Heap sort

**Heap sort** is a comparison-based sorting technique based on [Binary Heap](http://www.geeksforgeeks.org/binary-heap/) data structure. It is similar to the [selection sort](http://www.geeksforgeeks.org/selection-sort/) where we first find the minimum element and place the minimum element at the beginning. Repeat the same process for the remaining elements.

* Heap sort is an in-place algorithm.
* Its typical implementation is not stable, but can be made stable (See [this](https://www.geeksforgeeks.org/stability-in-sorting-algorithms/))
* Typically 2-3 times slower than well-implemented [QuickSort](http://www.geeksforgeeks.org/quick-sort/). The reason for slowness is a lack of locality of reference.

### **Advantages of heapsort:**

* **Efficiency –**  The time required to perform Heap sort increases logarithmically while other algorithms may grow exponentially slower as the number of items to sort increases. This sorting algorithm is very efficient.
* **Memory Usage –** Memory usage is minimal because apart from what is necessary to hold the initial list of items to be sorted, it needs no additional memory space to work
* **Simplicity –**  It is simpler to understand than other equally efficient sorting algorithms because it does not use advanced computer science concepts such as recursion.

### **Disadvantages of Heap Sort:**

* **Costly**: Heap sort is costly.
* **Unstable**: Heat sort is unstable. It might rearrange the relative order.
* **Efficient:** Heap Sort are not very efficient when working with highly complex data.

### **Applications of HeapSort:**

* Heapsort is mainly used in hybrid algorithms like the [IntroSort](https://www.geeksforgeeks.org/introsort-or-introspective-sort/).
* [Sort a nearly sorted (or K sorted) array](https://www.geeksforgeeks.org/nearly-sorted-algorithm/)
* [k largest(or smallest) elements in an array](https://www.geeksforgeeks.org/k-largestor-smallest-elements-in-an-array/)

The heap sort algorithm has limited uses because Quicksort and Mergesort are better in practice. Nevertheless, the Heap data structure itself is enormously used. See [Applications of Heap Data Structure](https://www.geeksforgeeks.org/applications-of-heap-data-structure/)

Recommended Problem

Heap Sort

[Heap](https://practice.geeksforgeeks.org/explore?page=1&category%5B%5D=Heap&sortBy=submissions)

[Sorting](https://practice.geeksforgeeks.org/explore?page=1&category%5B%5D=Sorting&sortBy=submissions)

+2 more

[24\*7 Innovation Labs](https://practice.geeksforgeeks.org/explore?page=1&company%5B%5D=24*7%20Innovation%20Labs&sortBy=submissions)

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5B%5D=Amazon&sortBy=submissions)

+7 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/heap-sort/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 68.4K

## **What is meant by Heapify?**

Heapify is the process of creating a heap data structure from a binary tree represented using an array. It is used to create Min-Heap or Max-heap. Start from the first index of the non-leaf node whose index is given by n/2 – 1. Heapify uses recursion.

### **Algorithm for Heapify:**

heapify(array)

Root = array[0]

Largest = largest( array[0] , array [2 \* 0 + 1]/ array[2 \* 0 + 2])

if(Root != Largest)

Swap(Root, Largest)

### **How does Heapify work?**

### 

Array = {1, 3, 5, 4, 6, 13, 10, 9, 8, 15, 17}

Corresponding Complete Binary Tree is:

1

/ \

3 5

/ \ / \

4 6 13 10

/ \ / \

9 8 15 17



**The task to build a Max-Heap from above array**.

Total Nodes = 11.

Total non-leaf nodes= (11/2)-1=5

last non-leaf node = 6.

Therefore, Last Non-leaf node index = 4.

To build the heap, heapify only the nodes: [1, 3, 5, 4, 6] in reverse order.

