Heap:

Heap is a complete or almost complete binary tree.

Alignment of the child is from left to right.

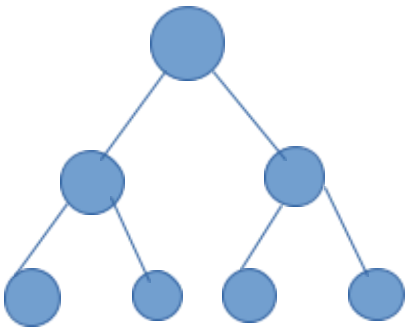


Fig. Complete binary tree

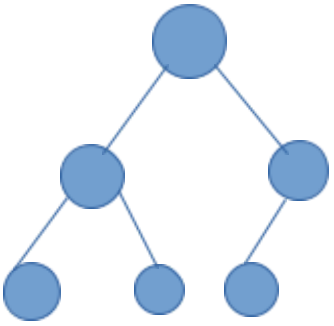
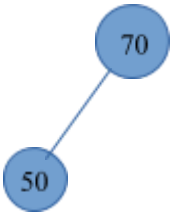
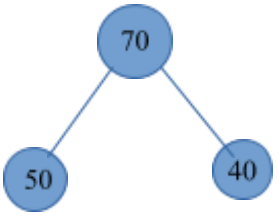


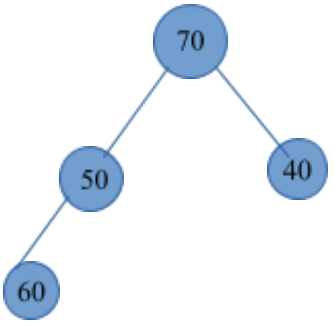
fig. Almost complete binary tree

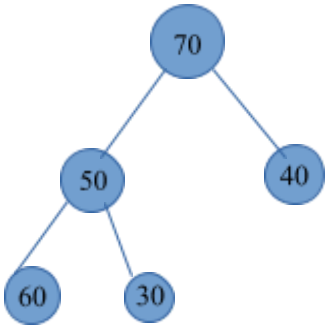
Lets build up the heap of the following

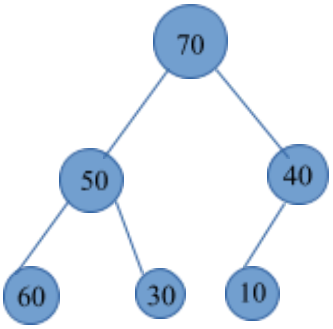
70, 50, 40, 60, 30, 10, 20

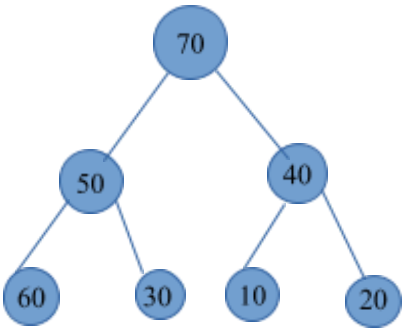








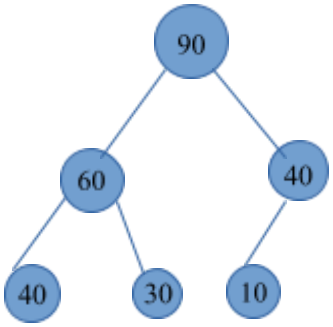




Heap are two types **max heap** and **min heap**

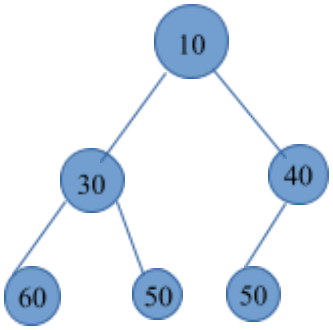
Max Heap: Parent nodes value are greater than its' child

