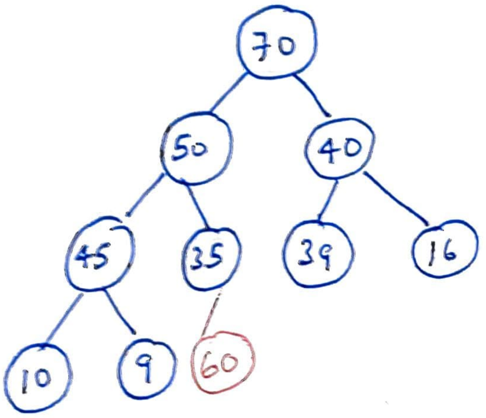
Heap → almost complete binary tree i.e (h-1) levels are completely filled and last level has all element shifted to left

Max heap → For every node i, the value of node is less than or equal to it's parent value.

$$A[\text{parent}(i)] \geq A[i]$$



1	2	3	4	5	6	7	8	9	10
70	50	40	45	35	39	16	10	9	60

↑ Array representation of binary tree

Insertion in Max_heap

- First insert a leaf node that maintains almost complete property of binary tree.
- ex. insert 60
- Now, start comparing 60 with it's parent and keep swapping until parent > 60.
- To get parent node, use array representation.
- for ex. 60 at index 10, parent = $[\frac{10}{2}] = 5$ → 35 swap(60,35), now, parent = $[\frac{i}{2}] = 2$ → 50, swap (50,60), now, parent $[\frac{2}{2}] = 1$, → 70, now stop.

Time complexity for insertion - O(log n),

because maximum we can insert an element that is larger than all the elements already there.

→ code

```
insertHeap ( Arr, n, new_value) {

        n = n + 1;
        A [n] = new_value
        int child = n
        ~~int parent = n/2~~
        int parent = 0;
       while ( child > 1) {

             parent = child / 2;
             if ( A[parent] < A[child]) {

                   swap ( A[parent], A[child]);
                   child = parent;
             }
           else {
              return
             }
       }
```

→ Delete in Max_heap

- root node is deleted
- pick last element and place it at root
- now size of max_heap is decreased by 1.
- now compare root with children until max_heap is satisfied
- in array, keep replacing parent with larger child. until end of array is reached or max heap property is satisfied.

→ Heap - Sort

Two steps
└→ (i) create max_heap
(ii) delete data one by one till array is sorted.

complete heap sort implementation →

```
void    maxheapify ( int arr[], int n, int i ) {

            int largest  =  i;
            int left  =  2xi + 1;
            int right  =  2xi+2;

            if ( left < n  &&  arr [left] > arr [largest ] ) {

                        largest  = left;
                }

            if ( right <n  &&  arr [right] > arr[largest]) {

                        largest  =  right;
                }

            if (i != largest ) {

                    swap ( arr[i], arr [largest]);

                    maxheapify ( arr, n, largest);

                }
        }

 void    buildheap ( int arr[], int n) {

        for( int i = (n/2) + 1 ; i > -1 ; i--){

                maxheapify ( arr, n, i);

            }
```

```
void heapsort ( int arr[], int n) {

        buildheap (arr, n);
        for( int i = n-1 ; i > -1 ; i--){
                swap ( arr[i], arr[0]);
                maxheapify ( arr, i, 0);
            }.
    }
```