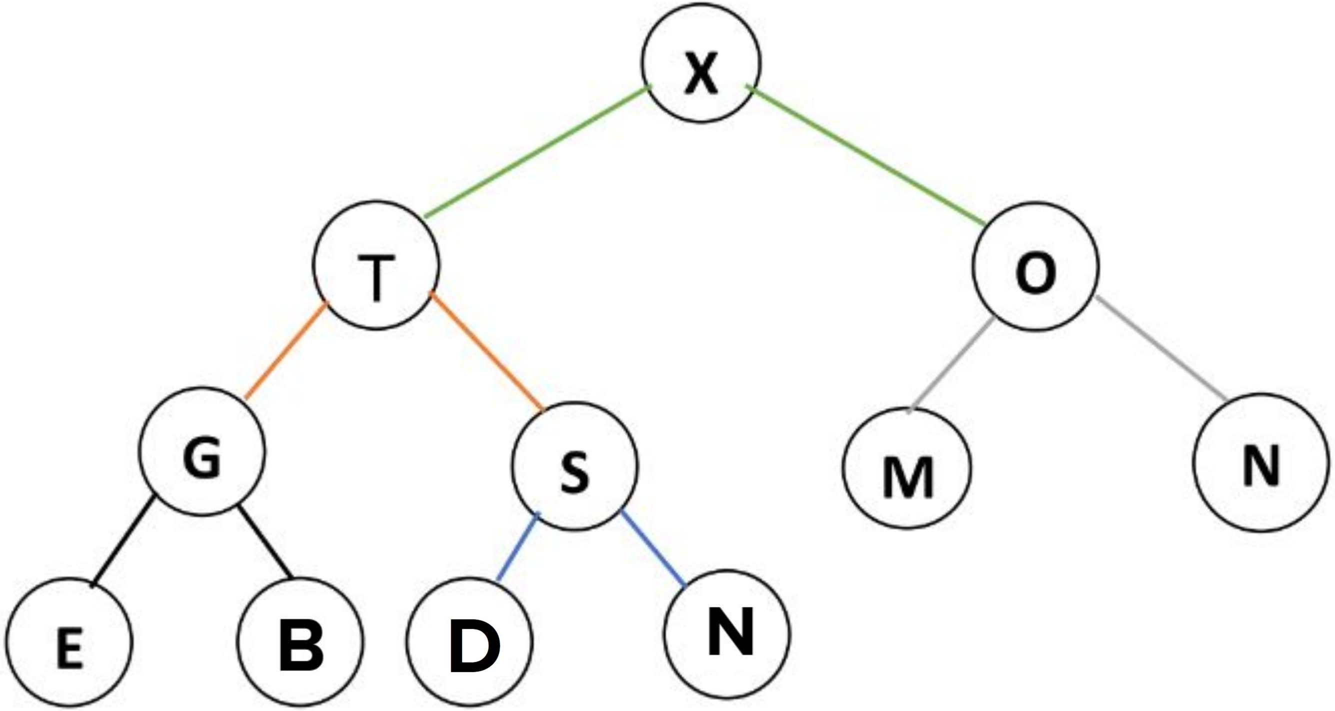
Data Structures

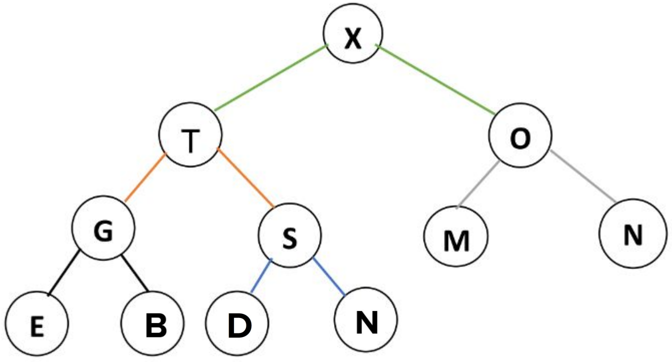
Lecture 14
Heaps

Credit: Prantik Paul, Lecturer, BRAC University

Heap - Special Binary Tree



Heap - Properties



