ t



$O(n^2)$ = complexity.

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

(II) Bubble Sort: (যে সংখ্যা এগুলোর বড় তা উপরে নিয়ে আসা- হয়; like as bubble উঠে.)



✓ প্রথম দুটো check করবে যদি হ্যাঁ হয় তাহলে swap. নিবে,
✓ selection sorting এর প্রথমটি ধরে 1st থেকে নেবে।

Note: এগুলোর বড় সংখ্যাগুলো শেষ list এ আসবে।

Selection Sort: ~~If Part~~ E

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

Bubble: — Complexity $O(n^2)$

```
//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 পাবো।

পুরো গুলো value check

✓ **Insertion sort:** ✓ Sorted list থাকে আগে থেকেই ধরি।

✓ then new value assign করে sorting শুরু করে।

12	20	50	40

(40) assign করেছি।

✓ 3 38 5 12

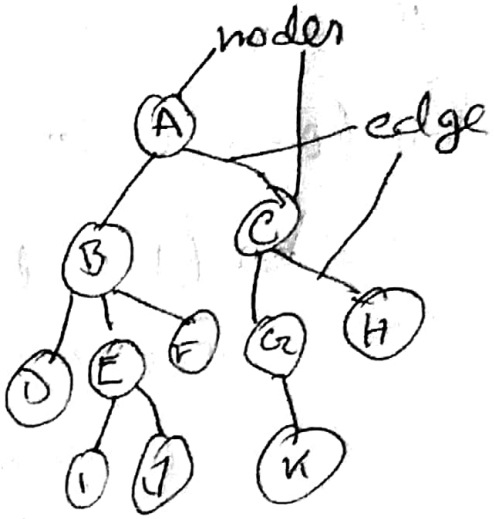
3 38 (sorted হয়েছে) 5 12

3 5 38 12 (sorted)

3 5 ~~10~~ 12 38 sorted.

BSC → site

Tree



- Parent যুক্ত গাছ।
- individual elements are node.
- if a tree of 'n' number 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 siblings, same as, (D, E, F), (G, H), (I, J) etc.
