DATA STRUCTURE

- My Introduction
- Reality of the course
- Time we will spend
- Theory + Code ratio
- My strategy
- Your devotion
- Book followed - Data structures by Seymour Lipschutz (Schaum Series)

LET’S START!

Blog-
www.delvingintothelight.wordpress.com

HEAP TREE

1. Heaps are usually implemented in an array
2. A heap can be classified further as either a "max heap" or a "min heap"
3. In a max heap, the keys of parent nodes are always greater than or equal to those of the children and the highest key is in the root node
4. In a min heap, the keys of parent nodes are less than or equal to those of the children and the lowest key is in the root node

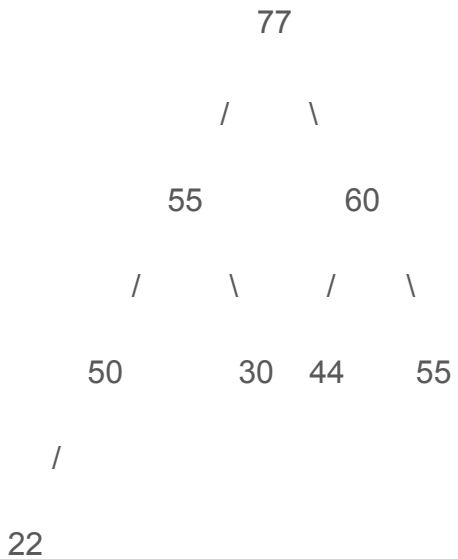
In other words

A Binary Heap is a Binary Tree with following properties.

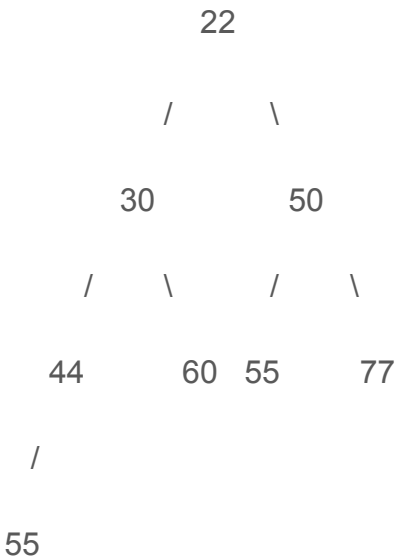
1) It’s a complete tree (All levels are completely filled except possibly the last level and the last level has all keys as left as possible). This property of Binary Heap makes them suitable to be stored in an array.

2) A Binary Heap is either Min Heap or Max Heap.

For the array { 44,30,50,22,60,55,77,55 }



MAX HEAP



MIN HEAP

Algorithm for insertion in Heap tree

n=no. Of elements stored in array

1.set n=n+1, ptr=n

2.[Find location to insert Item] Repeat steps 3 to 6 while ptr>1

3. Set par=ptr/2 [Location of parent node]

4. If Item<=Tree[par] then Set Tree[ptr]=item and return

5.Set Tree[ptr]=Tree[par] [Moves node down]

6.Set ptr=par [update ptr]

[End of step 2 loop]

7. Assign item as the root of tree

   Set Tree[1]=item

8.Return.

Algorithm for deletion in Heap tree

1.Set Item=Tree[1] [Removes root of Tree]

2.Set last=Tree[n] and n=n-1 [Removes last node of tree]

3.Set ptr=1, left=2, right=3

4.Repeat step 5 to 7 while right <=n

5.if last>=Tree[left] and last>= Tree[right] then : Set Tree[ptr]=last return.

6. If Tree[right]<=Tree[left] then Set Tree[ptr]=Tree[left] and ptr=left.

   Else Set Tree[ptr]=Tree[right] and ptr=right

7.Set left=2*ptr and right =left+1

8.If left=n and if last < Tree[left] then Set tree[ptr]=tree[left] and ptr=left

9. Set Tree[ptr]=last;

10. Return

Applications of Heaps:

1) Heap Sort: Heap Sort uses Binary Heap to sort an array in O(nLogn) time.

2) Priority Queue: Priority queues can be efficiently implemented using Binary Heap because it supports insert(), delete() and extractmax(), decreaseKey() operations in O(logn) time.

3) Graph Algorithms: The priority queues are especially used in Graph Algorithms like Dijkstra's Shortest Path and Prim's Minimum Spanning Tree.

4) Many problems can be efficiently solved using Heaps.

a) K'th Largest Element in an array.

b) Sort an almost sorted array

c) Merge K Sorted Arrays.

```
 void insert()
{

    int ch=1;
        while(ch==1){
    printf("Enter data: ");
    fflush(stdin);
        scanf("%d", &data);
        n++;
    heap_insert();

   printf("To continue press 1 else 0");
   scanf("%d",&ch);

        }
}
```

```
void heap_insert()
{
        int par,ptr=n;
        while(ptr>1)
        {
                par=ptr/2;
                if(data<=tree[par])
                {
                        tree[ptr]=data;
                        return;
                }
                tree[ptr]=tree[par];
                ptr=par;
        }
        tree[1]=data;
        return;
}
```

```
void delete_heap()
{
        int last,ptr,left,right;
        data=tree[1];
        last=tree[n];
        n--;
        ptr=1,left=2,right=3;
        while(right<=n)
        {
                if(last >= tree[right]&& last >= tree[left])
                {
                        tree[ptr]=last;
                        return;
                }
                 if(tree[right]<=tree[left])
                 {
                        tree[ptr]=tree[left];
                        ptr=left;
                 }
                else
                        {
                                tree[ptr]=tree[right];
                                ptr=right;
                        }
                        left=2*ptr;
                        right=left+1;
                }
                if(left==n && last < tree[left])
                {
                        tree[ptr]=tree[left];
                        ptr=left;
                }
                tree[ptr]=last;
                return;
```