- Used in External Sorting

Drawbacks of Merge Sort:

- Slower compared to the other sort algorithms for smaller tasks.
- The merge sort algorithm requires an additional memory space of 0(n) for the temporary array.
- It goes through the whole process even if the array is sorted.
- Recent Articles on Merge Sort
- Coding practice for sorting.
- Quiz on Merge Sort

Solution of the drawback for additional storage:
Use linked list.

Heap Sort:

Heap sort *is a comparison-based sorting technique based on Binary Heap data structure. It is similar to the 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)
- Typically 2-3 times slower than well-implemented QuickSort. 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

Applications of HeapSort:

- Heapsort is mainly used in hybrid algorithms like the IntroSort.
- Sort a nearly sorted (or K sorted) array
- k largest(or 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

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])*

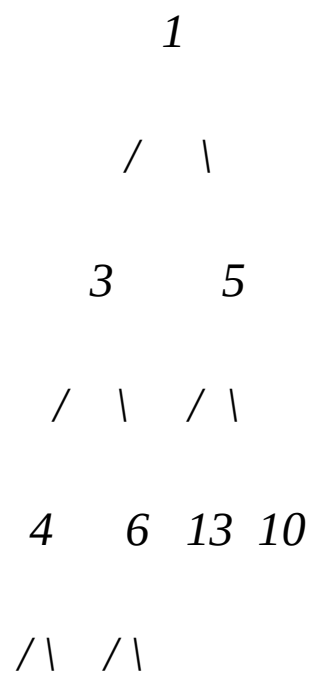
if(Root != Largest)

Swap(Root, Largest)

How does Heapify work?

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

Corresponding Complete Binary Tree is:



9 8 15 17

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

Total Nodes = 11.

Last Non-leaf node index = (11/2) – 1 = 4.

Therefore, last non-leaf node = 6.

