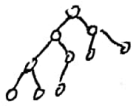
search engine.
enerally in
e do pre
lsing tries,
itself while
d the ~~tree~~
is termed as

UNIT 5

Heap

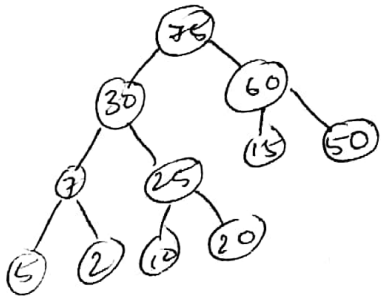
Heap is a Binary tree such that the keys stored in each node should have the following properties.

i) <u>structural property</u> -
The tree should be complete binary tree.
ie., all levels apart from last level should be completely filled and the last level should be filled from left to right.



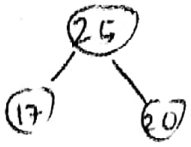
ii) <u>Parent dominant property</u>:

Parent node should always have higher priority when compared to its children

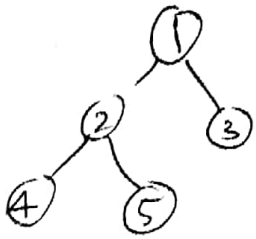


There are 2 types of heaps, namely,

i) Max heap - where the root node should have max element



ii) min heap - the root node should have the min element.



There are to approaches to construct a heap.

i) Bottom up approach

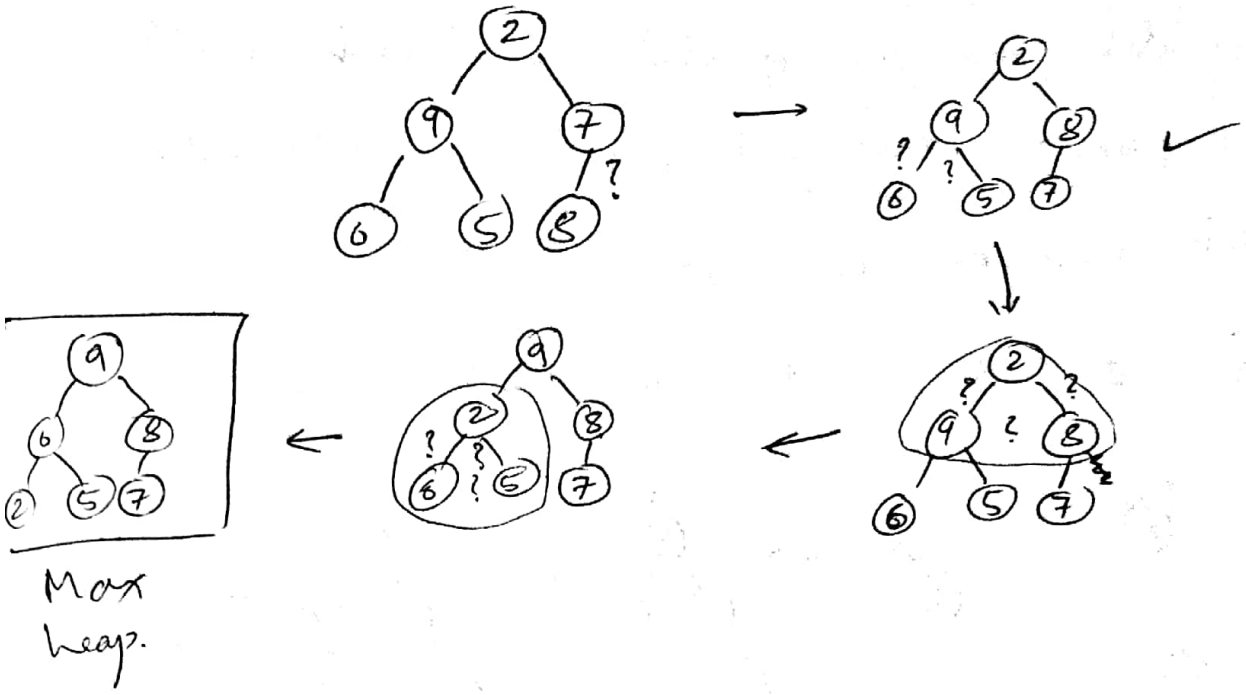
ii) Top down approach.

