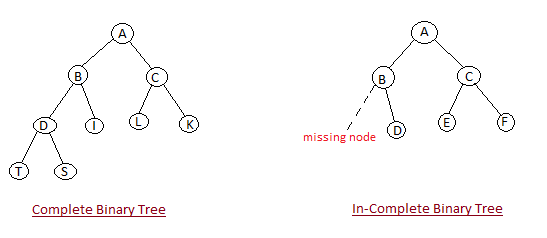
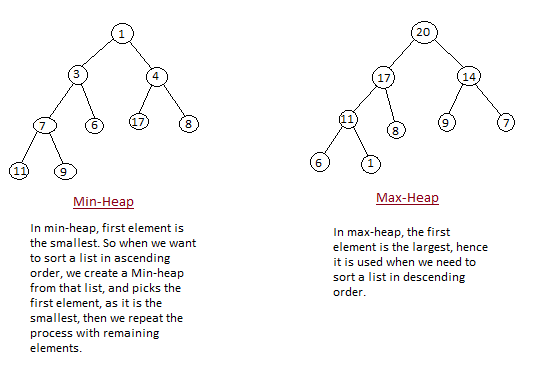
#### **What is a Heap ?**

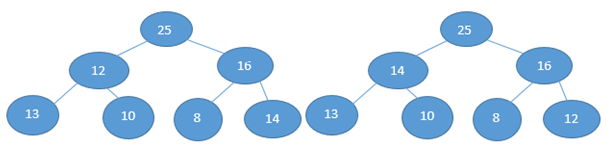
Heap is a special tree-based data structure that satisfies the following special heap properties:

1. **Shape Property:** Heap data structure is always a Complete Binary Tree, which means all levels of the tree are fully filled.



1. **Heap Property:** All nodes are either *[greater than or equal to]* or *[less than or equal to]* each of its children. If the parent nodes are greater than their children, heap is called a **Max-Heap**, and if the parent nodes are smaller than their child nodes, heap is called **Min-Heap**.





|  |  |
| --- | --- |
| It’s not a max heap as every node does not have a greater value Than its child node. Here 13 is leaf node and its parent node is having 12 so this is not a max heap. And the heap size will be 1. As only one node is following the max heap property Here that is 25. Node 16 is also following the max heap criteria but we won’t count it, once a node is failed the max heap criteria then there after we won’t consider the node for heap size. | It’s a max heap as every node have a greater value Than its child node. And the heap size will be 7 that is length of the array. As here all node is following the max heap property. **Leaf node is always a heap node.** |

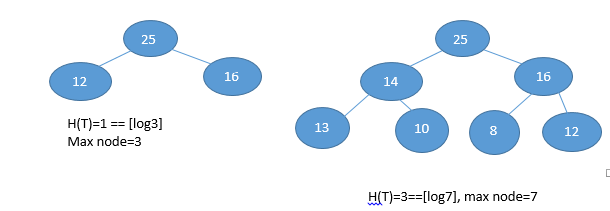
Heap is a complete binary tree because if it’s not a complete binary tree then there will be gap while putting the heap data into the array. As we take the heap data in index wise and put that data in array at the same index as it is in heap. Means in heap some data is at index 3 then in array also it should be at index 3. So if Heap is not a complete binary tree then we won’t find the successive index to put the data into the array so there will be some index gap in array on which index there is nothing in the heap. That’s why Heap is a complete binary tree.

**If the array is in ascending order then it is min heap, and if the array is in descending order then it is max heap.**

**For max heap sort time complexity is O(nlogn) like merge sort.**

**If there is purpose to just a construct a heap then their time complexity will be O(n).**

**Height of a tree is maximum no of edges from that node to a leaf node. Height of the tree is equal to height of the root node.**

****

**So, the relation between no of node and height of the tree for complete binary tree is No of**

**node=2 h+1-1**

**And the height of the tree by using the node, H(T)=[logn], where n=no of nodes (its applicable for both complete binary tree as well as un-complete binary tree)**

**If there are a complete binary tree with n nodes, then leaf nodes will be starting from [n/2] to n.**

**After finding the leaf node, find the largest index non-leaf node and heapify them, means check the child node of the largest index non-leaf node and make the largest value as root node.**

**Suppose there is an array 1, 5, 6, 8, 12, 14, 16**

|  |  |
| --- | --- |
| Here leaf node will be [7/2] to 6th index.  Means 3rd index to 6th index there is leaf node.    Now we will find the largest non-leaf node.  That is 2nd index which has value 6.  Now we will check the child element of 2nd Node that is 14 and 16. So we will swap the Value of 6 with the maximum of its child element that is 16.  After swapping the value it will look like bellow. |  |
| Now till 2,3,4,5,6 index arrays is max heap.  Now we will find the second largest non-leaf node. That is 1st index node which has value 5. Now we will check the child element of 1st Node that is 8 and 12. So we will swap 5 with maximum of 8 and 12. |  |

|  |
| --- |
|  |

|  |  |  |
| --- | --- | --- |
| After swapping the value 5 and 12 it will look like this.  So till here 1,2,3,4,5,6th index it is max heap.  Now we will find the 3rd largest non-leaf node that is root node here and its value is 1.  Now we will find the child element of the root node That is 12 and 16. Now we will swap 1 with maximum of 12 and 16. |  | |
| After swapping the value 1 and 16 it will look like this.  Once we swapped the 1 with 16, 2nd index node that has value 1 is now not a max heap.  So we will check its child element that is 14 and 6. So we will swap 1 with maximum of 14 and 6.  After swapping 1 with 14 it will look  Like bellow.  Here till 0,1,2,3,4,5,6 it’s a max heap.  Now it’s a max heap. enjoy ☺  **If height of the tree is h, then no of nodes will be upper limit of [n/2] h +1.**  **If the height of tree is h then we can find that at any height how many maximum no of nodes are present.** | |  | |

Heap size will be the length of the array till it satisfies the heap condition. Suppose there is an array which doesn’t satisfies the heap condition then the heap size will be zero.