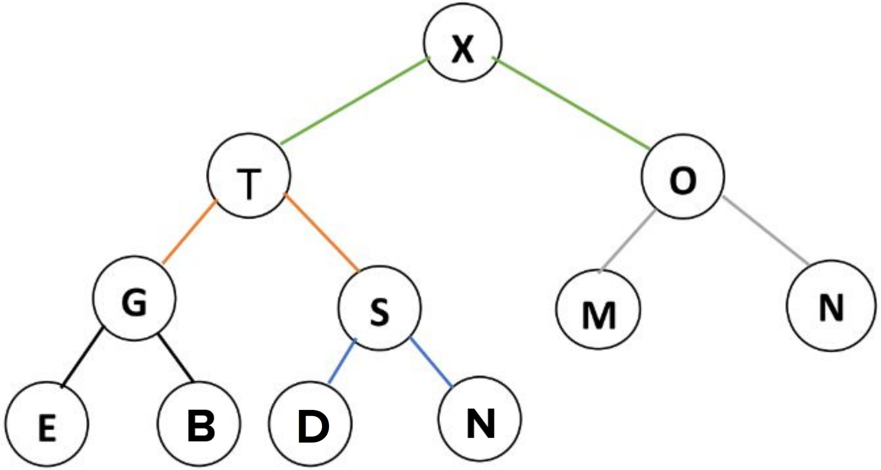
- → Complete

- → Satisfy Heap Property

- → Value of Parent >= Value of Children MaxHeap

- → Value of Parent <= Value of Children MinHeap

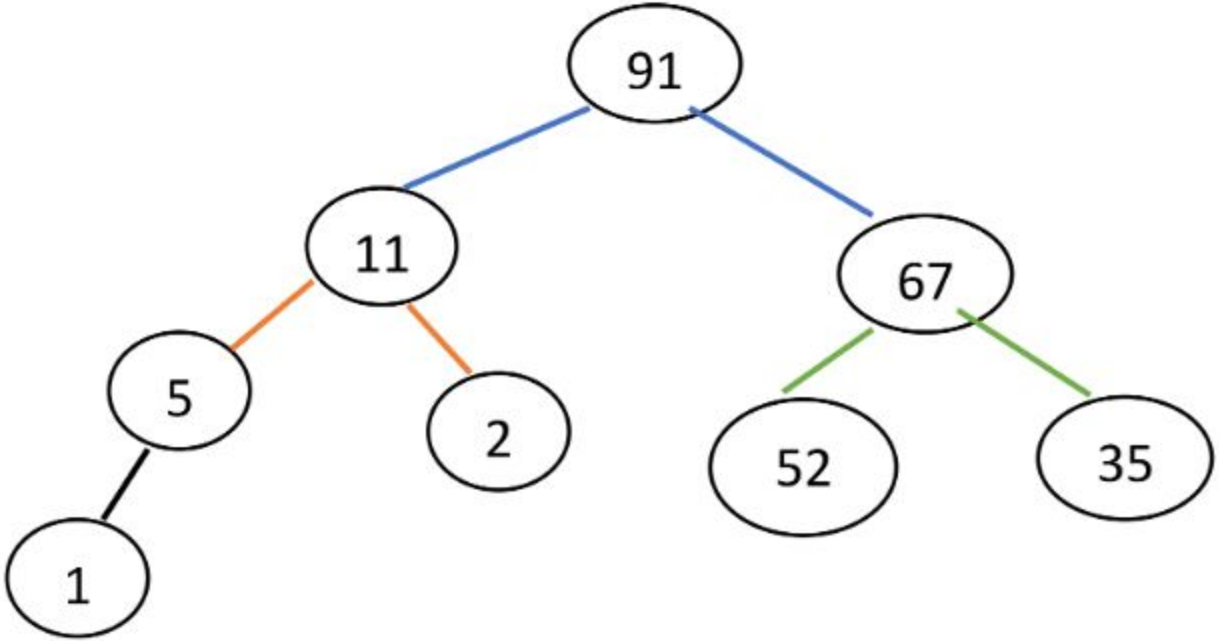