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

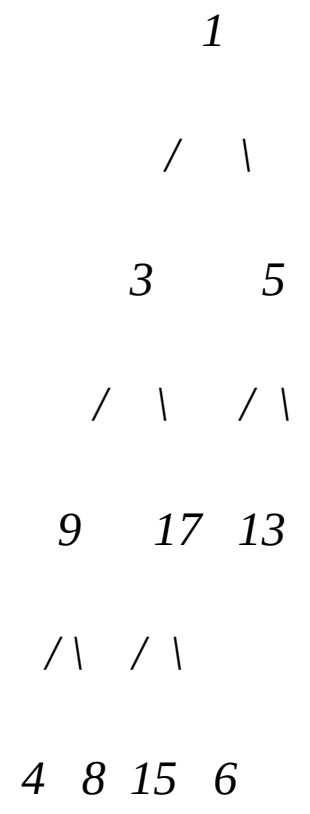
*/ \ / *

4 17 13 10

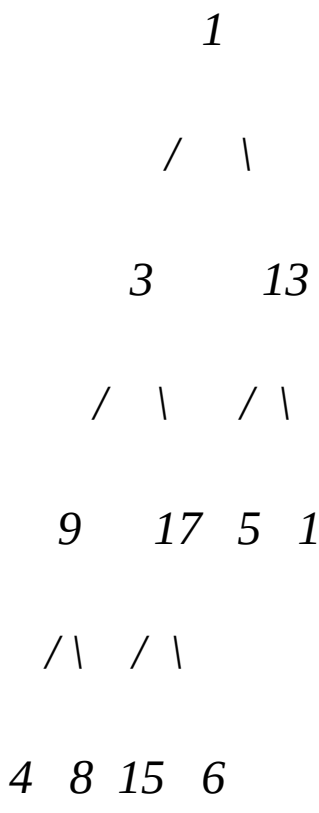
*/ \ / *

9 8 15 6

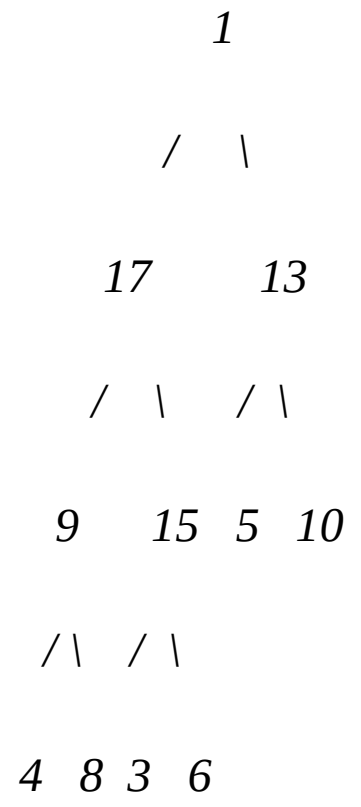
Heapify 4*: Swap 4 and 9.*



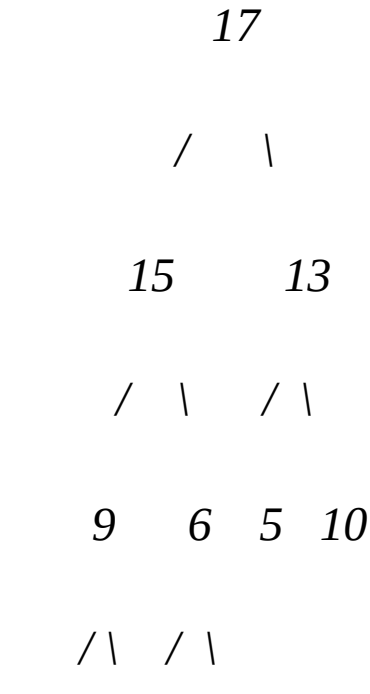
Heapify 5*: Swap 13 and 5.*



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



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



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, than 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.

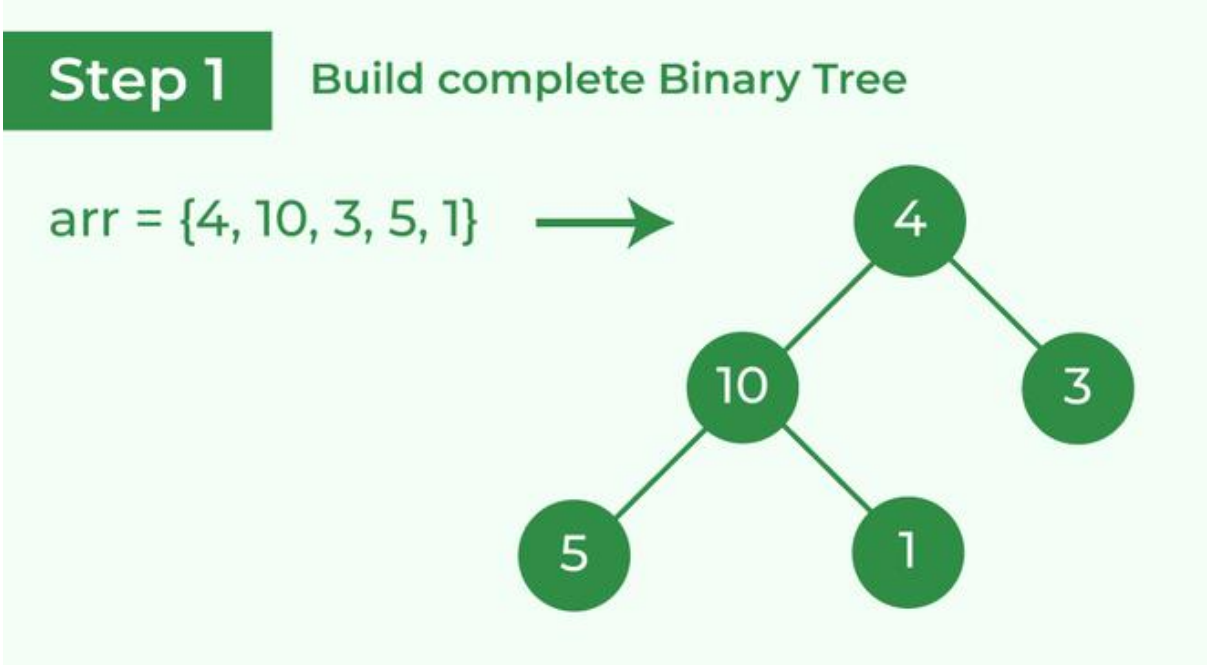
Note: The heapify procedure can only be applied to a node if its children nodes are heapified. So the heapification must be performed in the bottom-up order.

