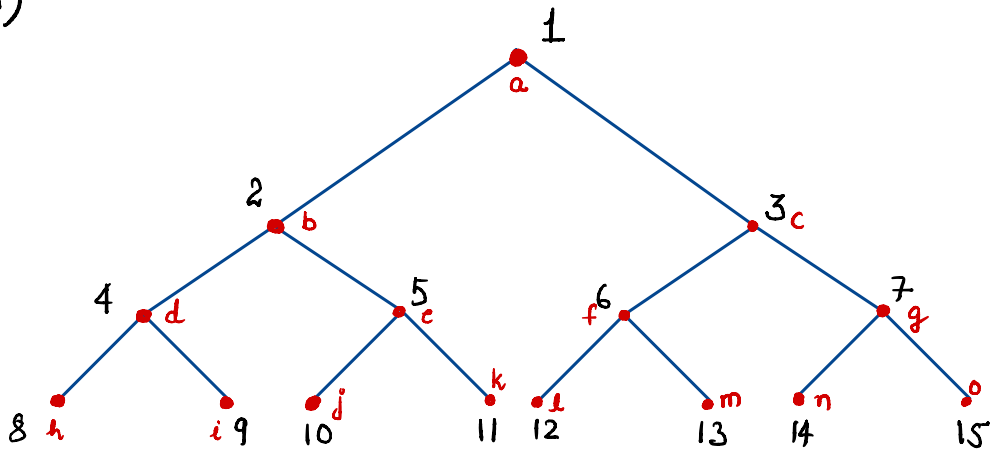
Complete Binary tree (completely filled)

No. of Nodes	Height (Max)
1	0
2	1
3	1
4	2
5	2
6	2
7	2
8	3
9	3



$$\text{Maximum height} = \left\lceil \log_2 (N+1) \right\rceil - 1$$

If you want to represent the above heap in an array.

Size of the array :- N+1 ('0' index empty)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o

parent of node at index i = parent(i) = $\left\lfloor \frac{i}{2} \right\rfloor$ (if $i \neq 1$)

left child of a node at index i = leftchild(i) = $2i$ (if $2i <= N$) else its a leaf node.

Right child of a node at index i = rightchild(i) = $2i+1$ (if $2i+1 <= N$) else its a leaf node.

Recall Insertion Operation:

1. Append the node to be inserted in the Array.
2. Traverse from the inserted node to the root till it satisfy heap property (particularly value property as structure property is already satisfied)

→ Heapify → for max-heap, we call it Max Heapify BU (Bottom-up)

```
MaxHeapifyBU( A, i )   A → heap (Array Representation)
                       i → index of the node on which heap property needs to be satisfied
    parent_idx ← ⌊i/2⌋
    if parent_idx != 0
        if A[i] > A[parent_idx]
            swap( A[i], A[parent_idx] )
            MaxHeapifyBU( A, parent_idx )   → Recursive calling MaxHeapify till it satisfies the heap property
        end if
    end if
end MaxHeapifyBU
```

Recall delete_Max Operation on Max-heap.

1. Swap value of last node and value of root node.
2. Delete the last node.
3. Traverse from root to leaf till it satisfy the value property of the heap.
   → MaxHeapifyTD (Top down from root to leaf)

```
MaxHeapifyTD( A, i, n )   n → heap size
                          i → Index on which MaxHeapifyTD is called
    largest_idx ← i
    leftChild_idx ← 2i
    rightChild_idx ← 2i+1
    if ( leftChild_idx <= n ) and ( A[leftChild_idx] > A[i] )
        largest_idx ← leftChild_idx
    endif
```

```
    if ( rightChild_idx <= n) and ( A[rightChild_idx] > largest_idx)
        largest_idx ← rightChild_idx
    end if
    if ( i ≠ largest_idx)
        swap( A[i], A[largest_idx] )
        MaxHeapifyTD( A, largest_idx, n)
    end if
end MaxHeapifyTD
```

Given a list of values $\{a_1, a_2 \cdots, a_n\}$

Task : Build a heap from these values.

$A =$ | – | a_1 | a_2 | ..... | a_{n-2} | a_{n-1} | a_n |

```
BuildHeap(A)
    for ( i = 2 to n)        ← heap size
        MaxHeapifyBU(A, i)
    end for
end BuildHeap
```

using MaxHeapifyBU

```
BuildHeap(A)
    for i = ⌊n/2⌋ to 1
        MaxHeapifyTD(A, i, n)        ← heap size
    end for
end BuildHeap
```

Using MaxHeapifyTD

Time Complexity for BuildHeap using MaxHeapifyBU

Total No. of swappings (worst Case) $\propto T(n)$ (h = height of tree = $O(\log n)$)

$$T(n) = \frac{n}{2} \times h + \frac{n}{2^2}(h-1) + \frac{n}{2^3}(h-2) + \cdots + 1 \cdot 0$$

$$\leq \frac{n}{2} \times h + \frac{n}{2^2} h + \frac{n}{2^3} h + \cdots + 1 \cdot h$$

$$= \frac{nh}{2}\left(1 + \frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^h}\right)$$

Lot of simplifying Assumption

$$= \frac{nh}{2}\left(\sum_{k=0}^{h}\frac{1}{2^k}\right)$$

$$\leq \frac{nh}{2}\left(\sum_{k=0}^{\infty}\frac{1}{2^k}\right)$$

$$= nh$$

$$T(n) \leq nh$$

$$T(n) = O(n\log n) \quad \text{——— (1)}$$

$$T(n) = \frac{n}{2}\times h + \frac{n}{2^2}(h-1) + \frac{n}{2^3}(h-2) + \cdots + 1\cdot 0$$

$$\geq \frac{nh}{2}$$

$$T(n) = \Omega(n\log n) \quad \text{——— (2)}$$

from (1) and (2) $\Rightarrow T(n) = \Theta(n\log n)$

Time Complexity for BuildHeap using MaxHeapifyTD

Total No. of swappings (worst Case) $\propto T(n)$ (h = height of tree = $O(\log n)$)

$$T(n) = \frac{n}{2}\cdot 0 + \frac{n}{2^2}1 + \frac{n}{2^3}\cdot 2 + \cdots + \frac{n}{2^{h+1}}\cdot h$$

$$= \frac{n}{2^2}\left(\sum_{k=1}^{h} k\cdot\frac{1}{2^{k-1}}\right)$$

$$= \frac{n}{2^2}\left(\sum_{k=0}^{h} k\cdot\frac{1}{2^{k-1}}\right)$$

$$\leq \frac{n}{2^2}\left(\sum_{k=0}^{\infty} k\cdot\frac{1}{2^{k-1}}\right)$$

$x = 1/2$ $\quad \sum_{k=0}^{\infty} k\cdot r^{k-1} = ?$

We know

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

$$\frac{d}{dx}\left(\sum_{k=0}^{\infty} x^k\right) = \frac{d}{dr}\left(\frac{1}{1-r}\right)$$

$$\sum_{k=0}^{\infty} k\,x^{k-1} = \frac{(-1)}{(1-x)^2}(-1)$$

$$\sum_{k=0}^{\infty} k\cdot\frac{1}{2^{k-1}} = \frac{1}{(1-1/2)^2} = 4$$

$$T(n) \leq \frac{n}{2^2}\left(\sum_{k=0}^{\infty} k \cdot \frac{1}{2^{k-1}}\right)$$

$$T(n) \leq n$$

$$T(n) = O(n)$$

Also trivally $T(n) = \Omega(n)$ (see Algorithm)

Hence, $T(n) = \Theta(n)$