Heap - Representation



1	2	3	4	5	6	7	8	9	10	11
X	T	O	G	S	M	N	E	**B**	**D**	**N**

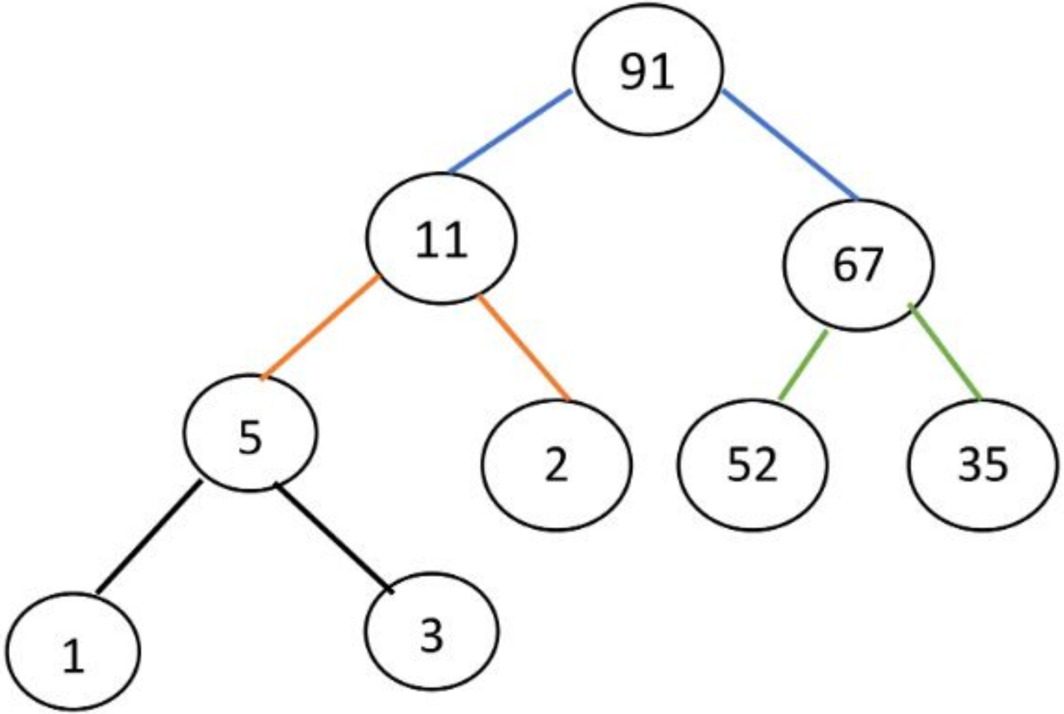
Heap - Insert

Insert 3



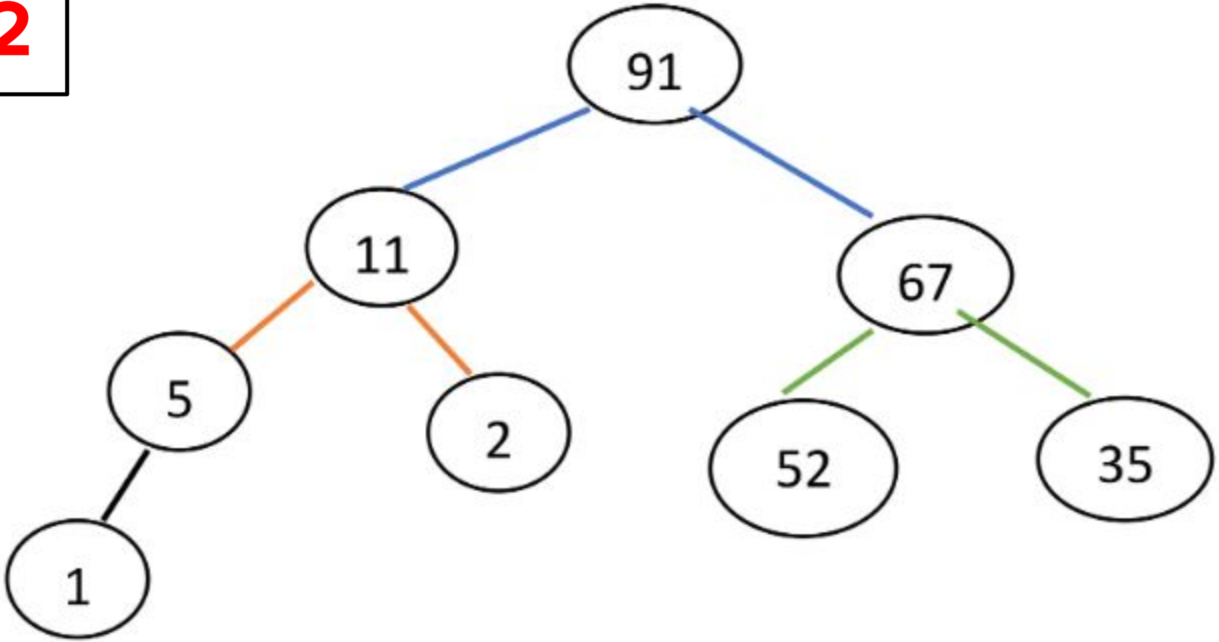
Heap - Insert

Insert 3



