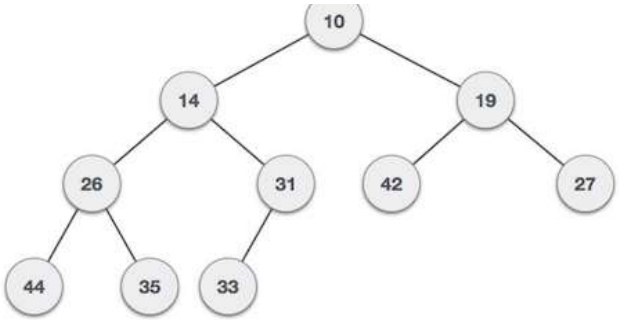
Heap Data Structure

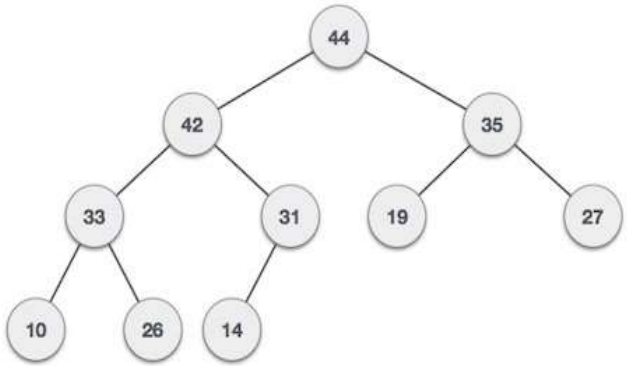
Govinda K
SCOPE

- Heap is a special case of balanced binary tree data structure where the root-node key is compared with its children and arranged accordingly. If **α** has child node **β** then.
- **key(α) ≥ key(β)**
- As the value of parent is greater than that of child, this property generates **Max Heap**. Based on this criteria, a heap can be of two types

- **Min-Heap**

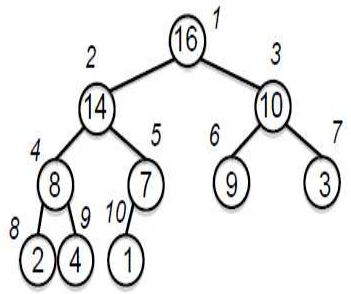


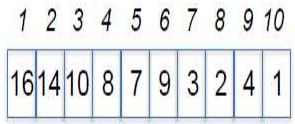
- **Max-Heap**



Heaps

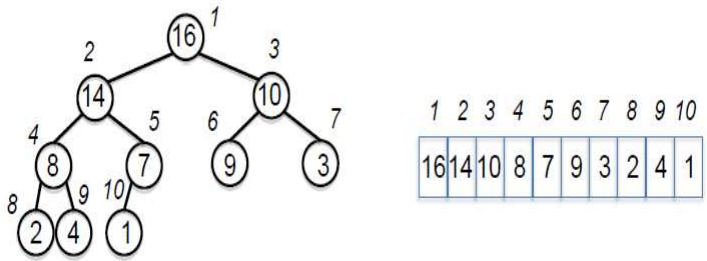
- An array, visualized as a nearly complete binary tree
- Max Heap Property: The key of a node is ≥ than the keys of its children





Heap as a Tree

- root of tree: first element in the array, corresponding to i = 1
- parent(i) =i/2: returns index of node's parent
- left(i)=2i: returns index of node's left child
- right(i)=2i+1: returns index of node's right child



- To create a min heap we first build a heap then we transform the heap into a min heap

Input 40 60 10 20 50 30

There are 6 elements
so our heap will have 6 nodes

We can represent the nodes of the heap in an array

Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

Create a min heap

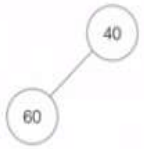
- In a min heap the parent node is always smaller than or equal to its child node
- We can represent a min heap using one dimensional array
- If N denotes the index of a parent node then
- 2N denotes the left child node and
- 2N+1 denotes the right child node
- Where N=1,2,3

Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

Build Heap

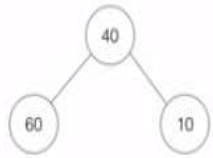
Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

