1. **Illustrate the operation of Max\_Heapify(A, 3) using the array**

**A= {27,17,3,16,13,10,1,5,7,12,4,8,9,0}**

Max-Heapify Process

Initial Array:  
A = {27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0}

Max-Heap Property:  
Each parent node must be greater than or equal to its children.

Step-by-Step Max-Heapify at Index 3

Step 1: Start at Index 3

* Value: 3
* Left child index = 2 \* 3 = 6 → A[6] = 10
* Right child index = 2 \* 3 + 1 = 7 → A[7] = 1
* Since 10 > 3, we swap 3 and 10.

Updated Array:  
{27, 17, 10, 16, 13, 3, 1, 5, 7, 12, 4, 8, 9, 0}

Step 2: Max-Heapify at Index 6

* Value: 3
* Left child index = 2 \* 6 = 12 → A[12] = 8
* Right child index = 2 \* 6 + 1 = 13 → A[13] = 9
* Since 9 > 3, we swap 3 and 9.

Updated Array:  
{27, 17, 10, 16, 13, 9, 1, 5, 7, 12, 4, 8, 3, 0}

Step 3: Max-Heapify at Index 13

* Index 13 has no children
* Heapify complete for this branch.

Final Max-Heapified Array:

{27, 17, 10, 16, 13, 9, 1, 5, 7, 12, 4, 8, 3, 0}

1. **Show the operation of Build-Heap using the array below:**

**A= {5,3,17,10,84,19,6,22,9}**

5

/ \

3 17

/ \ / \

10 84 19 6

/ \

22 9

We start from the last non-leaf node, which is at index n/2 = 9/2 = 4.  
We apply Max-Heapify from index 4 down to 1.

Step 1: Max-Heapify at index 4

* Value: 84
* Left child (index 8) = 22
* Right child (index 9) = 9
* 84 is already the largest, no swap needed.

Array remains:  
{5, 3, 17, 10, 84, 19, 6, 22, 9}

Step 2: Max-Heapify at index 3

* Value: 17
* Left child (index 6) = 19
* Right child (index 7) = 6
* 19 > 17 → Swap index 3 and 6

Updated Array:  
{5, 3, 19, 10, 84, 17, 6, 22, 9}

Step 3: Max-Heapify at index 2

* Value: 3
* Left child (index 4) = 84
* Right child (index 5) = 17
* 84 > 3 → Swap index 2 and 4

Updated Array:  
{5, 84, 19, 3, 17, 10, 6, 22, 9}

Step 4: Max-Heapify at index 1

* Value: 5
* Left child (index 2) = 84
* Right child (index 3) = 19
* 84 > 5 → Swap index 1 and 2

Final Array:  
{84, 5, 19, 22, 3, 17, 6, 10, 9}

Final Max-Heap:

{84, 5, 19, 22, 3, 17, 6, 10, 9}

1. **Illustrate the operation of Heapsort using the array below:**

**{5,13,2,25,7,17,20,8,4}**

Initial Array:  
A = {5, 13, 2, 25, 7, 17, 20, 8, 4}

Heap Sort Involves Two Steps:

1. Build a Max Heap
2. Sort the array by repeatedly swapping the max element with the last element and reducing the heap size.

Step 1: Build Max Heap

* Last non-leaf node = ⌊n/2⌋ = ⌊9/2⌋ = 4

Start heapifying from index 4 to index 1:

At index 4:

* Left = index 8 → 8
* Right = index 9 → 4
* 25 > both children → no swap

Array remains:  
{5, 13, 2, 25, 7, 17, 20, 8, 4}

At index 3:

* Value = 2
* Left = 17 (index 6), Right = 20 (index 7)
* 20 is largest → swap 2 and 20

Array:  
{5, 13, 20, 25, 7, 17, 2, 8, 4}

At index 2:

* Value = 13
* Left = 25 (index 4), Right = 7 (index 5)
* 25 > 13 → swap 13 and 25
* Then heapify 13 (now at index 4) with children 8 and 4 → swap with 8

Array:  
{5, 25, 20, 8, 7, 17, 2, 13, 4}

At index 1:

* Value = 5
* Left = 25 (index 2), Right = 20 (index 3)
* 25 is largest → swap 5 and 25
* Then heapify 5 at index 2 with 8 and 7 → swap with 8

Final Max-Heap:  
{25, 8, 20, 5, 7, 17, 2, 13, 4}

Step 2: Sorting the Array (Heap Sort Process)

Repeat the following:

* Swap the first element (max) with the last element
* Reduce heap size
* Apply Max-Heapify at root

Iterative Sorting:

1. Swap 25 and 4 → Heap size = 8  
   {4, 8, 20, 5, 7, 17, 2, 13, 25}
2. Max-Heapify → 20 is max → move it to top  
   {20, 8, 4, 5, 7, 17, 2, 13, 25}
3. Swap 20 and 13 → Heap size = 7  
   {13, 8, 4, 5, 7, 17, 2, 20, 25}
4. Max-Heapify → 17 is max  
   {17, 8, 4, 5, 7, 13, 2, 20, 25}
5. Swap 17 and 2 → Heap size = 6  
   {2, 8, 4, 5, 7, 13, 17, 20, 25}
6. Max-Heapify → 8 is max  
   {8, 2, 4, 5, 7, 13, 17, 20, 25}
7. Swap 8 and 7 → Heap size = 5  
   {7, 2, 4, 5, 8, 13, 17, 20, 25}
8. Max-Heapify → 7 is already max  
   {7, 2, 4, 5, 8, 13, 17, 20, 25}
9. Swap 7 and 5 → Heap size = 4  
   {5, 2, 4, 7, 8, 13, 17, 20, 25}
10. Max-Heapify → 5 is max  
    {5, 2, 4, 7, 8, 13, 17, 20, 25}
11. Swap 5 and 4 → Heap size = 3  
    {4, 2, 5, 7, 8, 13, 17, 20, 25}
12. Max-Heapify → 4 is max  
    {4, 2, 5, 7, 8, 13, 17, 20, 25}
13. Swap 4 and 2 → Heap size = 2  
    {2, 4, 5, 7, 8, 13, 17, 20, 25}

Final Sorted Array:

{2, 4, 5, 7, 8, 13, 17, 20, 25}

1. **Show that in an n-element heap, there are at most [n/2h+1] nodes of height h.**

Proof: Number of Nodes at Each Height in a Binary Tree

Base Case

* At height h = 0 (leaf level), the number of nodes is at most n/2
* This is true because in a binary tree, at least half of all nodes must be leaves

Inductive Hypothesis

* Assume that at height h, the number of nodes is at most n/2^(h+1)

Inductive Step

* For height h+1, each node must have come from the level below (height h)
* Since each node at height h+1 can have at most 2 children at height h
* The number of nodes at height h+1 is at most (nodes at height h)/2
* By our inductive hypothesis: nodes at height h+1 ≤ (n/2^(h+1))/2 = n/2^(h+2)

Conclusion

* Therefore, by mathematical induction, at any height h, the number of nodes is at most n/2^(h+1)