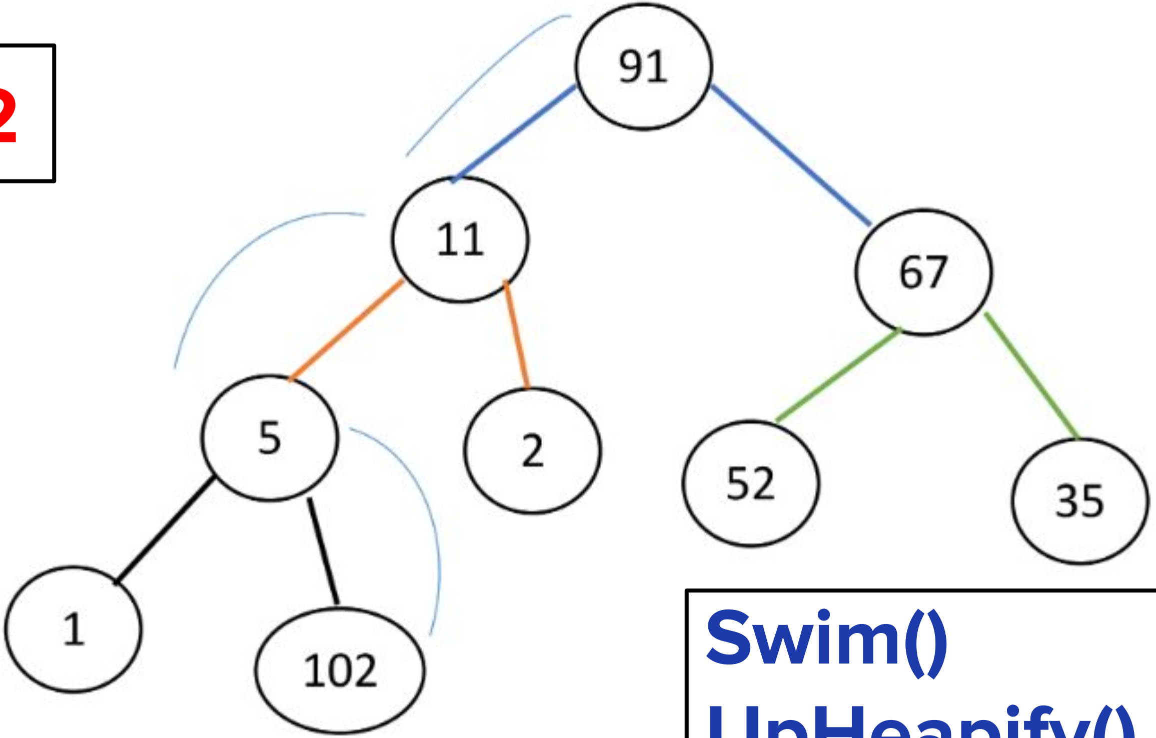
Heap - Insert

Insert 102



Heap - Insert

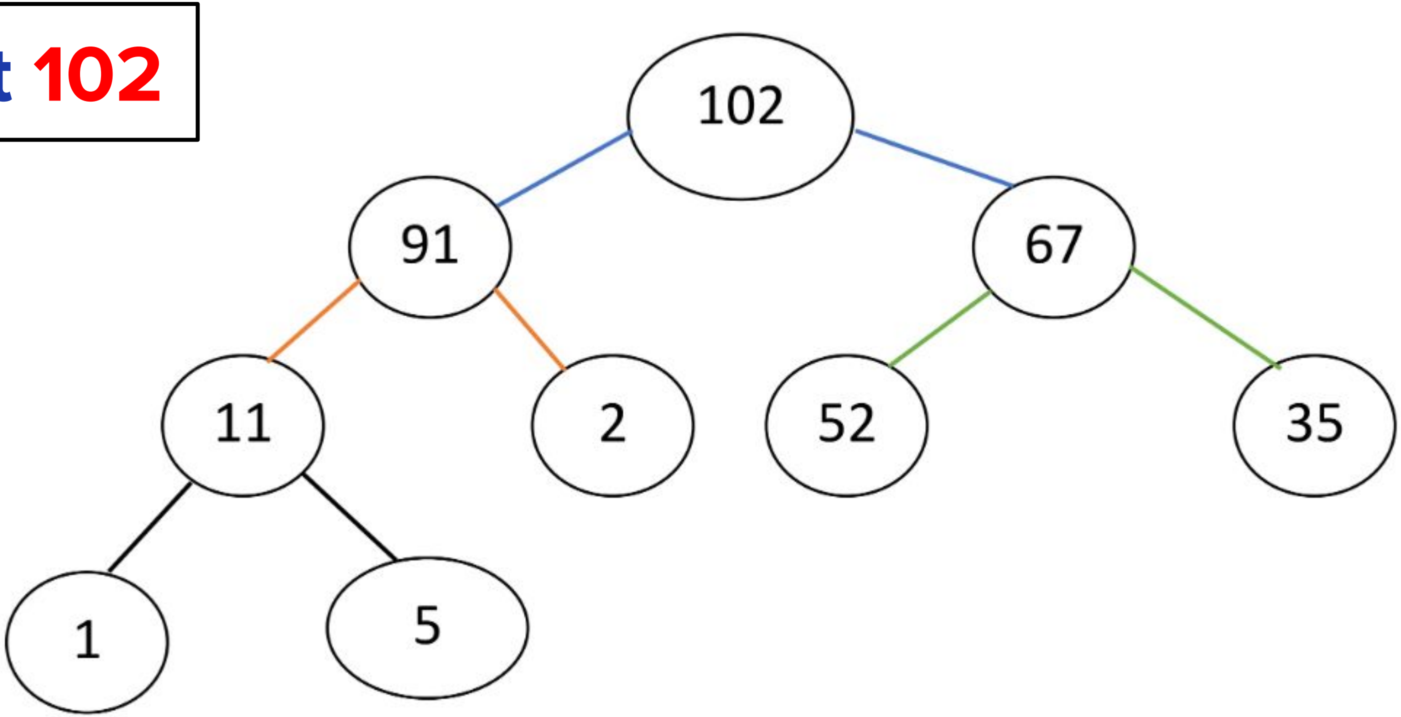
Insert 102



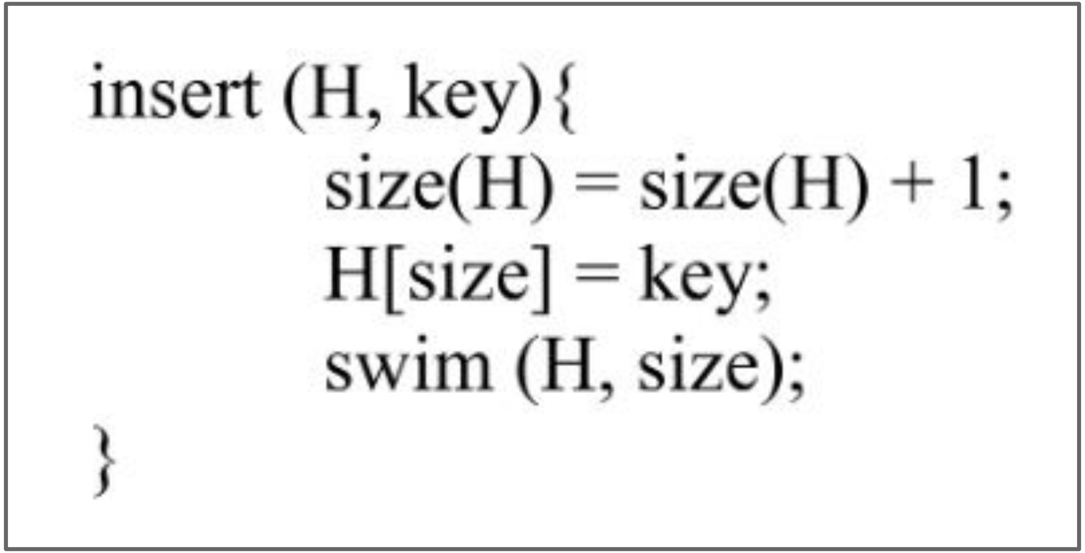
Swim()
UpHeapify()
IncreaseKey()

