**COMSATS Institute of Information Technology, Islamabad Campus**

**Department of Computer Science**

**Design and Analysis of Algorithms – CSC301**

**BCS – IV-A**



**Assignment #2**

**FA19-BCS-013 | FA19-BCS-033 | FA19-BCS-034 | FA19-BCS-037**

**Q1:**

1. **How *Heap Sort* works? Provide a dry run.**

**Heap Sort:**

*“Heap sort is a comparison-based sorting method that sorts objects using the Binary Heap data structure.”*

**Binary Heap:**

A Binary Heap is a Complete Binary Tree in which objects are stored in such a way that the value of a parent node is greater (or smaller) than the values of its two children’s nodes.

**Min Heap:**

Complete Binary Tree in which value of parent node is smaller than values of its children.

**Max Heap:**

Complete Binary Tree in which value of parent node is greater than values of its children.

**Mechanism:**

It's similar to selection sorting, in which we find the largest/Smallest element first and position it at the top depending on whether it is maxed heap or min-heap. The process is repeated for the remaining components.

**Heapify:**

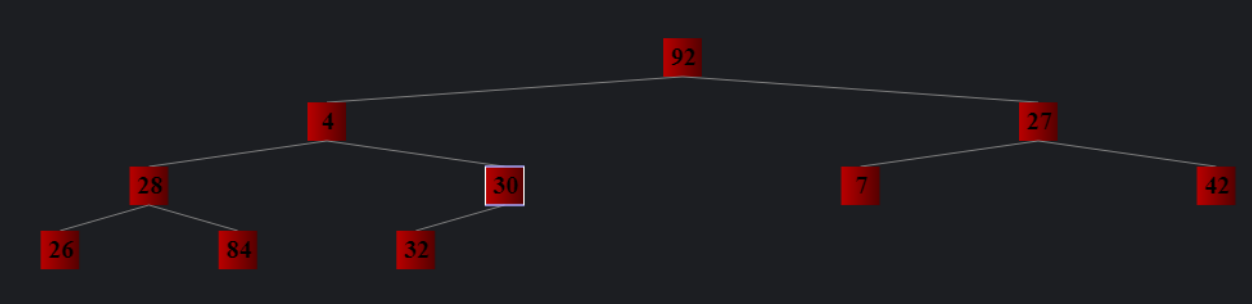
Procedure to convert a binary tree into Heap. It uses a bottom-up order to ensure the heapification of the array whether Max heap or Min heap.

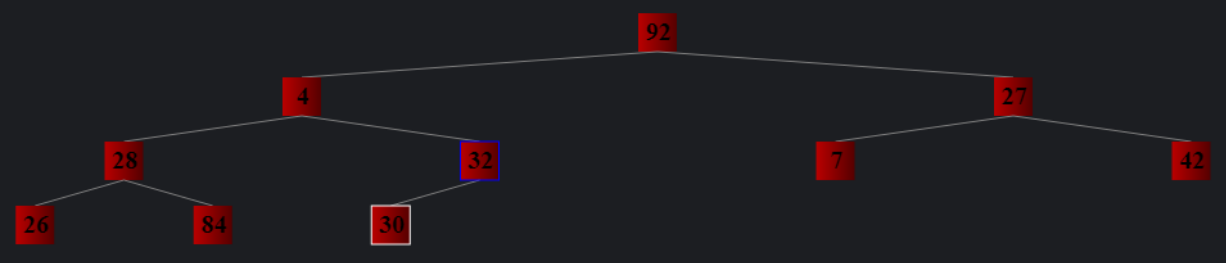
**Steps:**

1. Create a max heap using the input data.
2. The largest item is placed at the top of the heap at this stage. Replace it with the heap's last object, then reduce the heap's size by one. Finally, heapify the tree's base.
3. Phase 2 should be repeated as long as the size of the heap is greater than one.