Bottom Up

Here, for the given set of elements we construct a BT first and then we transform the binary tree to a heap.

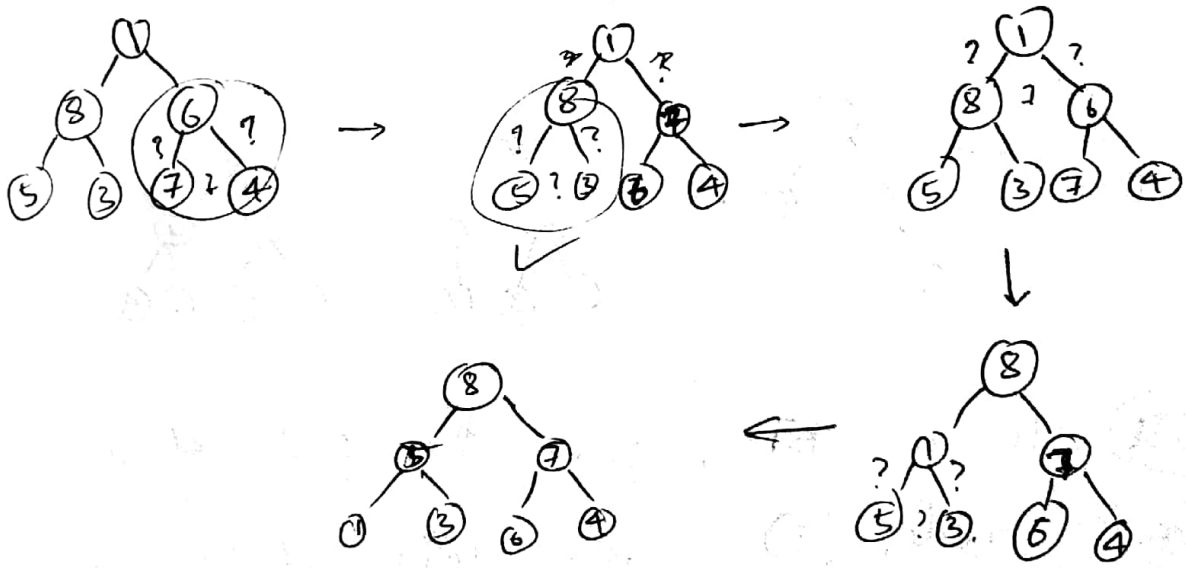
2 9 7 6 5 8

Build max heap.



Max heap.

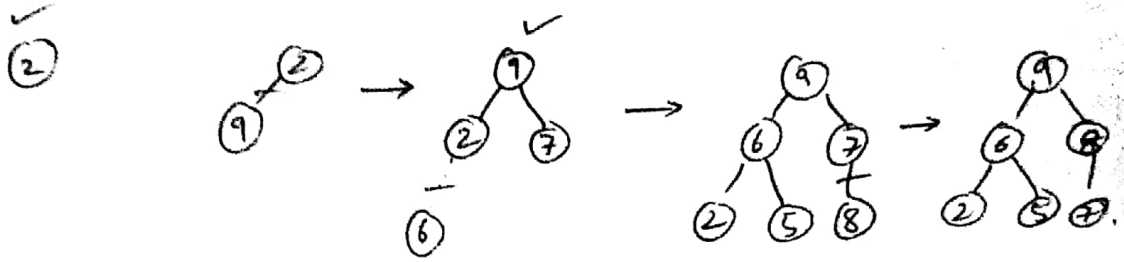
(3) 1 8 6 5 3 7 4.