Heap - Insert

Insert 102

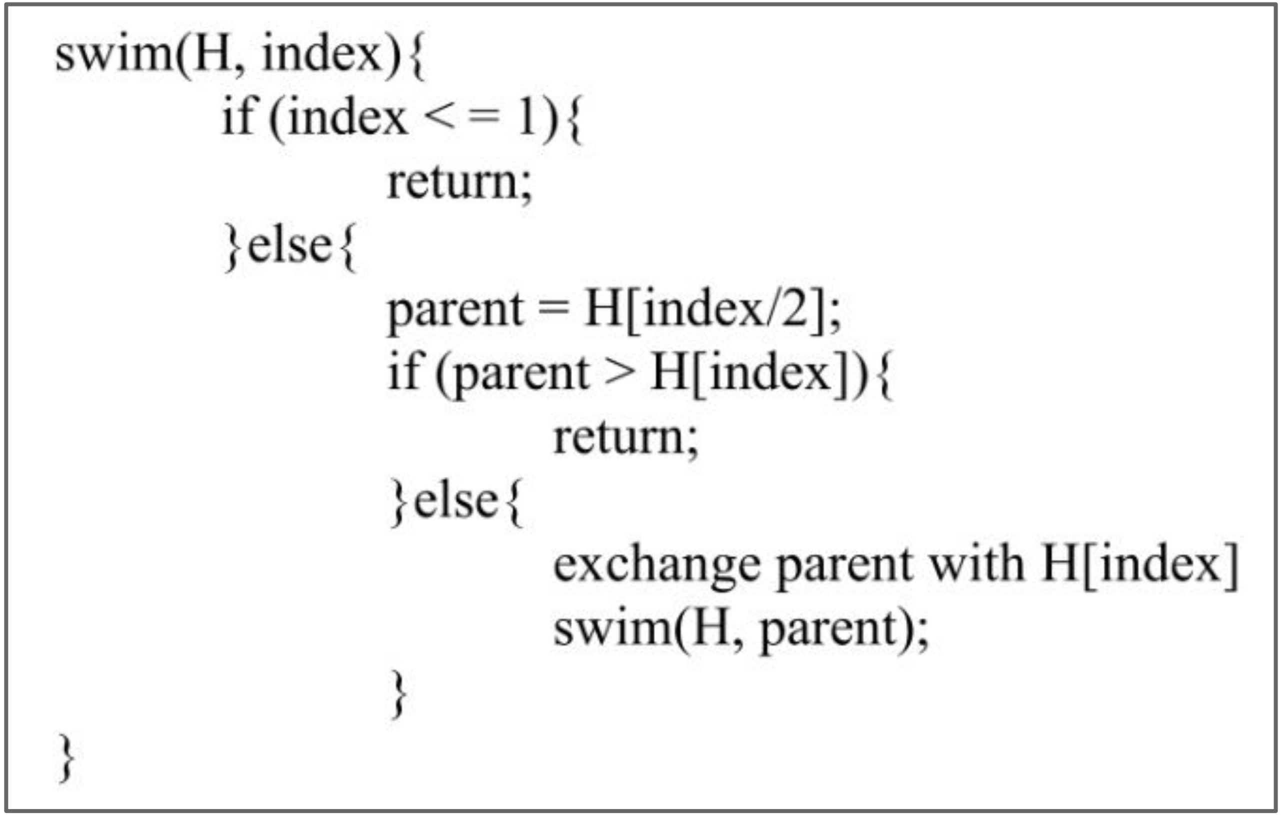


Heap - Insert (Pseudocode)



```
insert (H, key){
        size(H) = size(H) + 1;
        H[size] = key;
        swim (H, size);
}
```

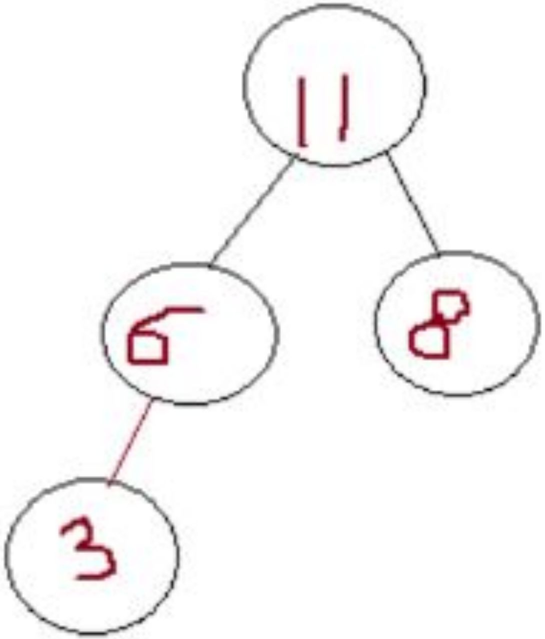
Heap - Swim (Pseudocode)



```
swim(H, index){
        if (index < = 1){
                return;
        }else{
                parent = H[index/2];
                if (parent > H[index]){
                        return;
                }else{
                        exchange parent with H[index]
                        swim(H, parent);
                }
}
```

