# Assignment 6

---

Name : Hrishikesh Rajan
Email : hrishikeshrajan3@gmail.com

---

Q.1)

1. arr = [1, 3, 7, 9, 12, 10, 8, 16, 18, 22, 27]
   Create a buildHeap method that returns a minheap.

   heapify(arr, n, i):
   // Write your own code

   buildHeap(arr, n):
   //Write your own code
   heapify(arr, n, i)

Ans)

* javascript implementation

**Code:**

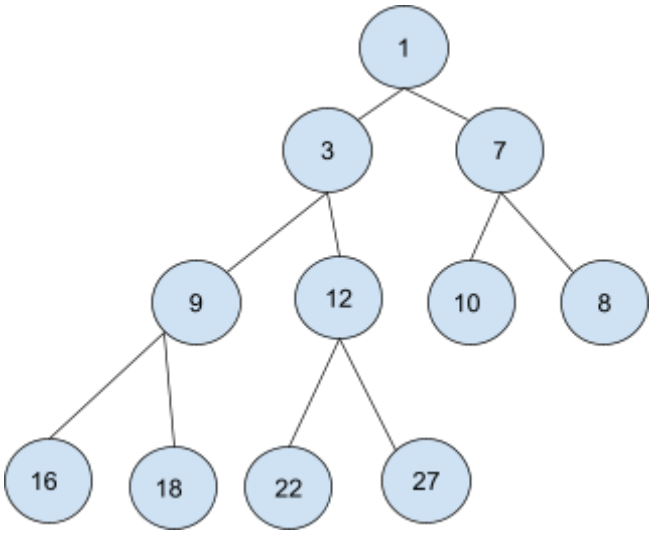
```
function swap(nthIndex,parentIndex,heap){
    let temp = heap[parentIndex];
    heap[parentIndex] = heap[nthIndex];
    heap[nthIndex] = temp;
    return heap;
}

function buildHeap(heap,arr,num){
    heap = arr;
    let n = num;
    let height = Math.floor((arr.length-1)/2)
    if(n<0){
    return heap
    }
    for (let i = 0; i < height; i++) {
    let parent = (Math.round(n/2))-1
    if(parent < 0){
        break;
    }
    if(heap[n] < heap[parent]){
        swap(n,parent,heap)
       }
       n = parent;
   }
return buildHeap(heap,arr,--num);
}

function minHeap(arr){
    let heap=[];
return  buildHeap(heap,arr,arr.length-1);
}

const arr = [1, 3, 7, 9, 12, 10, 8, 16, 18, 22, 27]
const result = minHeap(arr);
console.log(result);
```

**Output :**




**Comparison flowChart:**

* The output generated by printing nodes and indexes dynamically

childIndex: 10 value => 27 , parentIndex: 4 value => 12
childIndex: 4 value => 12 , parentIndex: 1 value => 3
childIndex: 1 value => 3 , parentIndex: 0 value => 1

childIndex: 9 value => 22 , parentIndex: 4 value => 12
childIndex: 4 value => 12 , parentIndex: 1 value => 3
childIndex: 1 value => 3 , parentIndex: 0 value => 1

childIndex: 8 value => 18 , parentIndex: 3 value => 9
childIndex: 3 value => 9 , parentIndex: 1 value => 3
childIndex: 1 value => 3 , parentIndex: 0 value => 1

childIndex: 7 value => 16 , parentIndex: 3 value => 9
childIndex: 3 value => 9 , parentIndex: 1 value => 3
childIndex: 1 value => 3 , parentIndex: 0 value => 1

childIndex: 6 value => 8 , parentIndex: 2 value => 7
childIndex: 2 value => 7 , parentIndex: 0 value => 1

childIndex: 5 value => 10 , parentIndex: 2 value => 7
childIndex: 2 value => 7 , parentIndex: 0 value => 1

childIndex: 4 value => 12 , parentIndex: 1 value => 3
childIndex: 1 value => 3 , parentIndex: 0 value => 1

childIndex: 3 value => 9 , parentIndex: 1 value => 3
childIndex: 1 value => 3 , parentIndex: 0 value => 1

childIndex: 2 value => 7 , parentIndex: 0 value => 1

childIndex: 1 value => 3 , parentIndex: 0 value => 1

## Analysis :

Total No of Nodes = 11;

*Reference
Level =0;
Height = 1;

To find the height of the tree

$N = 2^{h} - 1$

$N + 1 = 2^{h}$

Taking log on both sides

$log(N + 1) = log(2^{h})$

$h * log_{2} 2 = log(N + 1)$

$h = log(N + 1)$

Taking the upper bound of the $h$

$h = log(11 + 1)$

$h = 4$

## Explanation:

*Time Complexity = No of Comparisons + No of swaps*

From the given input we can see that the input array is already sorted.
Hence,

No of swaps = 0; Best case;
Therefore, swaps can be neglected

Comparing the Comparison flowchart, we see that the number of comparisons has been reduced by 2 (Parent node).

Hence,

No of comparison = n*Log(n)

n is the index of nodes starting at the leaf node. For each node n, the comparison goes until the root node with log(n) complexity

Total Time Complexity = (n * log ( n ) ) + 0;

= O ( n log( n ) )

Auxiliary Space = O ( n ), New array is created for heap