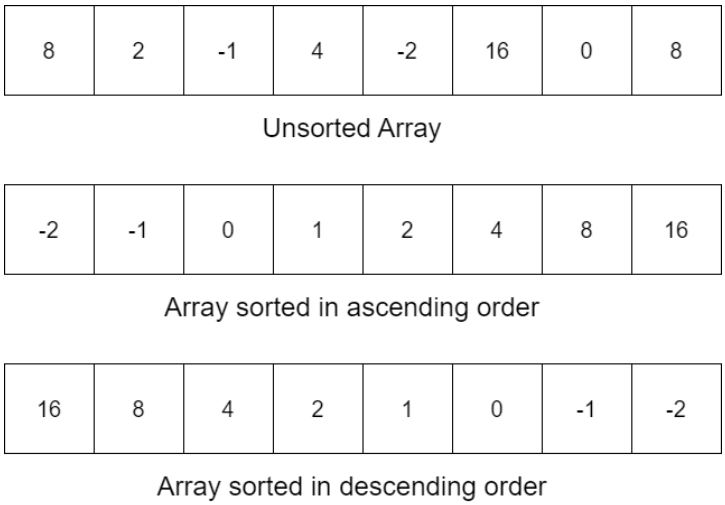
Sorting Algorithms

A Sorting Algorithm is used to rearrange a given array or list of elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of elements in the respective data structure.



An example of Sorting

Types of Sorting Algorithms:
Bubble Sort
Selection Sort
Insertion Sort
Merge Sort
Quick Sort
Heap Sort

Bubble Sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

Who invented the bubble sorting algorithm ?
The earliest description of the Bubble sort algorithm was in a 1956 paper by mathematician and actuary Edward Harry Friend,

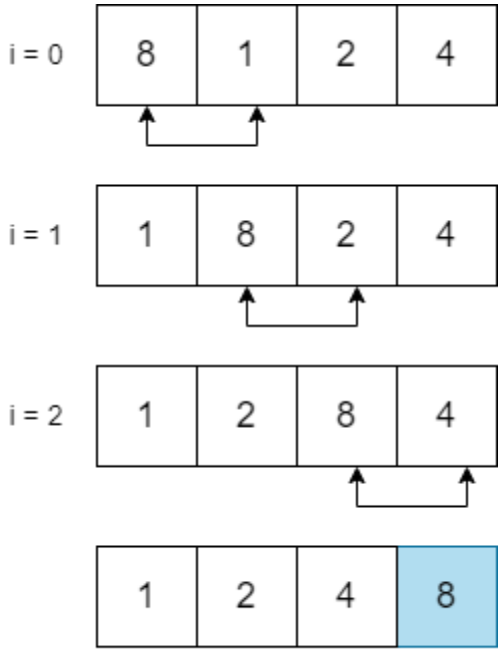
In this algorithm,

- Traverse from left and compare adjacent elements and the higher one is placed at the right side.
- In this way, the largest element is moved to the rightmost end at first.
- This process is then continued to find the second largest and place it and so until the data is stored

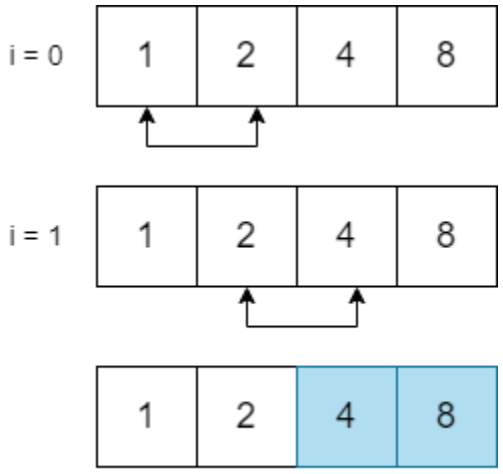
Example

int [] ar = {8,2,1,4}

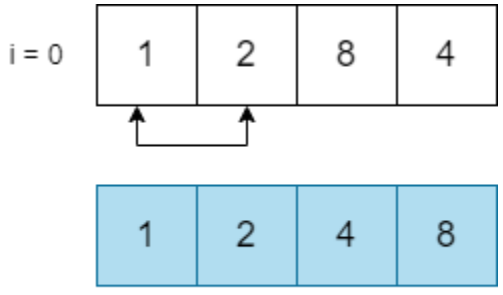
First Pass:
The largest element is placed in the last position.