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.*



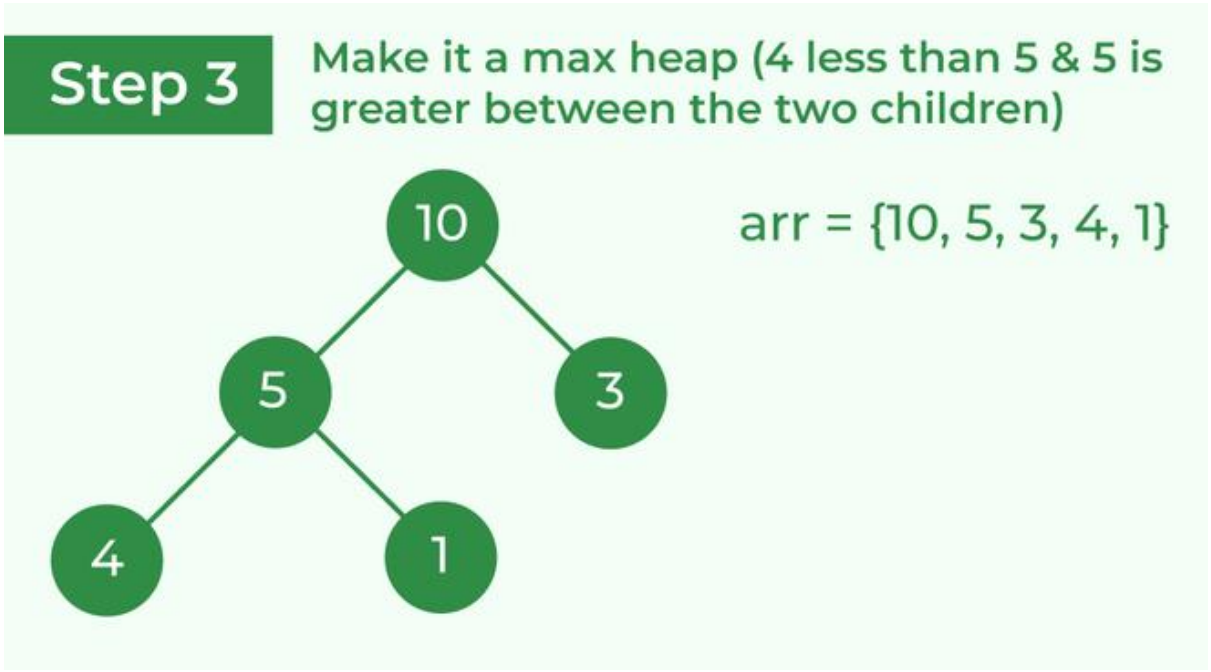
Build complete binary tree from the array

