**Heap**

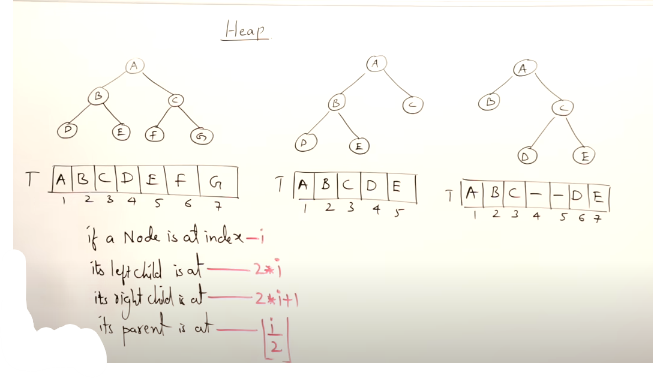
|  |  |  |  |
| --- | --- | --- | --- |
| **Easy Interview Questions on Heap Data Structure** | | | |
| 1. Kth smallest element in an array |  | 1. Height of a complete binary tree (or Heap) with N nodes |  |
| 1. Minimum product of k integers in an array of positive Integers |  | 1. Minimum sum of two numbers formed from digits of an array |  |
| 1. Sort an Almost Sorted Array |  | 1. Kth smallest element in a row-wise and column-wise sorted 2D array |  |
| 1. Top K Frequent Elements |  | 1. Sum of all elements between k1’th and k2’th smallest elements |  |
| **Medium Interview Questions on Heap Data Structure** | | | |
| 1. Implement a Min Heap |  | 1. Merge two binary max heaps |  |
| 1. Implement a Max Heap |  | 1. Find k closest numbers |  |
| 1. Heap Sort |  | 1. Sort an almost sorted array |  |
| 1. Convert max heap to min heap |  | 1. K maximum sum combinations from two arrays |  |
| 1. Convert min Heap to max Heap |  | 1. BST to max heap |  |
| 1. Check if a Binary Tree is a Min Heap |  | 1. Convert BST to Min Heap |  |
| 1. Check if a Binary Tree is a Max Heap |  | 1. K’th largest element in a stream |  |
| 1. Binary Heap |  | 1. Find k numbers with most occurrences in the given array |  |
| 1. Given level order traversal of a Binary |  | 1. Find the kth largest element in an array |  |
| 1. Tree, check if the Tree is a Min-Heap |  | 1. Merge overlapping intervals |  |
| 1. Implement a priority queue |  | 1. Game with String |  |
| 1. Heap Sort for decreasing order using min heap |  | 1. Maximize The Array |  |
| 1. Find kth smallest element in a row-column sorted matrix |  | 1. Rearrange characters |  |
| 1. Largest triplet product in a stream |  | 1. Minimum sum of squares of character counts in a given string after removing k characters |  |
| 1. Connect n ropes with minimum cost |  | 1. Maximum sum of at most two non-overlapping intervals in a list of Intervals |  |
| 1. Merge two binary max heaps |  | 1. K-th Largest Sum Contiguous Subarray |  |
| **Hard Interview Questions on Heap Data Structure** | | | |
| 1. Merge k sorted arrays |  | 1. Minimum cost to connect all cities |  |
| 1. Merge k Sorted Lists |  | 1. Single-Source Shortest Paths – Dijkstra’s Algorithm |  |
| 1. Find the median of a stream of running integers |  | 1. Sliding Window Maximum (Maximum of all subarrays of size K) |  |
| 1. Smallest range in K lists |  | 1. K maximum sum combinations from two arrays |  |
| 1. Huffman Encoding |  | 1. Merge two sorted arrays in O(1) extra space using Heap |  |

|  |
| --- |
| Link: <https://www.geeksforgeeks.org/top-50-problems-on-heap-data-structure-asked-in-interviews/> |

Subtopic of head

1. Array representation of Binary tree
2. Complete Binary tree
3. Heap
4. Insert and Delete
5. Heap sort
6. Heapify
7. Priority Queue

