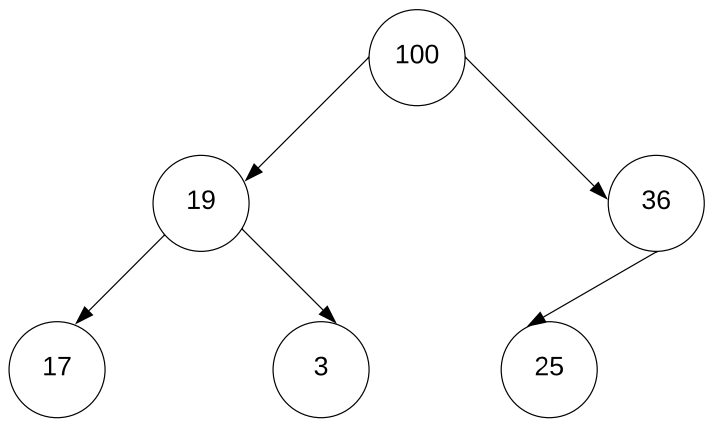
Array-based Representation of Heap



In case of 1-based Indexing, means root node is present at index 1, some people uses 1-based indexing for better understanding, its totally your choice.

	-1	100	19	36	17	3	25
Index:	0	1	2	3	4	5	6

Storing -1 at index 0 because we don't want to use index 0.

For any Node at index "i", in case of 1-based indexing,

1. **Left Child** of "i" is present at index: **(2 * i)**th index.
2. **Right Child** of "i" is present at index: **(2 * i + 1)**th index.
3. **Parent Node** of "i" is present at index: **(i / 2)**th index.

Example:

For node at index 1 i.e., 100,

Left Child of 100 is present at index 2*i i.e., 2*1 = 2. (because i = 1).
So the left child of 100 is present at index 2 and the element at index 2 is 19, so the left child of 100 is 19 and you can verify it through the diagram.

Right Child of 100 is present at index 2*i+1 i.e., 2*1+1 = 2+1 = 3. (because i = 1).
So the right child of 100 is present at index 3 and the element at index 3 is 36, so the right child of 100 is 36 and you can verify it through the diagram.

Parent Node of 100 i present at index (i / 2) i.e., (1/2) = 0/5. (because i = 1).
and 0.5 is not a valid index, it means there is no parent node of 100.

For node at index 2 i.e., 19,

Left child of 19 is present at index 2*i i.e., 2*2 = 2+2 = 4. (because i = 2).
Element at index 4 is 17 so the left child of 19 is 17.

Right child of 19 is present at index 2*i+1 i.e., 2*2+1 = 4+1 = 5. (because i = 2).
Element at index 5 is 3 so the right child of 19 is 3.

Parent Node of 19 is present at index (i / 2) i.e., (2 / 2) = 1. (because i = 2).
It means parent node of 19 is present at index 1 and element at index 1 is 100, so , 100 is a parent of 19 and you can verify it through the diagram.

For node at index 3 i.e., 36,

Left child of 36 is present at index 2*i i.e., 2*3 = 6. (because i = 3).
Element at index 6 is 25 so the left child of 36 is 25.

Right child of 36 is present at index 2*i+1 i.e., 2*3+1 = 6+1 = 7. (because i = 3).
There is no Element at index 7 so the there is no right child of node 36.

Parent Node of 36 is present at index (i / 2) i.e., (3 / 2) = 1. (because i = 3).
It means parent node of 36 is present at index 1 and element at index 1 is 100, so , 100 is a parent of 36 and you can verify it through the diagram.

For node at index 4 i.e., 17,

Note: In 1-based indexing, All the nodes froms ((n/2)+1) to n - 1 are all leaf nodes, where n is the size of the array in Integer.

And in this example size of arrray = 7, so from index 7/2 + 1 to index 7-1 (last index) all nodes are leaf nodes.

It means from index 4 (7/2 = 3, 3+1 = 4) to index 6 (7-1 = 6), all nodes are leaf node.

Values from index 4 to 6 are 17, 3 and 25, so these are all leaf node and we know that leaf nodes are the nodes with no child. So there is no need to check of left and right child.

Parent Node of 17 is present at index (i / 2) i.e., (4 / 2) = 2. (because i = 4).
It means parent node of 17 is present at index 2 and element at index 2 is 19, so , 19 is a parent of 17 and you can verify it through the diagram.

Now, you can easily find the parent node of 3 and 25.

Note: In 1-based indexing, All the nodes froms n/2 + 1 to n-1 are all leaf nodes, where n is the size of the array in Integer.

In above example size of array is 7 and from index 7/2 + 1 to index 7-1 (last index) all nodes are leaf nodes.

n/2 = 7/2 = 3 and 3+1 = 4, and n-1 = 7-1 = 6, it means from index 4 to index 6, all nodes are leaf nodes.

Value from index 4 to 6 are 17, 3 and 25, so these are all leaf node and you can verify it through the above diagram.