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.*

- *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

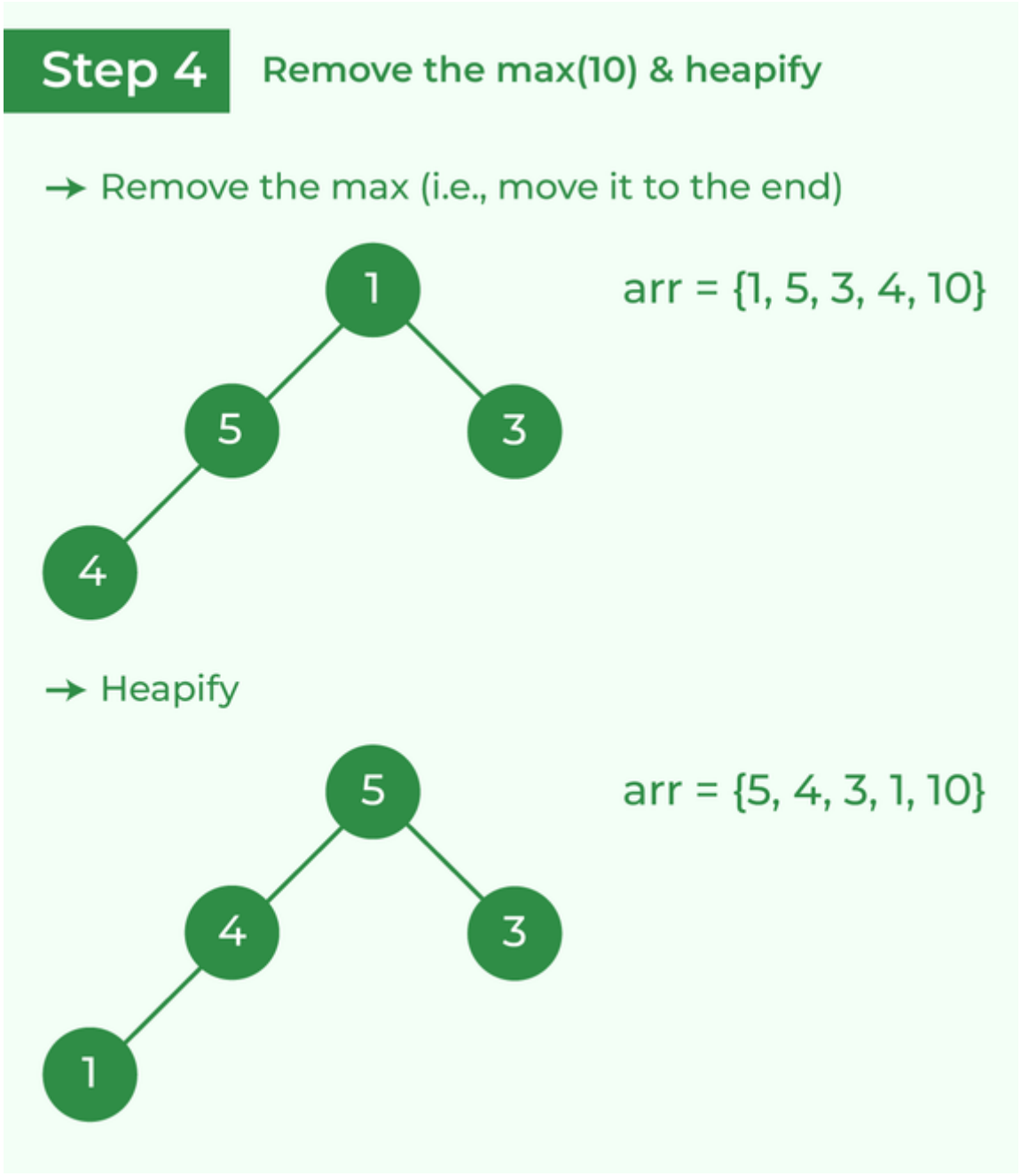
- *Now, as seen, **4** as a parent is smaller than the child **5**, thus swap both of these again and the resulted heap and array should be like this:*