**Heapify 6**: Swap 6 and 17.

1

/ \

3 5

/ \ / \

4 17 13 10

/ \ / \

9 8 15 6

**Heapify 4**: Swap 4 and 9.

1

/ \

3 5

/ \ / \

9 17 13 10

/ \ / \

4 8 15 6

**Heapify 5**: Swap 13 and 5.

1

/ \

3 13

/ \ / \

9 17 5 10

/ \ / \

4 8 15 6

**Heapify 3**: First Swap 3 and 17, again swap 3 and 15.

1

/ \

17 13

/ \ / \

9 15 5 10

/ \ / \

4 8 3 6

**Heapify 1**: First Swap 1 and 17, again swap 1 and 15, finally swap 1 and 6.

17

/ \

15 13

/ \ / \

9 6 5 10

/ \ / \

4 8 3 1

## **Heap Sort Algorithm**

To solve the problem follow the below idea:

First convert the array into heap data structure using heapify, then one by one delete the root node of the Max-heap and replace it with the last node in the heap and then heapify the root of the heap. Repeat this process until size of heap is greater than 1.

Follow the given steps to solve the problem:

* Build a max heap from the input data.
* At this point, the maximum element is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of the heap by 1. Finally, heapify the root of the tree.
* Repeat step 2 while the size of the heap is greater than 1.

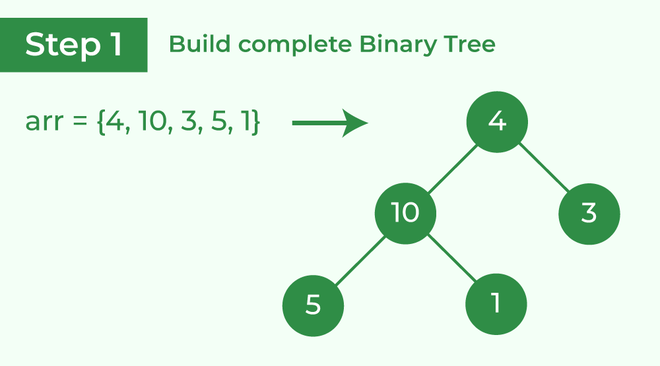
**Note:** The heapify procedure can only be applied to a node if its children nodes are heapified. So the heapification must be performed in the bottom-up order.

## **Detailed Working of Heap Sort**

To understand heap sort more clearly, let’s take an unsorted array and try to sort it using heap sort.

Consider the array: arr[] = {4, 10, 3, 5, 1}.

**Build Complete Binary Tree:** Build a complete binary tree from the array.



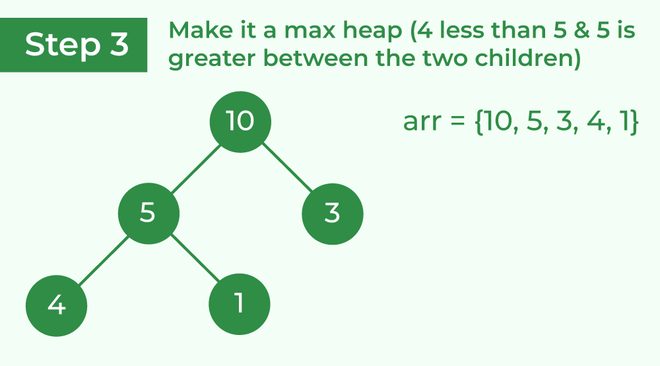
Build complete binary tree from the array

**Transform into max heap:** After that, the task is to construct a tree from that unsorted array and try to convert it into [max heap.](https://www.geeksforgeeks.org/difference-between-min-heap-and-max-heap/)

* To transform a heap into a max-heap, the parent node should always be greater than or equal to the child nodes
  + Here, in this example, as the parent node **4** is smaller than the child node **10,** thus, swap them to build a max-heap.