example

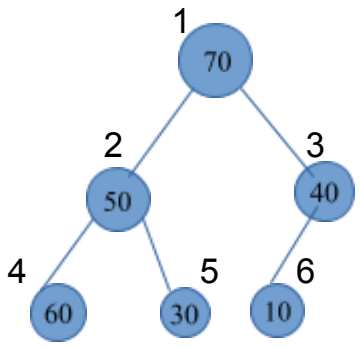


Min Heap: Parent nodes value are less than its' child

example



Lets take an example of heap



If I represents it using an array then it will be following

index: 1	2	3	4	5	6
70	50	40	60	30	10

It means if the root/ parent index is i then

left child index will be 2i and // left shift

right child index will be 2i + 1 // left shift + 1

If any child index is i then

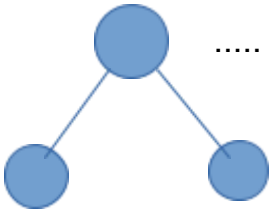
parent index will be $\lfloor i/2 \rfloor$ // right shift

If the input list is sorted order and if we want to build up a heap the heap will be either max heap(for descending order) or min heap(for ascending order).

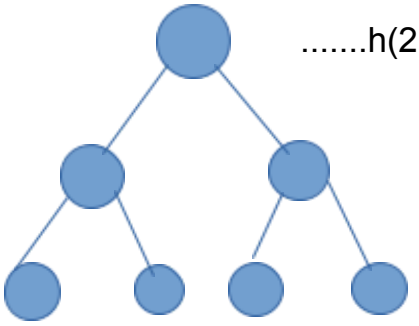
Therefore if the input list is unsorted order and we sorted it first by applying any best comparison sort technique then time complexity will be O(n log n).

can we build up the heap of O(n)?

Before we solve that problem first we learn about some properties of complete binary tree



..... h(1)



.....h(2)

Height of tree(h)	1	2	3	4
Max no. Of nodes	3	7	15	31

so max no. nodes of complete binary tree= $2^{h+1} - 1$

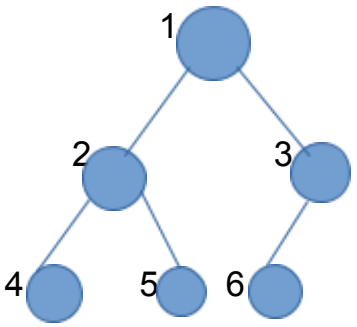
if no. Of nodes of complete or almost complete binary tree is n then

height of the tree(h)= $\lfloor \log n \rfloor$

for complete or almost complete binary tree(also heap) if the no. Of nodes n then

leaf nodes started from: $\lfloor n/2 \rfloor + 1$ to n

non leaf node started from: 1 to $\lfloor n/2 \rfloor$



leaf nodes: $\lfloor n/2 \rfloor + 1$ to n

= 6/2 + 1 to 6

= 4 to 6

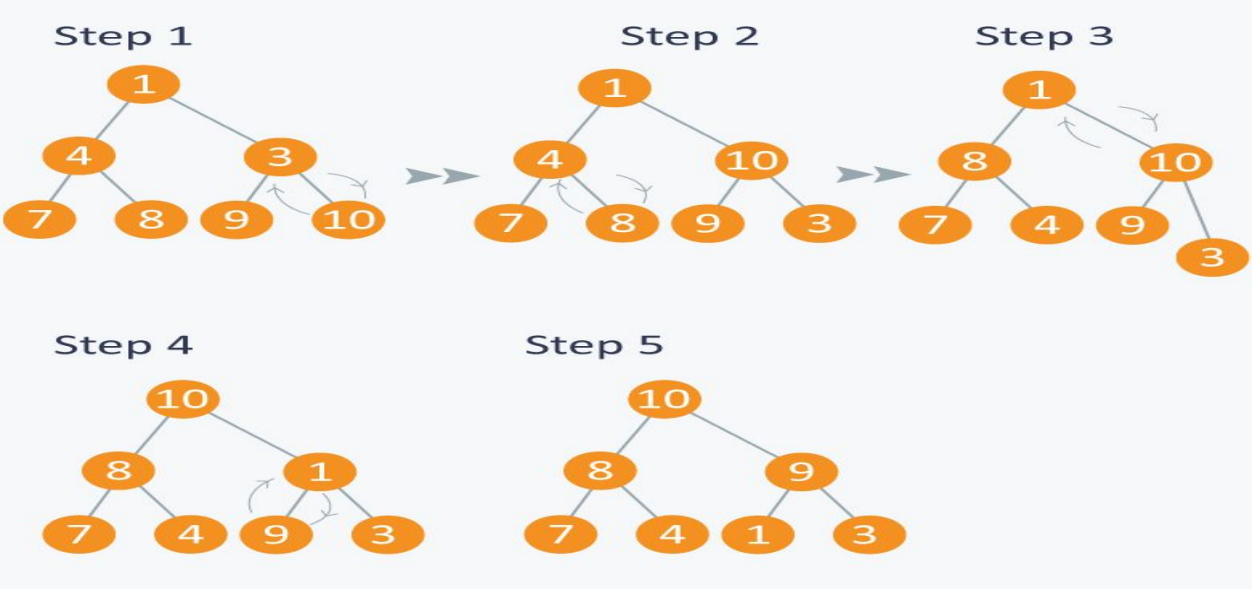
non-leaf nodes: 1 to $\lfloor n/2 \rfloor$