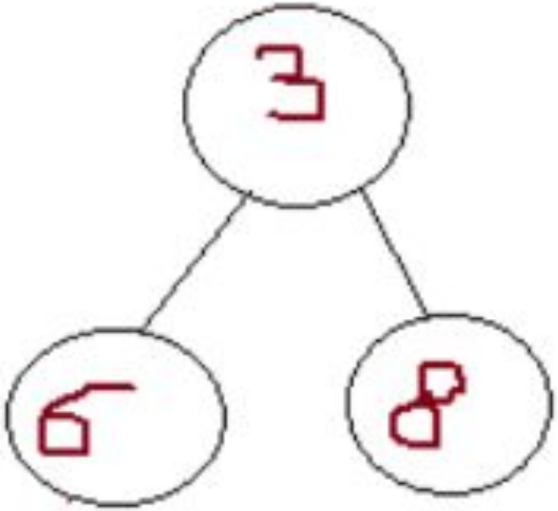
Heap - Delete

Delete Root



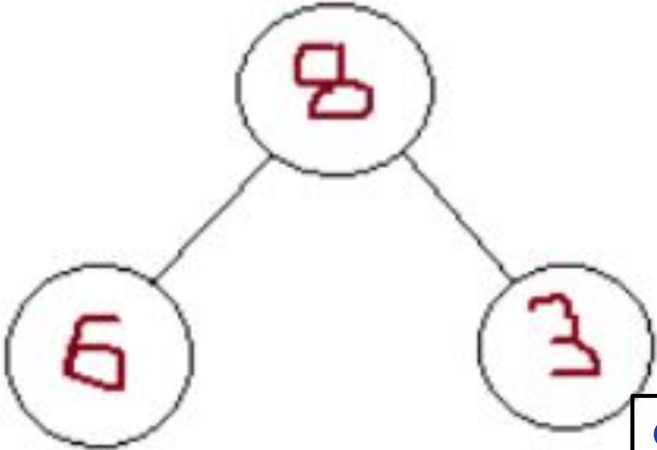
Heap - Delete

Delete Root



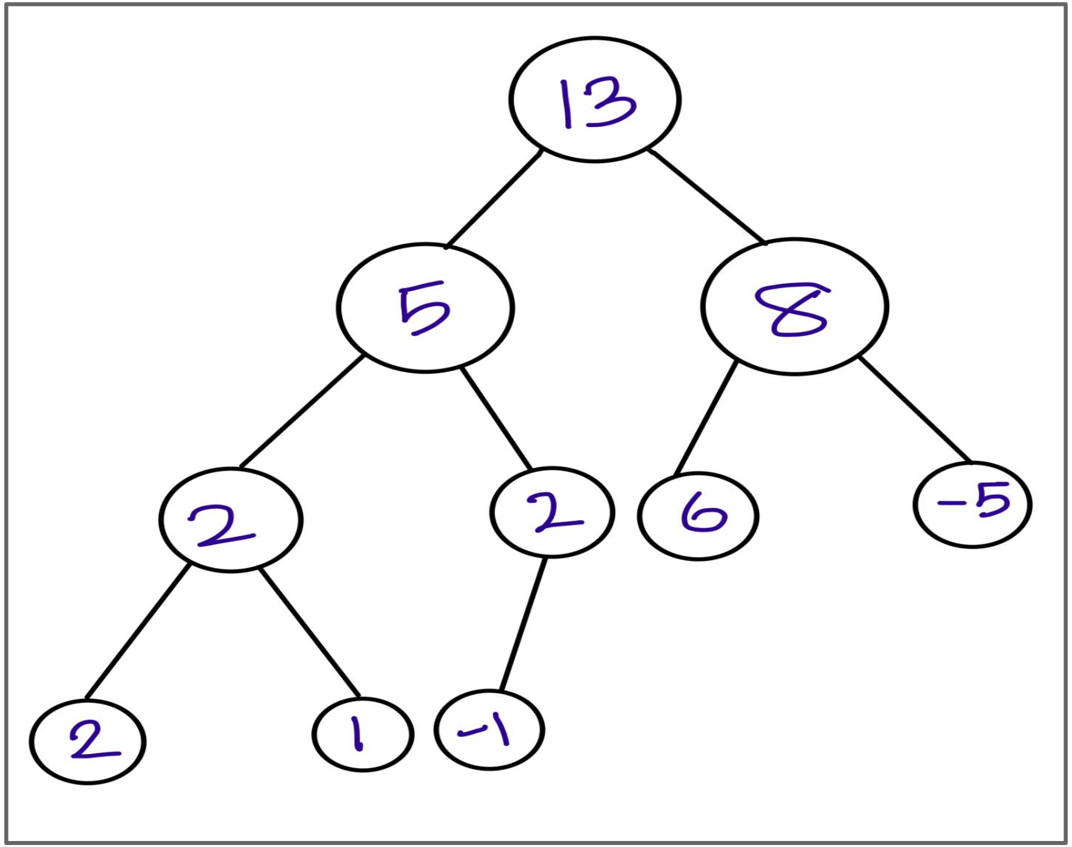
Heap - Delete

Delete Root

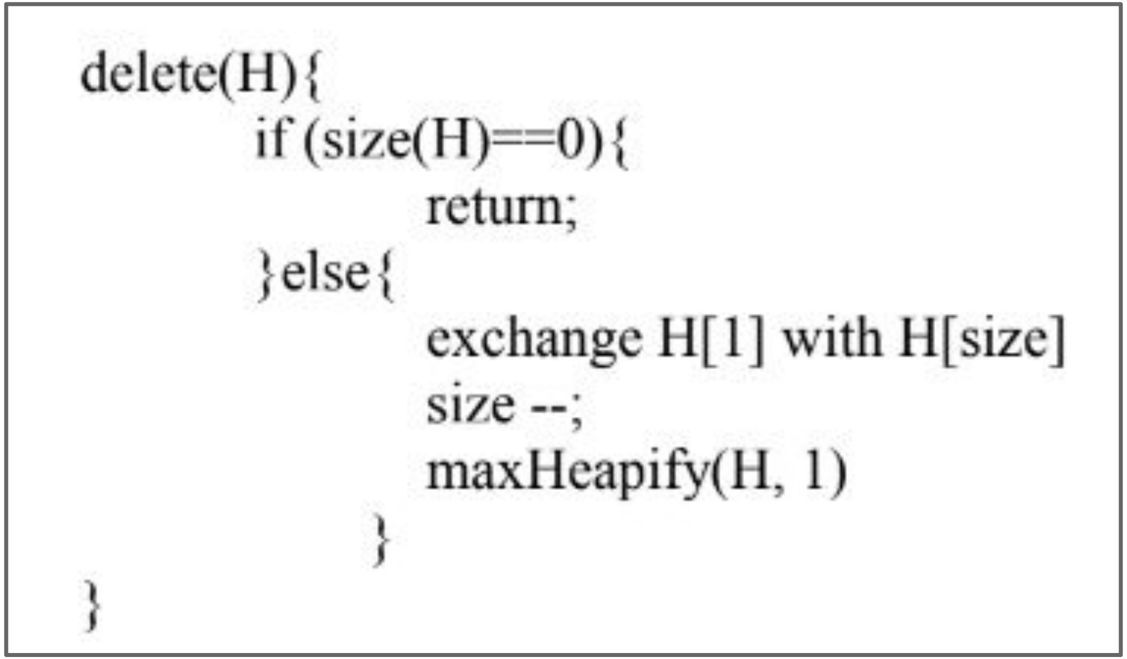


Sink()
DownHeapify()
MaxHeapify()