Transform it into a max heap image widget

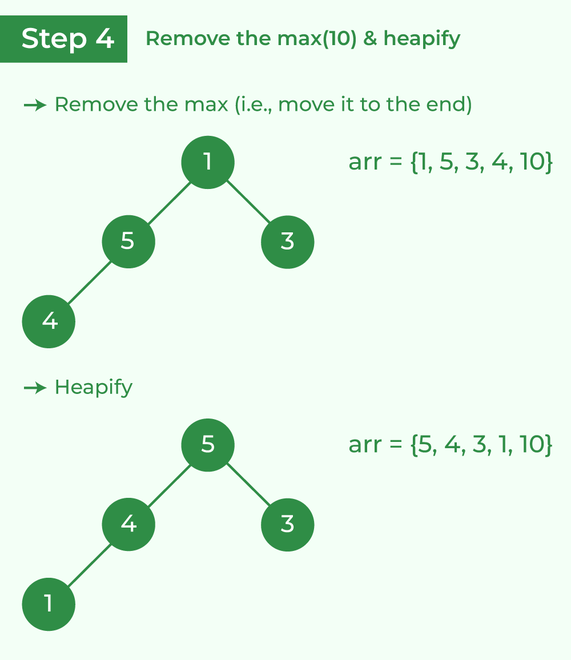
* Now, as seen, **4** as a parent is smaller than the child **5**, thus swap both of these again and the resulted heap and array should be like this:



Make the tree a max heap

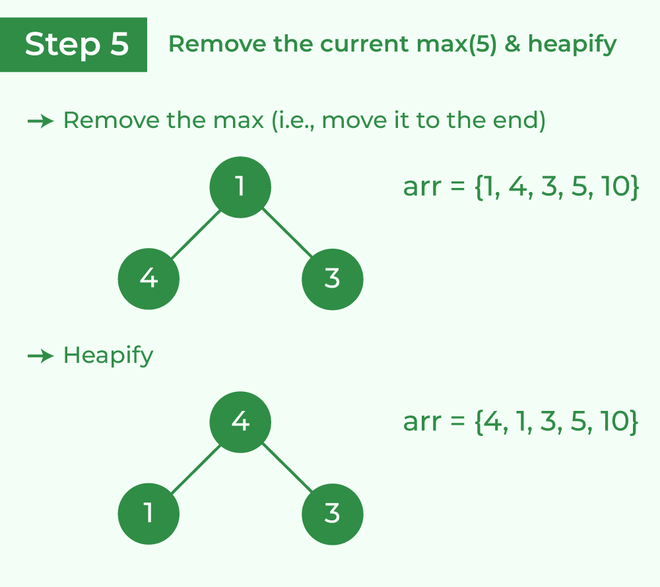
**Perform heap sort:** Remove the maximum element in each step (i.e., move it to the end position and remove that) and then consider the remaining elements and transform it into a max heap.

* Delete the root element **(10)** from the max heap. In order to delete this node, try to swap it with the last node, i.e. **(1).** After removing the root element, again heapify it to convert it into max heap.
  + Resulted heap and array should look like this:



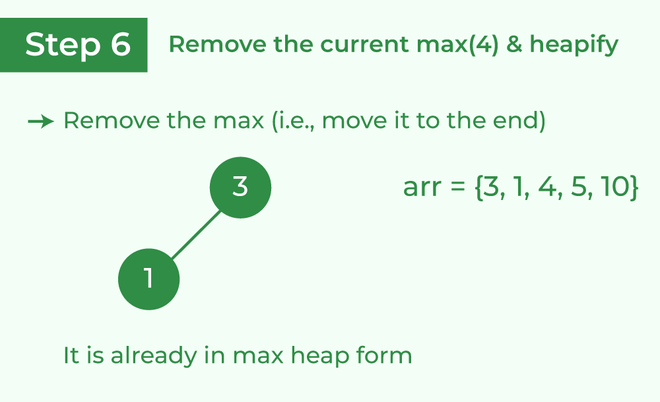
Remove 10 and perform heapify

* Repeat the above steps and it will look like the following:



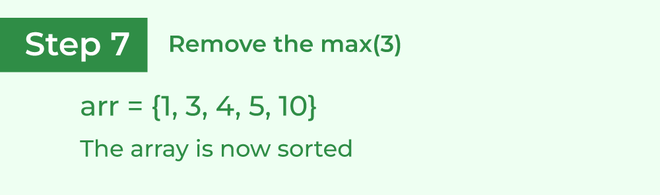
Remove 5 and perform heapify

* Now remove the root (i.e. 3) again and perform heapify.



Remove 4 and perform heapify

* Now when the root is removed once again it is sorted. and the sorted array will be like **arr[] = {1, 3, 4, 5, 10}**.



The sorted array