Build Heap



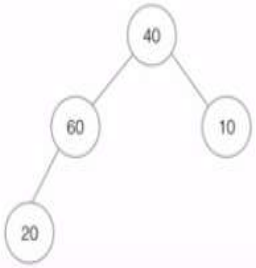
Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

Build Heap



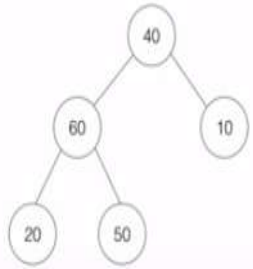
Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

Build Heap



Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

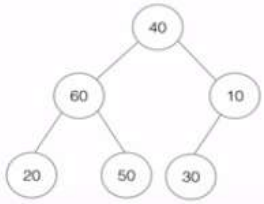
Build Heap



Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

↑

Build Heap



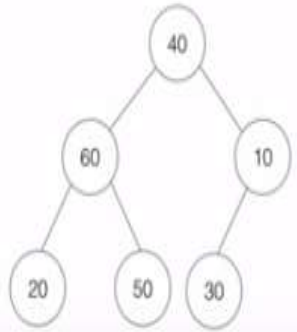
- We have build the heap now we need to transform it into Min heap

Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

Note!
In a Min Heap the parent node is always smaller than or equal to its child nodes

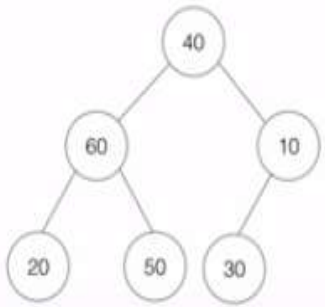
Note!
If there are N nodes then we start comparison from floor(N/2) index

Create a Min Heap



Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

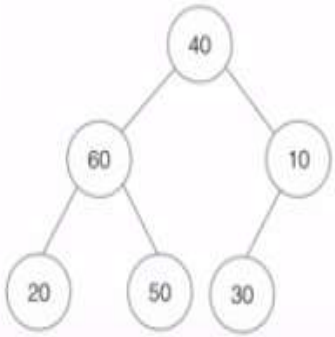
Create a Min Heap



Example:
If N = 5
Then floor(N/2)
= floor(5/2)
= floor(2.5)
= 2
We lower down the value to nearest integer

Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

Create a Min Heap

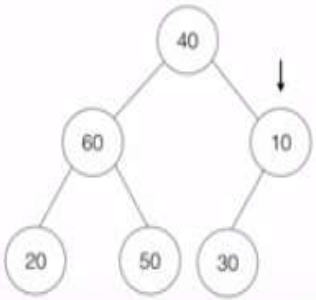


In this case there are 6 nodes
i.e., N = 6
So, floor(N/2)
= floor(6/2)
= floor(3)
= 3

We will start comparison from index 3
or the 3rd node

Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

Create a Min Heap



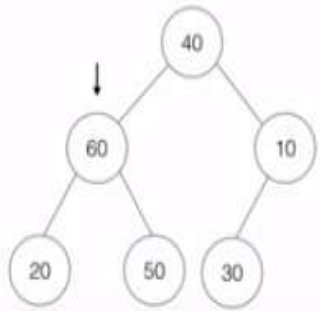
Is there any child node smaller than 10?

NO

So, we move to index 2 or the 2nd node

Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

Create a Min Heap



Is there any child node smaller than 60?

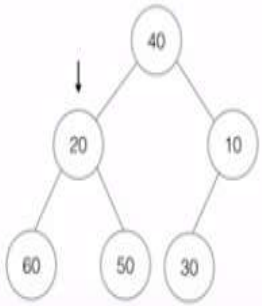
YES, its 20

So, we swap position of 20 and 60

Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

Create a Min Heap

Remember!
We swap position only with the smallest child to create Min Heap



Is there any child node smaller than 60?