- who'n (node) doesn't have child is called leaf node. / terminal / external.
- যার child আছে internal node বা / non-leaf.

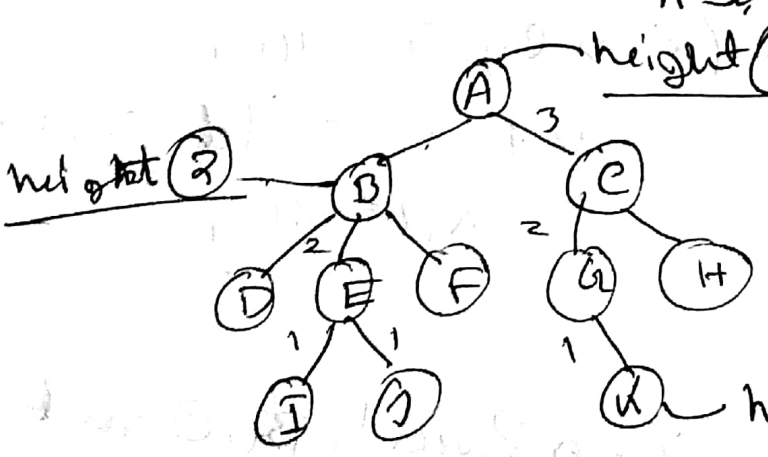
✓ Degree - total numbr of child

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

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

✓ level: (A) - lvl 0
(B) (C) - lvl 1

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



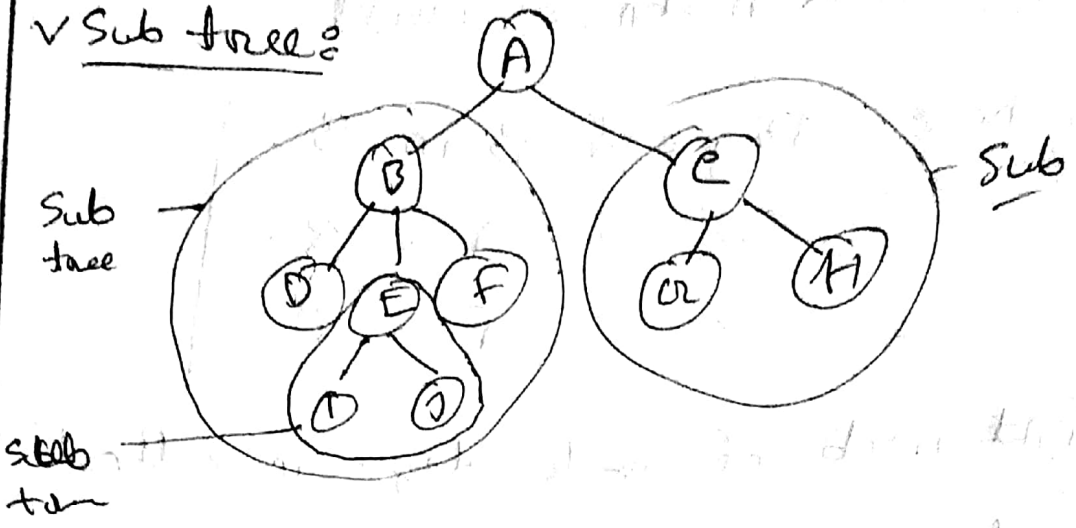
height (3) [leaf → Root]
height (2)
height in 0.

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

✓ Path: (A) থেকে node থেকে node যে

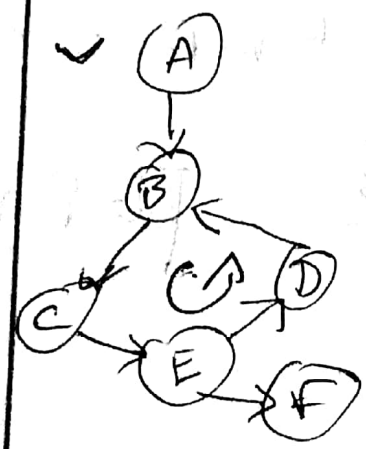
(A) & (J) ⇒ A - B - E - J
Path

✓ Sub tree:



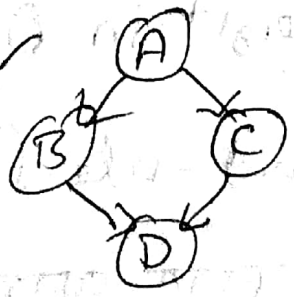
Ques:

✓ A → B ✓ ✗ → cycle হয় তাই না।



✓

✗ → cycle আছে।
→ E এর child B হলে আছে না।



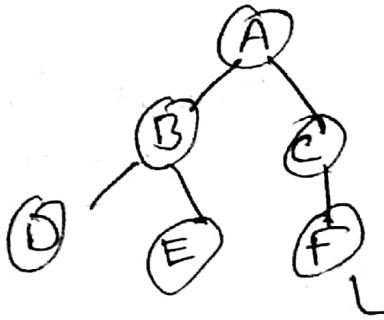
✓

✗ → Parent B C not possible.

✓ Structure for Tree Representation.

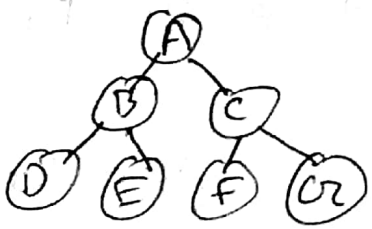