Representation of binary tree using array



Full Binary tree and Full complete binary tree

Full binary tree can have maximum nodes

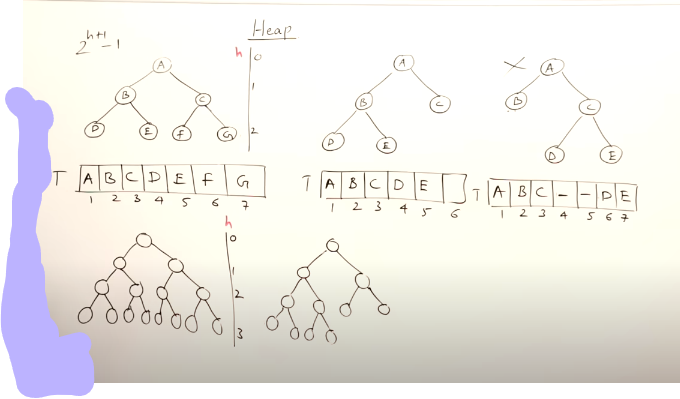
1st to last element of array for binary tree if there is **no missing** element in between then it’s a **complete binary tree**

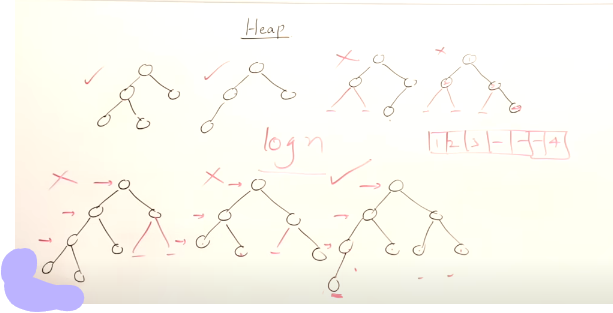
1st to last element of array for binary tree if there is a **single missing** element in between nodes then it’s a **binary tree**

Every **full binary tree** is a **complete binary tree**

**Complete binary tree** element are filled from **left to right**

Height of a **complete binary tree** will always be (minimum)



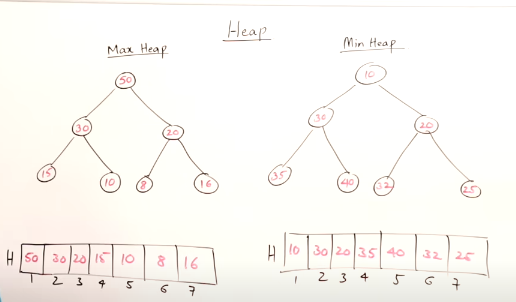


Heap

Heap is a complete binary tree heap types:

Max heap 🡪 duplicate elements are allowed (root is bigger than all of its decedents)

Min heap 🡪 (root is smaller than all of its decedents)



Insert operation in max heap

Insertion is not performed at roots

Set the new element as leave then compare

**New node** will be compared with all of its **decedents**

Then new node will move forward until it reaches its place

Time complexity of a complete binary tree **minimum O(1)** and **maximum** (time taken for insertion)

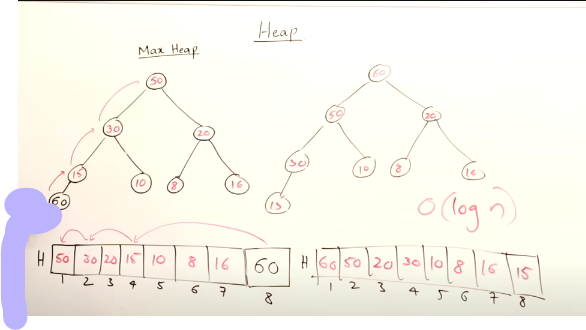
Swapes are depend on height

Elements moves **leaves towards roots** so the directions of adjustment is **upwards**