Make the tree a max heap

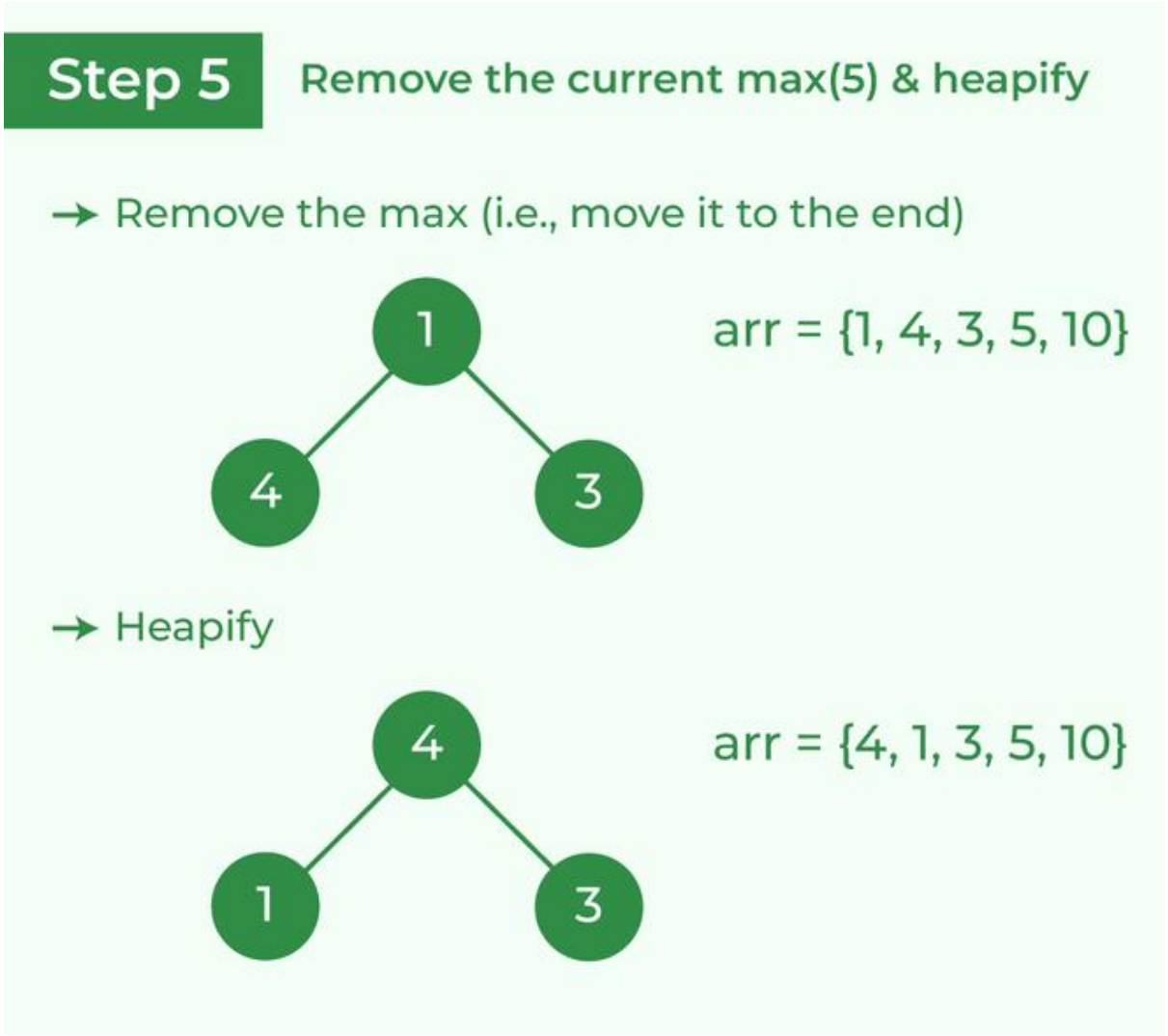
***Perform heap sort:** Remove the maximum element in each step (i.e., move it to the end position and remove that) and then consider the remaining elements and transform it into a max heap.*