Second Pass:
Placing the second largest element in the second largest element.



Third Pass:
Place the remaining two elements at their correct positions.



Implementation Code

```
import java.util.Arrays;

class Main {

  // perform the bubble sort
  static void bubbleSort(int array[]) {
    int size = array.length;

    // loop to access each array element
    for (int i = 0; i < size - 1; i++)

      // loop to compare array elements
      for (int j = 0; j < size - i - 1; j++)

        // compare two adjacent elements
        // change > to < to sort in descending order
        if (array[j] > array[j + 1]) {

          // swapping occurs if elements
          // are not in the intended order
          int temp = array[j];
          array[j] = array[j + 1];
          array[j + 1] = temp;
        }
  }

  public static void main(String args[]) {

    int[] data = { 1, 2, 4 ,8 };

    // call method using class name
    Main.bubbleSort(data);

    System.out.println("Sorted Array in Ascending Order:");
    System.out.println(Arrays.toString(data));
  }
}
```

Selection Sort

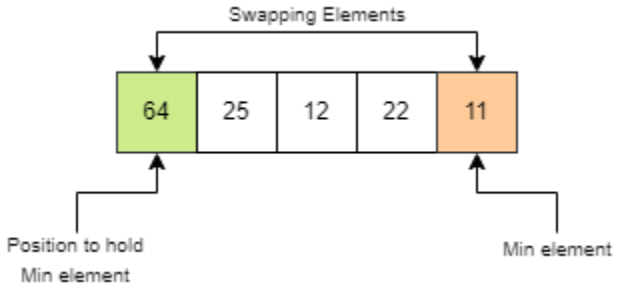
The algorithm repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first element of the unsorted part. This process is repeated for the remaining unsorted portion until the entire list is sorted.

Example

int arr[] = {64, 25, 12, 22, 11};

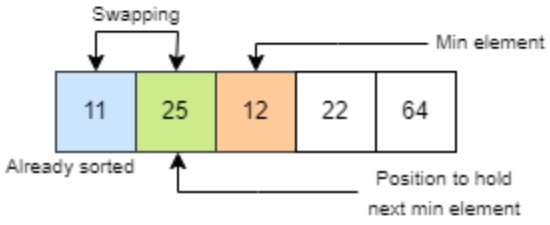
First Pass:

- For the first position in the sorted array, the whole array is traversed from index 0 to 4 sequentially. The first position where 64 is stored presently, after traversing the whole array it is clear that 11 is the lowest value.
- Thus, replace 64 with 11. After one iteration 11, which happens to be the least value in the array, tends to appear in the first position of the sorted list.



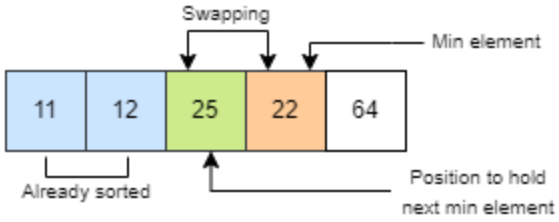
Second Pass:

- For the second position, where 25 is present, again traverse the rest of the array in a sequential manner.
- After traversing, we found that 12 is the second lowest value in the array and it should appear at the second place in the array, thus swap these values.



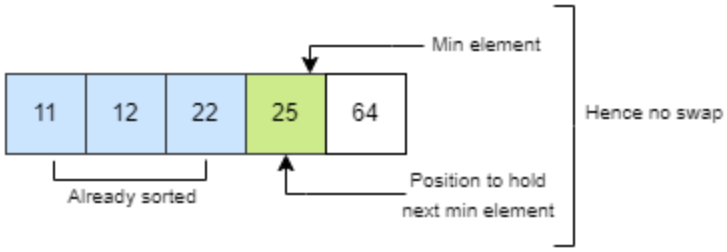
Third Pass:

- Now, for third place, where 25 is present again, traverse the rest of the array and find the third least value present in the array.
- While traversing, 22 came out to be the third least value and it should appear at the third place in the array, thus swap 22 with element present at third position.