Binary Tree

Condition: প্রতি node এর দুটি child (max).

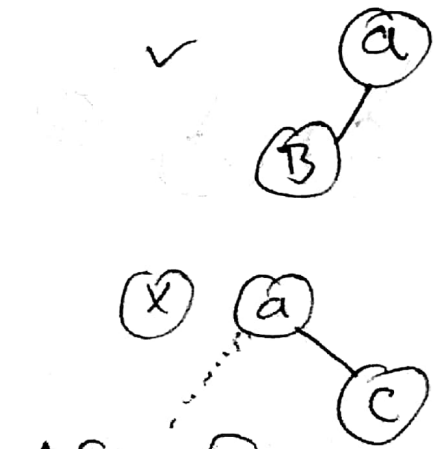


- it's a bin tree.
  ↳ it's not a full bin tree..

✓ Full Bin Tree: (child max বা কম হবে,)



✓ Complete bin tree: Condition: প্রতি node এর left side থেকে mendatory.



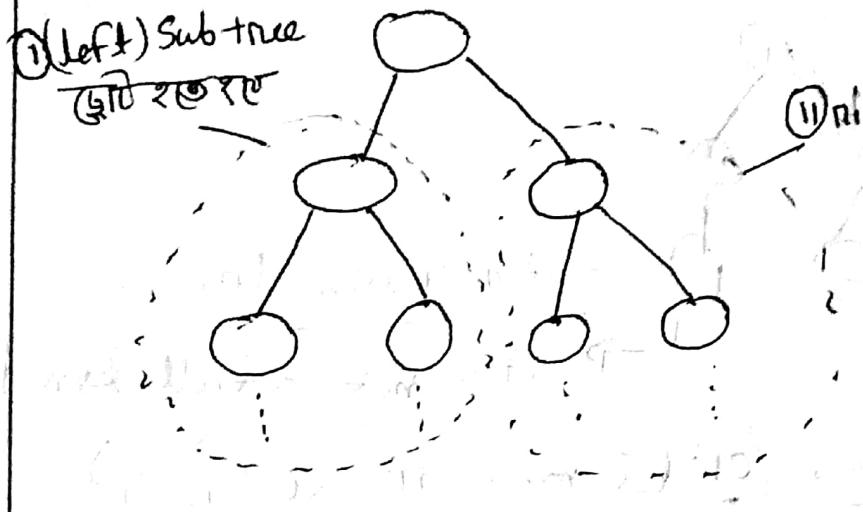
left নাই
so, হবে না.

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

6/12/21

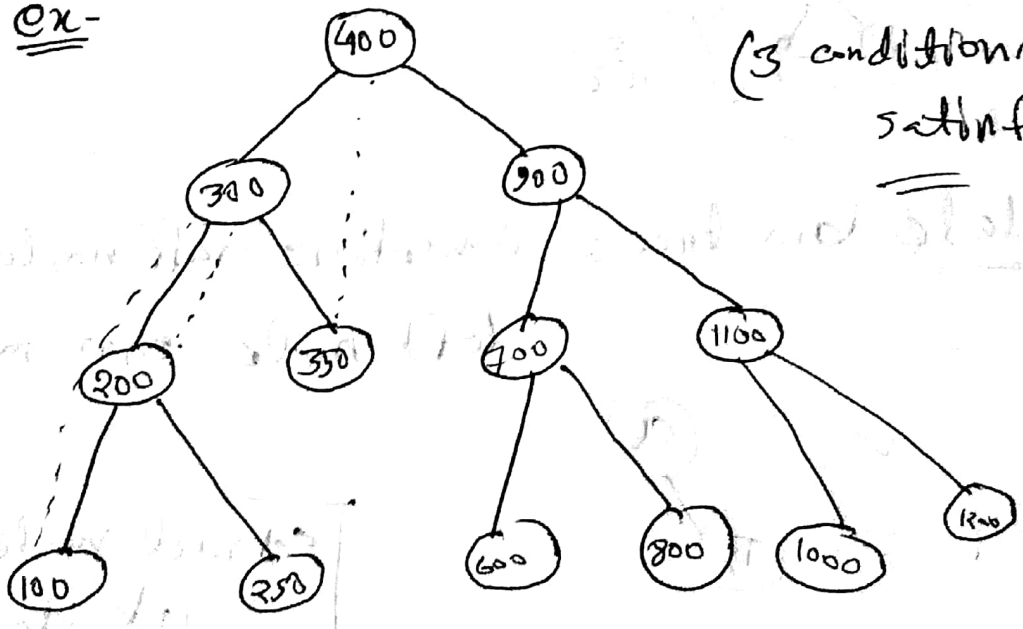
Binary Search Tree : (BST)

[difference - values থাকবে]



(iii) কোনো duplicate node থাকতে পারে না।

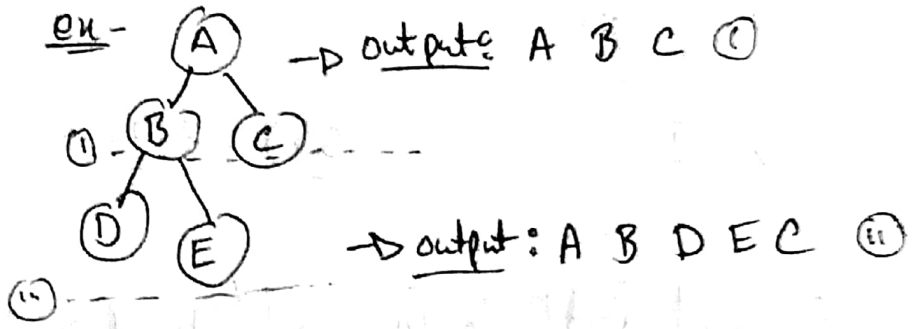
Ex-



(3 conditions satisfied)

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)

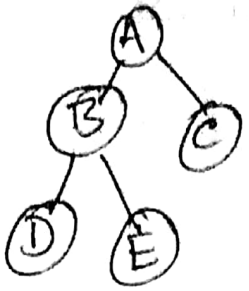
Code:

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

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)

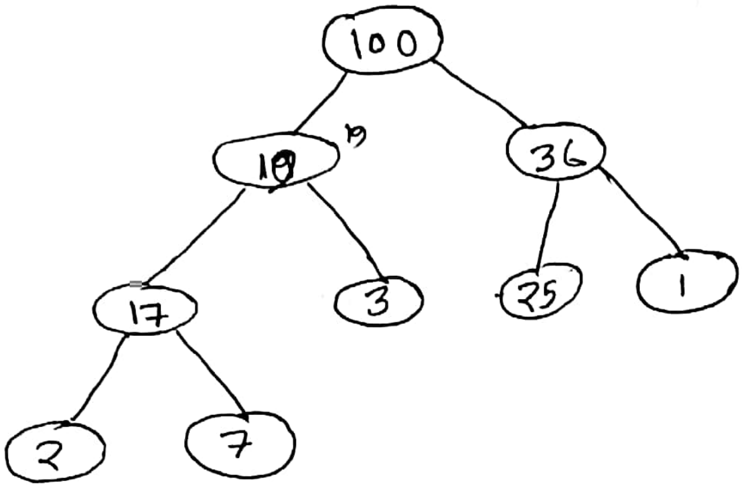
Print ("%d ->", root -> item);

InOrder (root -> Right) }

Class - 19

Heap:

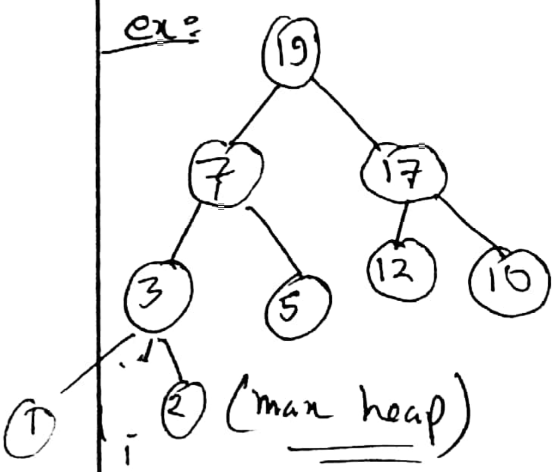