- *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:*



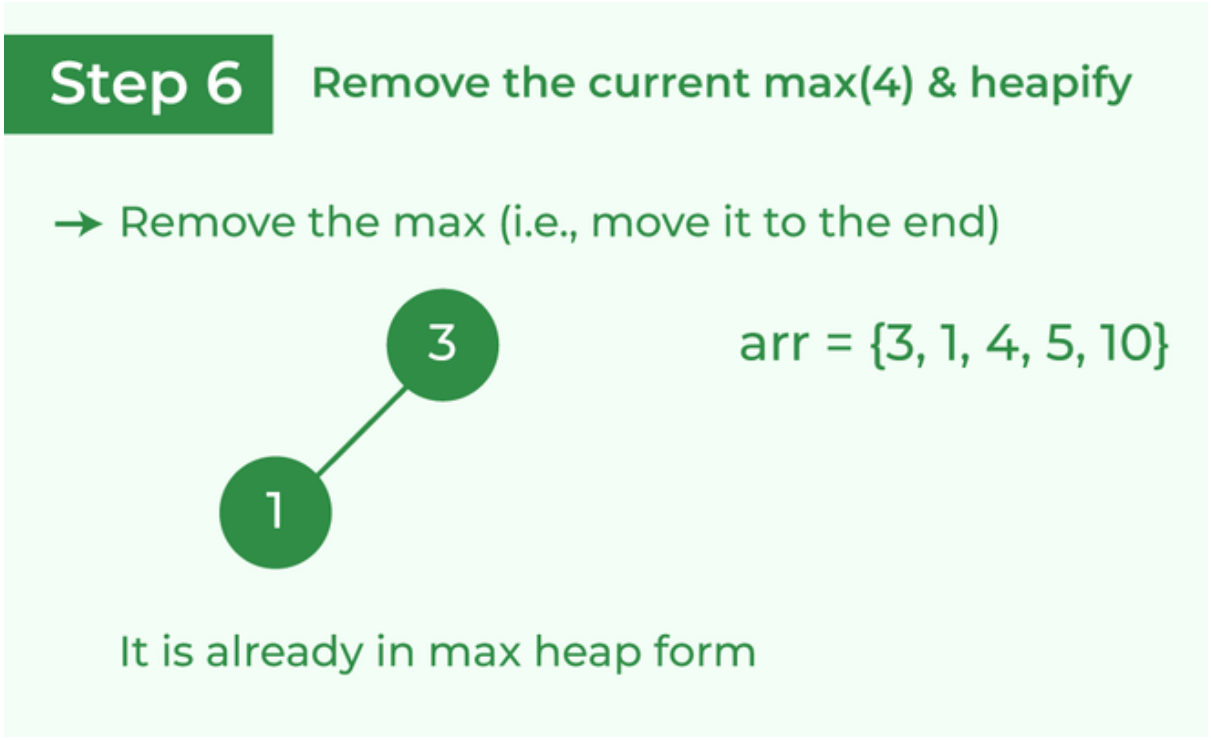
Remove 10 and perform heapify

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



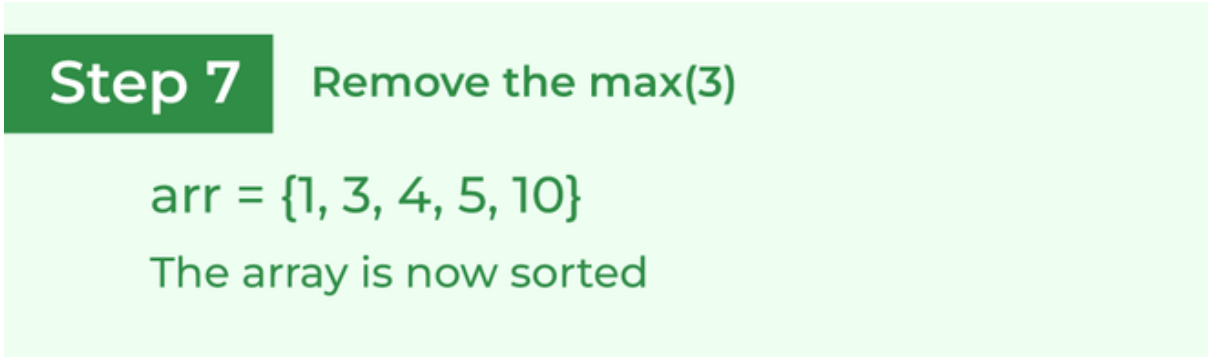
Remove 5 and perform heapify

- *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

Implementation of Heap Sort

Below is the implementation of the above approach:

- C
- C++
- Java
- Python3
- C#
- PHP
- Javascript

```
// Heap Sort in C

#include <stdio.h>

// Function to swap the position of two elements

void swap(int* a, int* b)
{

    int temp = *a;

    *a = *b;

    *b = temp;
}

// To heapify a subtree rooted with node i
// which is an index in arr[].
// n is size of heap
void heapify(int arr[], int N, int i)
```

```
{
    // Find largest among root, left child and right child

    // Initialize largest as root
    int largest = i;

    // left = 2*i + 1
    int left = 2 * i + 1;

    // right = 2*i + 2
    int right = 2 * i + 2;

    // If left child is larger than root
    if (left < N && arr[left] > arr[largest])

        largest = left;

    // If right child is larger than largest
    // so far
    if (right < N && arr[right] > arr[largest])
```

```
        largest = right;

    // Swap and continue heapifying if root is not largest
    // If largest is not root
    if (largest != i) {

        swap(&arr[i], &arr[largest]);

        // Recursively heapify the affected
        // sub-tree
        heapify(arr, N, largest);
    }
}

// Main function to do heap sort
void heapSort(int arr[], int N)
{

    // Build max heap
```

```
    for (int i = N / 2 - 1; i >= 0; i--)

        heapify(arr, N, i);

    // Heap sort
    for (int i = N - 1; i >= 0; i--) {

        swap(&arr[0], &arr[i]);

        // Heapify root element to get highest element at
        // root again
        heapify(arr, i, 0);
    }