**Dry Run:**

****

****

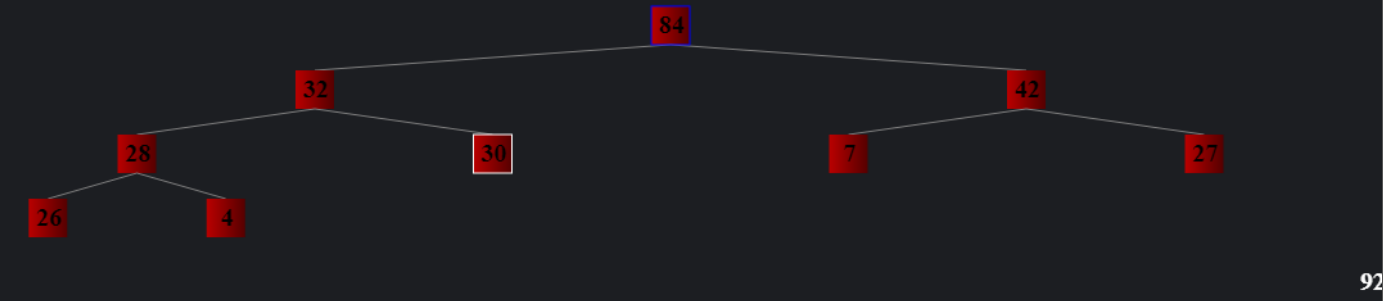
****

****

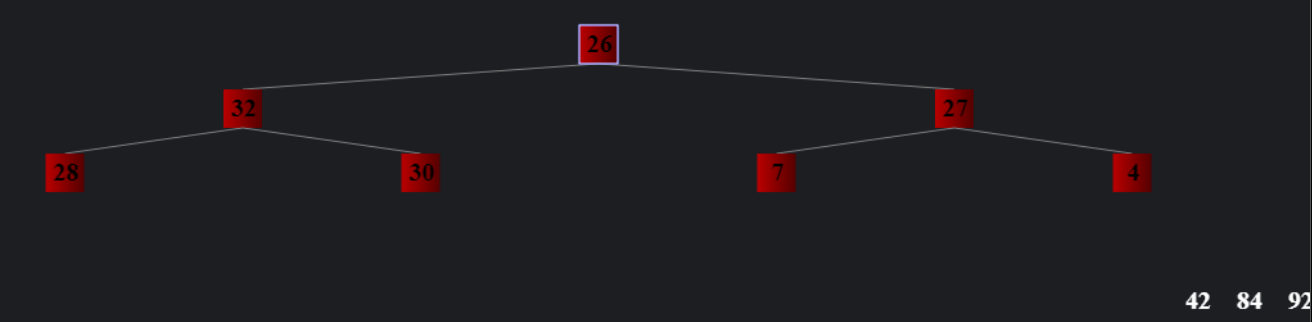
****

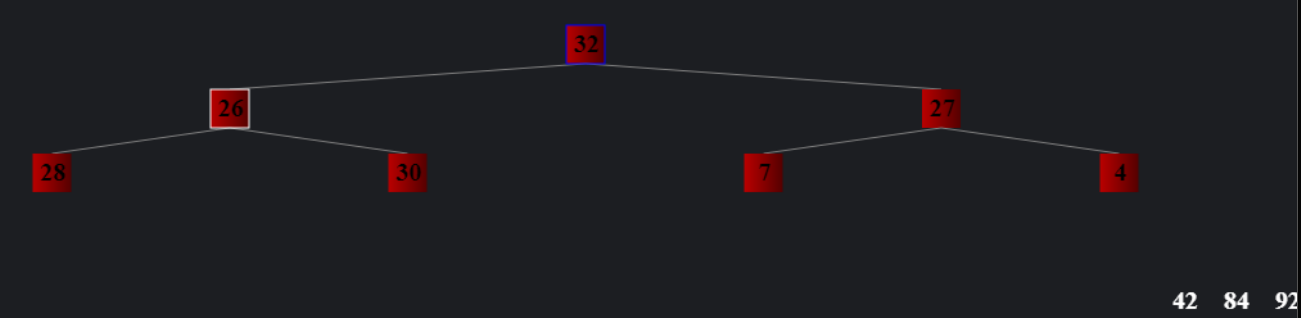
****

****

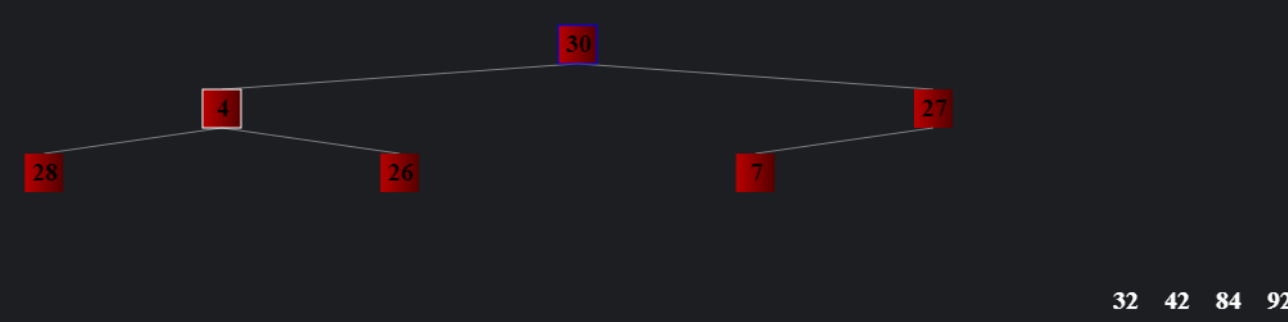
****

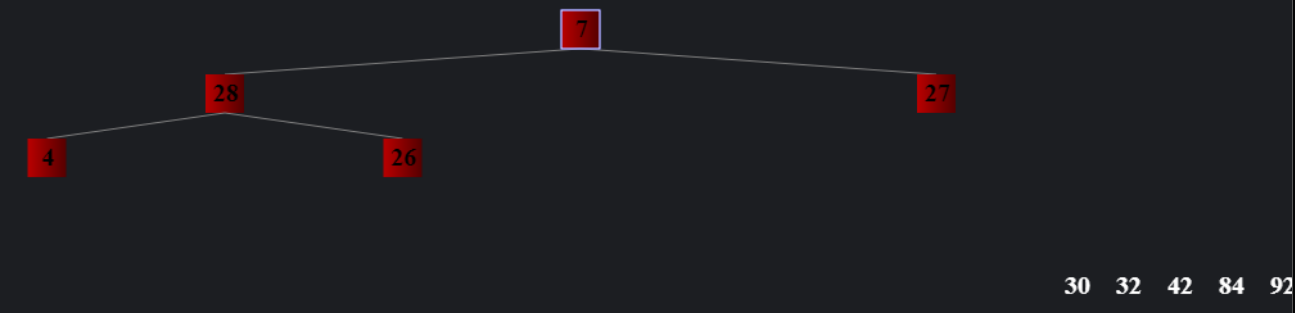
****

****

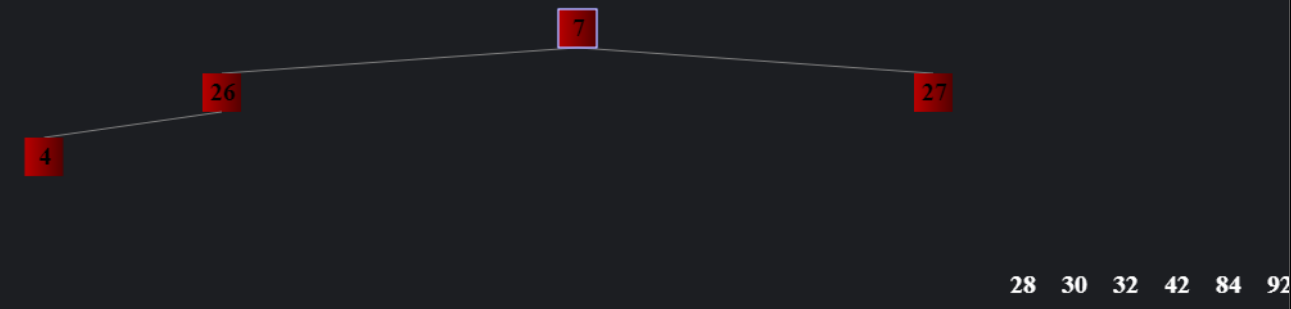
****

****

****

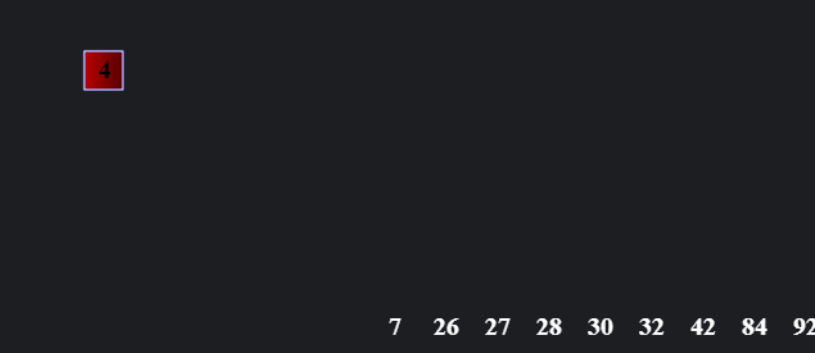
****

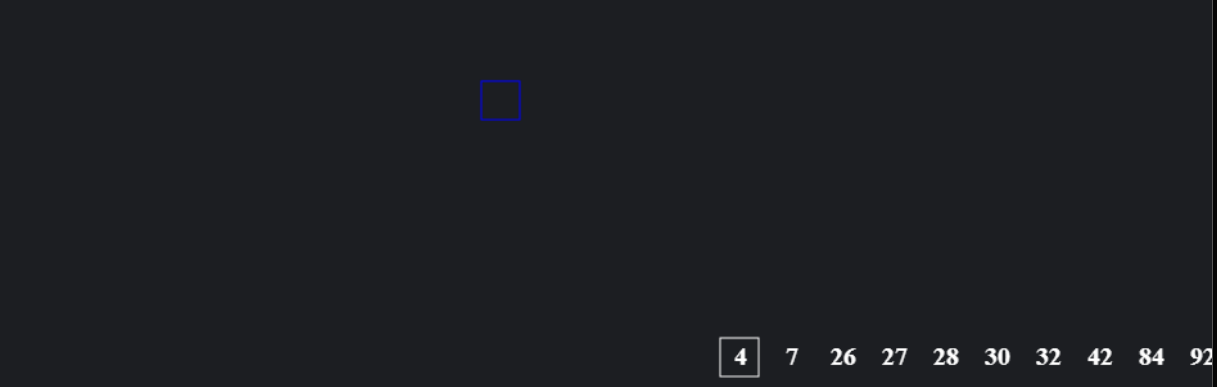
****



****

****

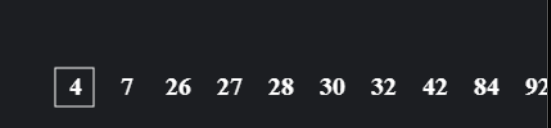
****

****

**Start Array:**

****

**Sorted Array:**

****

1. **Compare the time complexities of *Merge Sort* and *Heap Sort*. Discuss all the three cases (best, average, and worst).**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sorting Algorithm** |  | **Time Complexity** |  | **Space Complexity** |
| **Best** | **Average** | **Worst** |
| **Merge Sort** | O (N log n) | O (N log n) | O (N log n) | O (n) |
| **Heap Sort** | O (N log n) | O (N log n) | O (N log n) | O (1) |