Heap - Delete

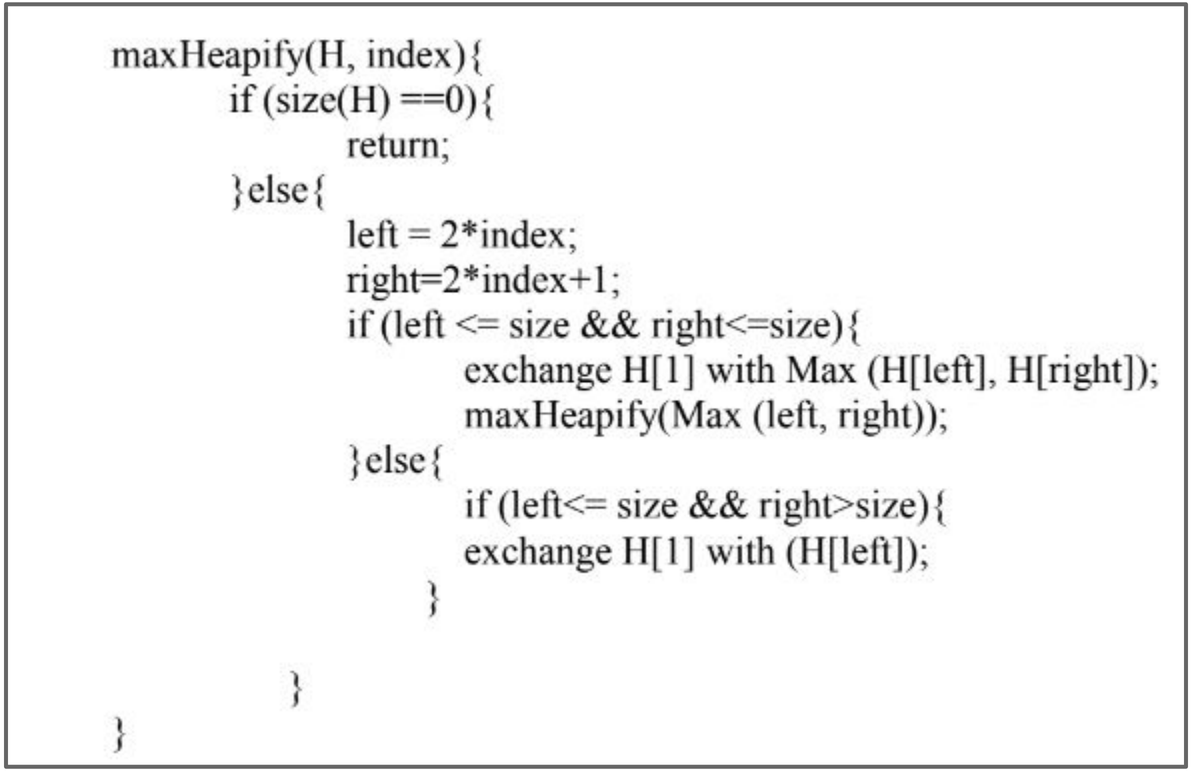


Heap - Delete (Pseudocode)



```
delete(H){
        if (size(H)==0){
                return;
        }else{
                exchange H[1] with H[size]
                size --;
                maxHeapify(H, 1)
            }
}
```

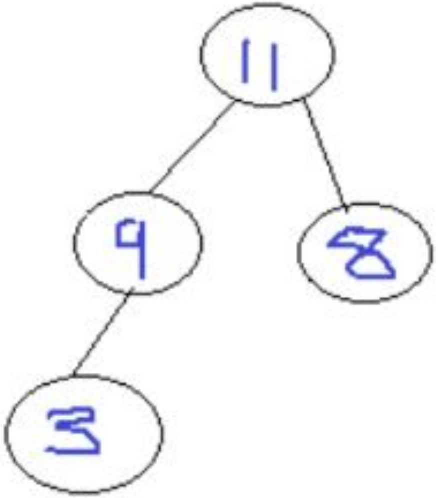
Heap - Sink (Pseudocode)



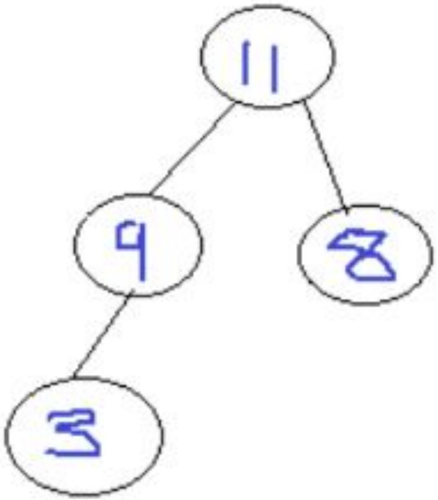
```
maxHeapify(H, index){
        if (size(H) ==0){
                return;
        }else{
                left = 2*index;
                right=2*index+1;
                if (left <= size && right<=size){
                        exchange H[1] with Max (H[left], H[right]);
                        maxHeapify(Max (left, right));
                }else{
                        if (left<= size && right>size){
                        exchange H[1] with (H[left]);
                      }

            }
}
```