}

// A utility function to print array of size n
void printArray(int arr[], int N)
{
    for (int i = 0; i < N; i++)
        printf("%d ", arr[i]);
```

```
    printf("\n");
}

// Driver's code
int main()
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int N = sizeof(arr) / sizeof(arr[0]);

    // Function call
    heapSort(arr, N);
    printf("Sorted array is\n");
    printArray(arr, N);
}

// This code is contributed by _i_plus_plus_.
```

Output

Sorted array is

5 6 7 11 12 13

Time Complexity: O(N log N)
Auxiliary Space: O(1)

Some FAQs related to Heap Sort

What are the two phases of Heap Sort?

The heap sort algorithm consists of two phases. In the first phase the array is converted into a max heap. And in the second phase the highest element is removed (i.e., the one at the tree root) and the remaining elements are used to create a new max heap.

Why Heap Sort is not stable?

Heap sort algorithm is not a stable algorithm. This algorithm is not stable because the operations that are performed in a heap can change the relative ordering of the equivalent keys.

Is Heap Sort an example of "Divide and Conquer" algorithm?

Heap sort is **NOT** at all a Divide and Conquer algorithm. It uses a heap data structure to efficiently sort its element and not a "divide and conquer approach" to sort the elements.

Which sorting algorithm is better – Heap sort or Merge Sort?

The answer lies in the comparison of their time complexity and space requirement. The Merge sort is slightly faster than the Heap sort. But on the other hand merge sort takes extra memory. Depending on the requirement, one should choose which one to use.

Why Heap sort better than Selection sort?

Heap sort is similar to selection sort, but with a better way to get the maximum element. It takes advantage of the heap data structure to get the maximum element in constant time.