New compare with roots

New compare with childern

Adjust upwards leaves towards roots



Delete operation on max heap

Only root element is deleted and last element replace that’s place

Top heap = max heap

New compare with roots

New compare with children

Adjust downwards roots towards leaves

Deletion depends on height

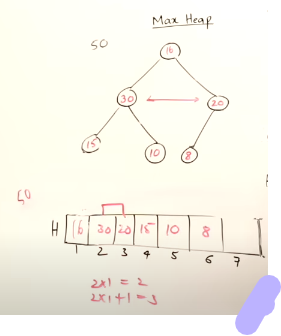
Max time:

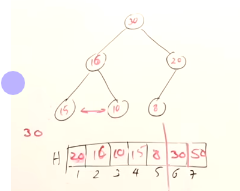
Min time:

Max head – deletion – you get next --largest element

Min head – deletion – you get next --smallest element

It gets sorted





Heap sort

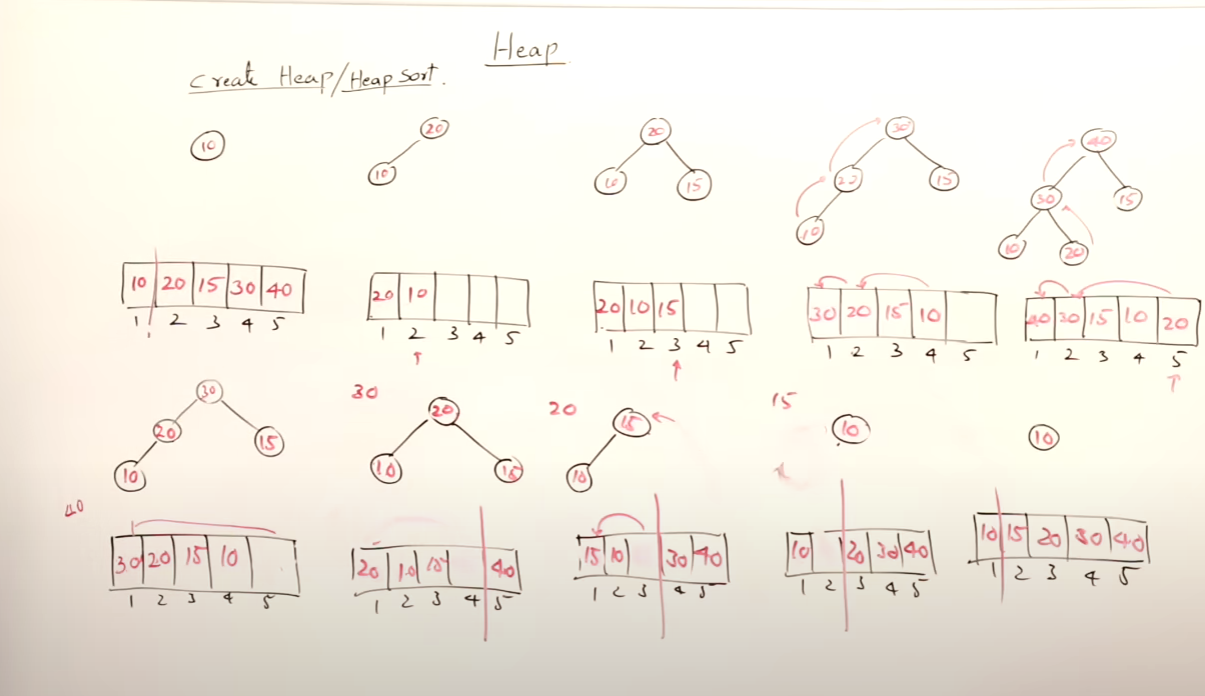
It has two steps

1. For given set of elements create a heap by inserting all the element one by one
2. One the heap is formed delete all the element from heap one by one the element will get sorted

Time complexity for insertion: **nlogn**

**1st step:**

2nd step



Time complexity for deletion: **nlogn**

Total time : **O(nlogn)**

Heapify