And all the leaf node is heap by default.

**Heap sort**

**Worst Case Time Complexity :** O(n log n)

**Best Case Time Complexity :** O(n log n)

**Average Time Complexity :** O(n log n)

**Space Complexity :** O(n)

* Heap sort is not a Stable sort, and requires a constant space for sorting a list.
* Heap Sort is very fast and is widely used for sorting.

## Heap Sort Algorithm

Heap Sort is one of the best sorting methods being in-place and with no quadratic worst-case scenarios. Heap sort algorithm is divided into two basic parts:

* Creating a Heap of the unsorted list.
* Then a sorted array is created by repeatedly removing the largest/smallest element from the heap, and inserting it into the array. The heap is reconstructed after each removal.

**Identification of problems on Heap**

* 1. Largest/smallest we have to finf
  2. K will be given

1. **Kth Smallest Element**

**int**[] nums = {3, 2, 1, 5, 6, 4};

**int** k = 2;

Here we have to find out the 3rd smallest element from the array.

**Algo:**

As we have to find out the k smallest element, we have to take Max heap.

Add all the element one by one into heap and check the queue size. If queue size is greater than k then poll the element from the queue. At last there will be only 3 element in the queue and then peek the top element from the queue.

<https://github.com/hareramcse/Datastructure/blob/master/Algorithm/src/com/hs/heap/KthLargestElement.java>

1. **Return K largest element in the array**

**int** arr[] = { 11, 3, 2, 1, 15, 5, 4, 45, 88, 96, 50, 45 };

**int** k = 4;

Algo:

Here we have to use Min heap as in question it is asked to get the K largest element in the array.

We will just add the element of the array into heap and will check the size of the heap. If size> k then we will pop up the element. Once loop will over in the heap there will be k element which will be the k largest element of the array.

<https://github.com/hareramcse/Datastructure/blob/master/Algorithm/src/com/hs/heap/KLargestElement.java>

1. **Sort an almost sorted order**

**int**[] arr = { 2, 6, 3, 12, 56, 8 };

**int** k = 3;

**Algo:**

Here K is given it means in sorted array ith element is placed in i+kth or i-kth index in this array

We will take min heap and store the element one by one and check the heap size. Once heap size > k the we will pop the element from the heap and store in the array.

Once loop is over k largest element will be there in the heap. Then we will iterate the heap and append this element from the heap into array.

<https://github.com/hareramcse/Datastructure/blob/master/Algorithm/src/com/hs/heap/SortAnAlmostSortedArray.java>

1. **K closest number from origin**

**int** arr[] = { 11, 3, 2, 1, 15, 5, 4, 45, 88, 96, 50, 45 };

**int** k = 4;

Here we have to find out the 4 closest number from 4

**Algo:**

Here we create a Pair of key ad value. Key will be absolute difference between arr[i] – k and value will be i

We will store this pair in the heap and f size of the heap > k then poll the pair from the heap.

1. **K top frequent elements in the array**

**int**[] nums = { 1, 1, 1, 2, 2, 3 };

**int** k = 2;

we have to find out k most frequent element in the array.

Algo:

Here we will take map and store the array elements and its count. Then we will add map data in heap of Pair. we add map value as key and map key as value in pair. So that in the heap we will get most frequent element on top of heap. Then iterate the heap k times and poll the data and store into result.

<https://github.com/hareramcse/Datastructure/blob/master/Algorithm/src/com/hs/heap/KMostFrequentElement.java>

1. **Frequency Sort**

**int**[] arr = { 1, 1, 2, 2, 2, 3 };

Here we have to sort the element based on the frequency in increasing order

Algo:

Take a Map<Integer, Integer> map and store the element and its frequency.

Take a Queue<Pair> minheap and store the data into it from map. In pair key will be value and value will be key.

While heap is not empty poll the data from the heap and store it in array and return it.

<https://github.com/hareramcse/Datastructure/blob/master/Algorithm/src/com/hs/heap/FrequencySort.java>

1. **K closest point to origin**

**int**[][] arr = { { 1, 3 }, { -2, 2 } };

**int** k = 1;

Here we have to find 1 co-ordinate which is nearest to origin

Algo:

Take a maxheap of type Pair

Store the distance of the coordinates into the maxheap as key and value will be i

Once heap size greater than k then poll the pair from heap.

Now heap will contain the result. Iterate the heap and from the pair get the value element.

<https://github.com/hareramcse/Datastructure/blob/master/Algorithm/src/com/hs/heap/KClosestPointToOrigin.java>

1. **Connect n ropes with minimum cost**

**int** len[] = { 4, 3, 2, 6 };

we have to find out the minimum cost to connect all the ropes.

**Algo:**

Take a minheap and add all the array element into heap. Then in the while lop poll 2 min elements and add the values and store into result. Once we get the summation of the these polled element add this to heap again.

Return the result.

<https://github.com/hareramcse/Datastructure/blob/master/Algorithm/src/com/hs/heap/ConectNRopesWithMinimumCost.java>