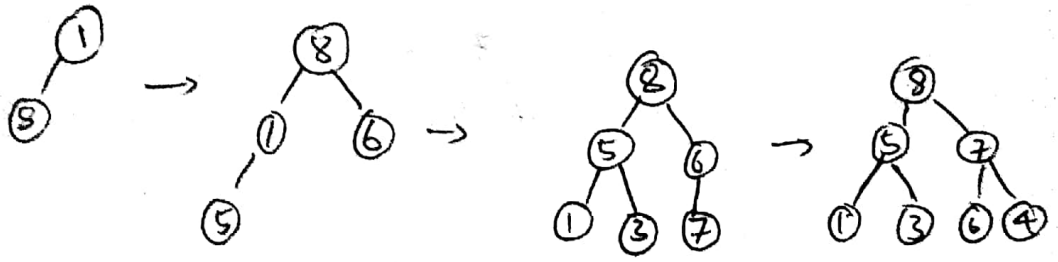
Top Down Approach

In this approach, while creating the BT using repeated insertion operation, we check whether the parent dominant propert is achieved after each insertion or not.

(a) 2 9 7 6 5 8



(b) 1 8 6 5 3 7 4.



NOTE: In a BT, given the posn of a parent node, (i), it's left child posn is denoted with (2i) and right ~~child~~ child is denoted with (2i+1).

Given the position of a child node, (c) it's parent node posn is denoted by

$$P = \lceil (C-1)/2 \rceil$$

Procedure or Algo

to construct a max heap.

O(n) to build heap with n elements.

```
Buildmaxheap (A[ ], n)
// builds max heap for the given set of n elements.
// Input : A set of n elements stored in A.
// Output : An ordered set of elements which
// represents max heap.

for i ← ⌊n/2⌋ to 1
    heapify (A, i, n)


heapify (A, i, n)
//
//
//
left ← 2i
right ← 2i + 1
if (left <= n and A[left] > A[i])
    largest ← left
else
    largest ← i
```

```
if ( right <= n  &&  A[right] > A[largest])
        largest = right.
if (largest != i)
{
        swap (a[i] , a[largest])
        heapify [A, largest , n]
```

~~Procedure to construct delete from max heap.~~

Algorithm to extract max element. (deletion)

```
                        array, rep heap
Algorithm Extract max ( A , n )
// Retrieves max element from the heap.

if n is 0
        heap is empty
else                                    index is from 1 to n
        max = A[1];
        A[1] = A[n];
        n = n-1;
        heapify (A, 1, n);
```

O(1) to extract max element

Program 10

Implement priority Queue using heap.

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

int n;

void swap (int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

void heapify (int a[10], int i)
{
    int left, right, largest;
    left = 2*i;
    right = 2*i+1;
    if (left <= n && a[left] > a[i])
        largest=left;
    else
        largest = i;
    if (right <= n && a[right] > a[largest])
        largest=right;
    if(largest!=i)
    {
        swap (&a[i], &a[largest]);
        heapify(a, largest);
    }
```

```
void buildheap (int a[10])
{
   int i;
   for(i= n/2 ; i >= 1 ; i--)
        heapify(a , i);
}

int extract max (int a [10])
{
    int max;
    if (n==0)
    {
       pf(" Heap is empty ");
       return -1 ;
    }
    else
    {
        max= a[1];
        a[1]= a[n];
        n=n-1;
        heapify (a, 1);
    } return max;
    }
}

int main ()
{
  int a[10]; i , ch
  while(1)
  {
    pf(" Enter choice\n1. create heap\n2. delete\n3. exit\n:");
    sf("%d", &ch);
    switch(ch);
    {
```

```
case 1: pf(" Enter value for n");
        sf("%d", &n);
        pf("Reade elements\n");
        for(i=1 ; i<=n ; i++)
            scanf("%d ", &a[i]);
        buildheap(a);
        printf("Elements after heap are \n");
        for (i=1; i<=n; i++)
            pf("%d\t", a[i]);
        break;
case 2: pf("\n The element retrieved is %d",
                extract max(a));
        pf("Element afte deletion\n");
        for(i=1; i<=n;i++)
            pf("%d\t", a[i]);
        break;

default: exit(0);

}
}
return 0;
```