= 1 to 6/2

= 1 to 3

Max Heapify

How to build up a max heap? Consider the following example



In a max heap or min heap properties all the leaf nodes treated as a heap of single node. So in the above example we start apply max heap properties from longest non-leaf to root i.e. node 3 to node 1($\lfloor n/2 \rfloor$ to 1).

At first we compare longest non-leaf (i.e. a root of a sub tree) with its' child. If the root node value is less than of its' child then maximum child node value will be swap with root node.

```
Max-Heapify (A, i)
{
   left ← 2*i
   right ← 2*i + 1
   largest ← i

  if left ≤ heap_length[A] and A[left] > A[largest] then
      largest ← left

   if right ≤ heap_length[A] and A[right] > A[largest] then
      largest ← right

   if largest ≠ i then
      swap A[i] and A[largest]
   Max-Heapify(A, largest)
}
```

Time complexity: In the above example from step 3 root node 1 come down to leaf at step 5. So root node traverse all the levels i.e. the total height of the tree. So total time taken is O(log n).

Build-Max-Heap:

Build-Max-Heap(A)

{

A.Heap_size=A.length;

for(i=⌊A.length/2⌋ to 1)

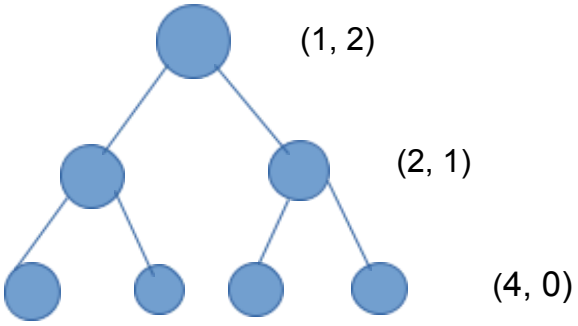
Max-Heapify (A, i);

}

Time Complexity: Line-3 of Build-Max-Heap runs a loop from the index of the last internal node (Heap_size/2) with height=1, to the index of root(1) with height = lg(n).

Hence, Max_Heapify takes different time for each node, which is O(h).

A heap of size n has at most $\lceil n/2^{h+1} \rceil$ nodes with height h.



$$T(n)= \sum_{h=0}^{\lceil \log n \rceil} \frac{n}{2^{h+1}} O(h) = O\left(n \sum_{h=0}^{\lceil \log n \rceil} \frac{h}{2^{h+1}}\right)$$
$$= O\left(\frac{n}{2} \sum_{h=0}^{\lceil \log n \rceil} \frac{h}{2^{h}}\right)$$
$$= O\left(n \sum_{h=0}^{\infty} \frac{h}{2^{h}}\right)$$
$$= O(2n)$$
$$= O(n)$$

Heap Sort:

Sorting Strategy:

1. Build Max Heap from unordered array;

2. Find maximum element A[1];

3. Swap elements A[*n*] and A[1]: now max element is at the end of the array!

4. Discard node *n* from heap (by decrementing heap-size variable)

5. New root may violate max heap property, but its children are max heaps. Run max_heapify to fix this.

6. Go to Step 2 unless heap is empty.