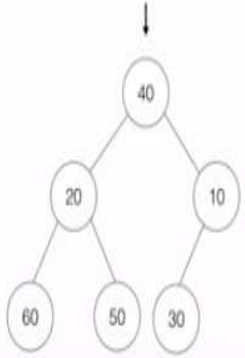
YES, its 20

So, we swap position of 20 and 60

Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

Create a Min Heap

Remember!
We swap position only with the smallest child to create Min Heap



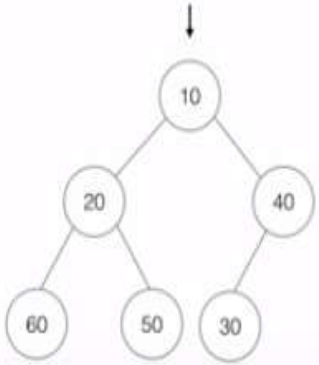
Is there any child node smaller than 40?

YES, its 10

So, we swap position of 10 and 40

Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

Create a Min Heap



Is there any child node smaller than 40?

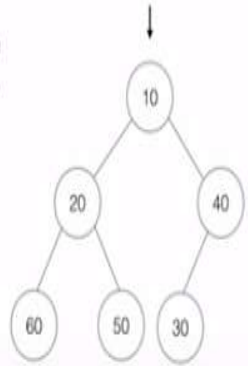
YES, its 10

So, we swap position of 10 and 40

Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

Now that we have reached the 1st node we will check whether we have created a Min Heap

Create a Min Heap



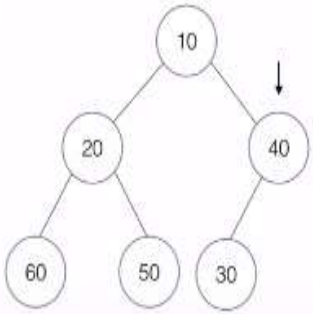
Is there any child node smaller than 40?

YES, its 10

So, we swap position of 10 and 40

Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

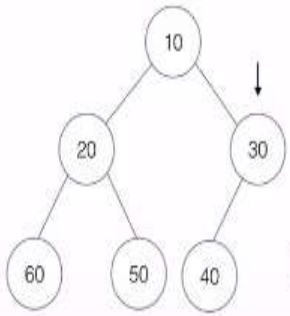
Create a Min Heap



Looking at the heap we find 40 is greater than its child node 30 so, we have to correct that by swapping positions of 30 and 40

Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

Create a Min Heap

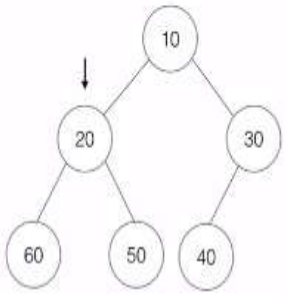


Looking at the heap we find 40 is greater than its child node 30 so, we have to correct that by swapping positions of 30 and 40

Now we move to the 2nd node

Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

Create a Min Heap



20 has no child smaller than itself

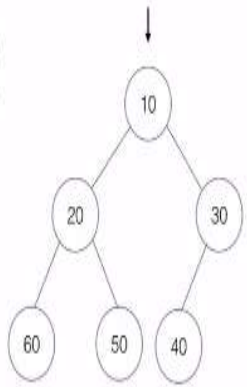
So we move to the 1st node

Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

Now that we have reached the 1st node we will check whether we have created a Min Heap

Remember!
In Min Heap
a parent node is always smaller than or equal to its child nodes

Create a Min Heap



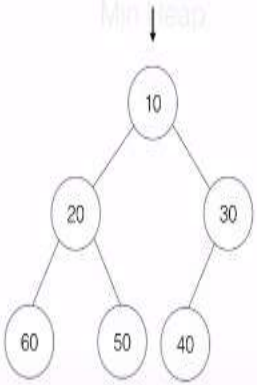
10 has no child smaller than itself

Array Index	0	1	2	3	4	5	6
Input		40	60	10	20	50	30

Now that we have reached the 1st node we will check whether we have created a Min Heap

Remember!
In Min Heap
a parent node is always smaller than or equal to its child nodes

Create a Min Heap



Looking at the heap we find that all parent node are smaller than their respective child node

So we have a Min Heap

Heap Operations

- build_max_heap : produce a max-heap from an unordered array
- max_heapify : correct a single violation of the heap property in a subtree at its root