**Heapsort:**

Time complexity to heapify the array is O(Logn) meanwhile the time complexity to create and build Heap is O(n) so the overall time complexity of Heap Sort is O (n Logn).

**Merge Sort:**

The cost to find the middle of any subarray is O (1). The cost to divide the array in half in every step would be O (log n +1) [max]. The cost to merge the sub-array would take 0 (n)

So, the Total Cost would be O (n log n).

Merge Sort's time complexity is O (n\*log n) in all three cases (worst, average, and best) since it always splits the array into two halves and merges the two halves in linear time.

1. **Solve the following recurrences using the recursion tree and master method.**

* ***T*(*n*) = 3*T*(⎣*n*/4⎦) + Θ(*n*2).**

# **Recursion Tree:**

n2

O(n/4)2

O(n/16)2

O(n/16)2

O(n/16)2

O(n/16)2

O(n/16)2

O(n/16)2

O(n/4)2

O(n/16)2

O(n/16)2

O(n/16)2

O(n/4)2

T(1) T(1) T(1) T(1) T(1) T(1) T(1) T(1)………………… T(1) T(1) T(1) T(1) ----------O(nlog43)

* n2 is the root node
* Nodes on the second layer have cost of O(n/4)2
* At the ith level the sub problem size is (n/4)i
* i = log4 n. So the tree has log4 n + 1 levels

Total Cost of the Tree:

**T(n)= cn2 + Θ(n log4 3 )**

Since functions in Θ (n log4 3) are also in O (n 2 ), this whole expression is O ( n 2 ).

Thus the cost is:

**T(n) = O (n 2)**

# **Master Theorem:**

Here,

a = 3, b = 4 and f(n) = n2

This implies,

= 0.79

n r = n log 43

f(n) = Ω(nlog43+ε) …………….*Case 3*

Thus, T(n) = O(f(n))

**T(n) = O(n2)**

* *T*(*n*) = *T*(*n*/4) + *T*(*n*/2) + *O*(*n2*).

# **Recursion Tree:**

**………. n2**

**n2**

**………. n2**

(n/2)2

(n/4)2

**………. n2**

(n/4)2

(n/8)2

(n/8)2

(n/16)2

O (1)

Total Cost = n2 + n2 + n2 + …..

T (n) = O (n2) ……. Geometric Series

# **Master Theorem:**

Here,

a = 1, b = 2 and f(n) = n2

This implies,

= 0

f(n) = Ω(nlog43+ε) …………….*Case 3*

Thus, T(n) = O(f(n))

**T(n) = O(n2)**