Heap Sort

Delete All Nodes

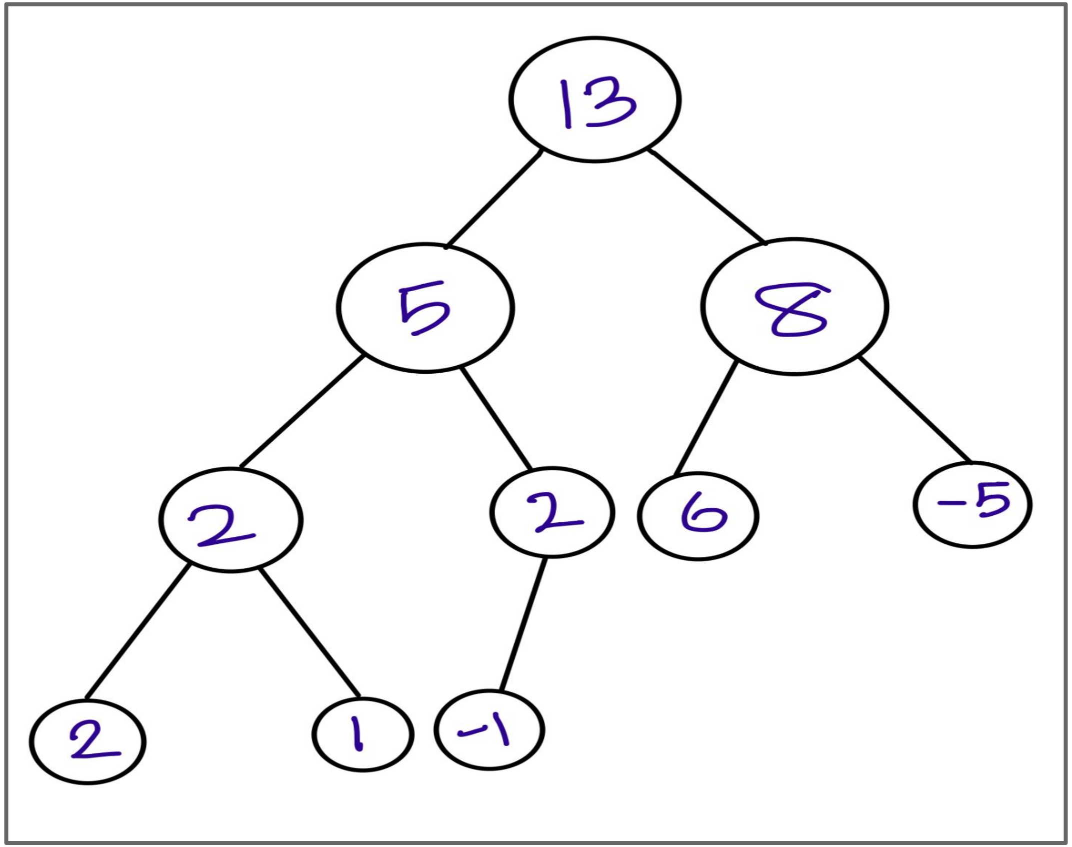


Heap Sort



3	8	9	11

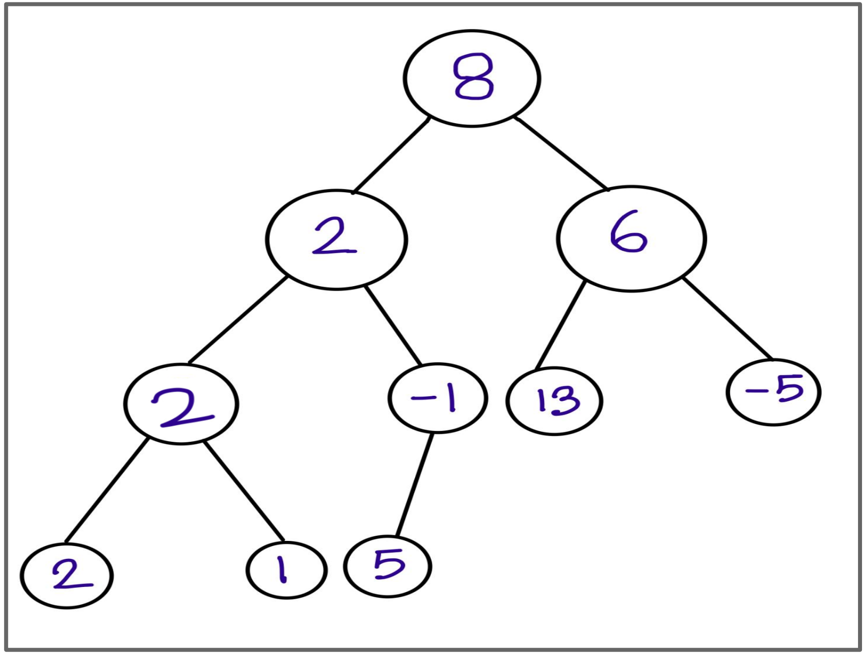
Heap Sort



Heap - Build Heap

N	8	2	6	2	-1	13	-5	2	1	5

Tree

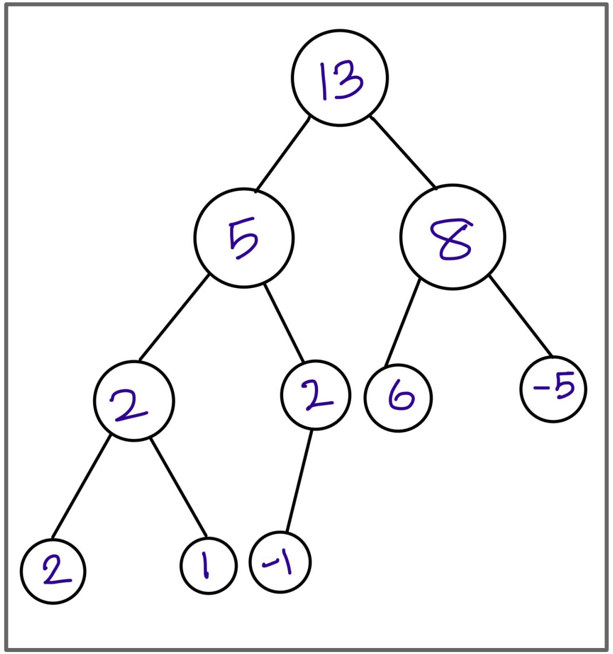


Heap - Build Heap

N	8	2	6	2	-1	13	-5	2	1	5

```
for all nodes i = 1 to n{
        swim (H, i);
}
```

Heap

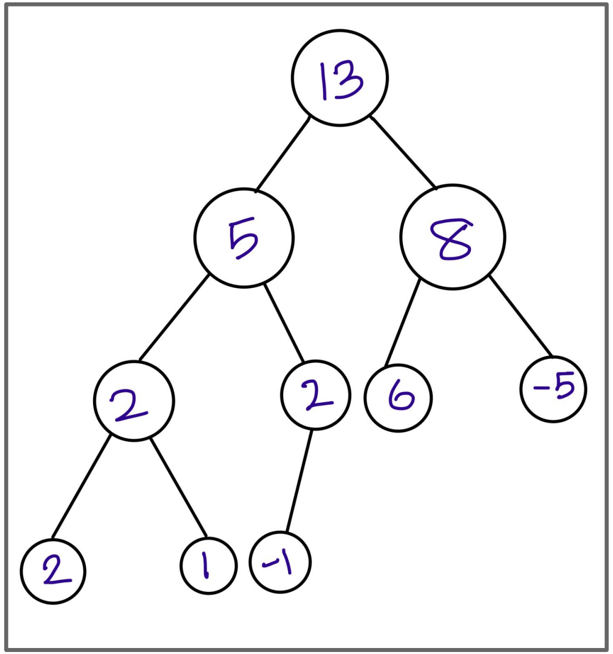


Heap - Build Heap

N	13	5	8	2	2	6	-5	2	1	-1

```
for all nodes i = 1 to n{
        swim (H, i);
}
```

Heap


13
5
8
2
2
6
-5
2
1
-1