Condition: (i) Bintree হতে হবে।
(ii) Complete Bintree,



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

why 0 ? - 1

ex:



max heap creation: tip: Serially left - Right maintain করে করে।

int heap = 0

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

heap[1] = 19
heap[2] = 7
heap[3] = 17

যেটা বের করবো,

Left এর জন্য logic -> heap[x * 2]

Right " " logic -> heap[2 * 2 + 1]

@x =>

Ideen

(4) যদি child -> n = 4;

left child = (n * 2)

Right child = (n * 2 + 1)

child/ (node এর left & Right বের করা)

@x =>

index / 2

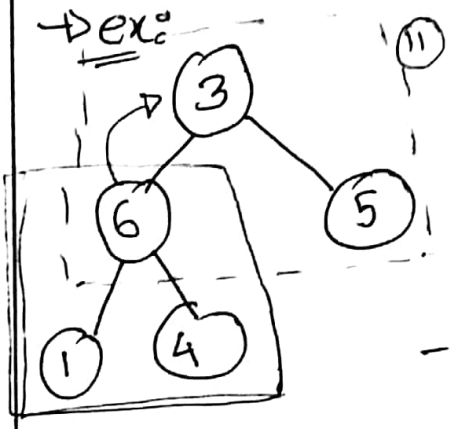
Parent/ (Root বের করা)

Code:

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

- How to Convert a tree to Max heap?

→ ex:



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

- So, এটিকে heapify করতে হবে - - -

(1) • bottom node কে ধরে করতে হবে,
(replace) ✓ সবচেয়ে বড় Val Root এ আসবে,
(1), (11) - - -

(11) Top nod কে ধরে নিচে নেমে হবে আসবে,
যাতে Parent গুলো বড় থাকে,