Fourth Pass:

- Similarly, for fourth position traverse the rest of the array and find the fourth least element in the array.
- As 25 is the 4th lowest value hence, it will place at the fourth position.



Fifth Pass:

- At last the largest value present in the array automatically get placed at the last position in the array.
- The resulting array is the sorted array.

Implementation Code

```
import java.util.Arrays;

class SelectionSort {
  void selectionSort(int array[]) {
    int size = array.length;

   for (int step = 0; step < size - 1; step++) {
     int min_idx = step;

     for (int i = step + 1; i < size; i++) {

       // To sort in descending order, change > to < in this line.
```

```
      // Select the minimum element in each loop.
      if (array[i] < array[min_idx]) {
        min_idx = i;
      }
    }

    // put min at the correct position
    int temp = array[step];
    array[step] = array[min_idx];
    array[min_idx] = temp;
  }
 }

 // driver code
 public static void main(String args[]) {
  int[] data = { 20, 12, 10, 15, 2 };
  SelectionSort ss = new SelectionSort();
  ss.selectionSort(data);
  System.out.println("Sorted Array in Ascending Order: ");
  System.out.println(Arrays.toString(data));
 }
}
```

Merge Sort

In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.

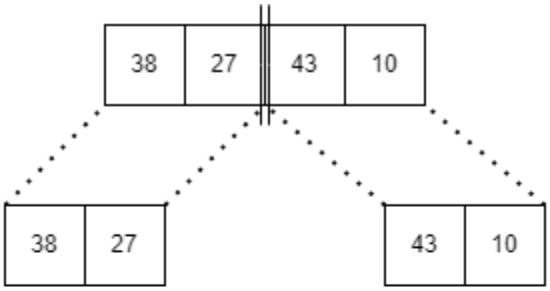
Who invented the merge sorting algorithms?

Mergesort is a divide and conquer algorithm that was invented by John von Neumann in 1945.

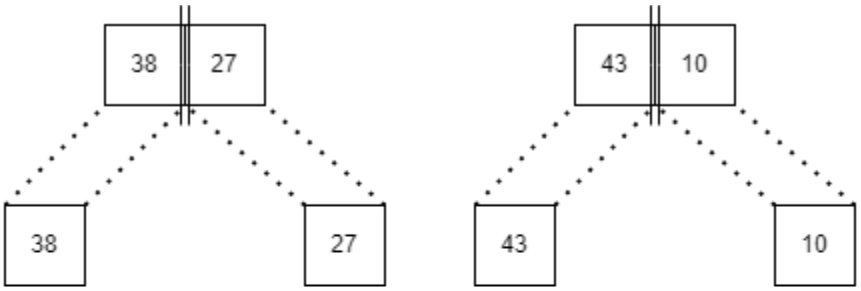
Example

int array[]={38,27,43,10}

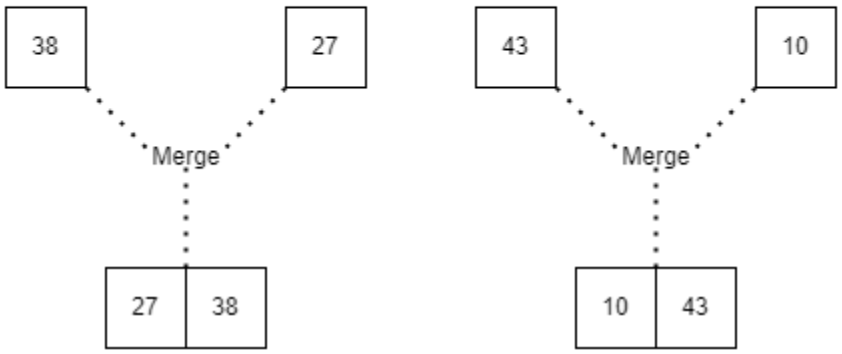
- Initially divide the array into two equal halves:



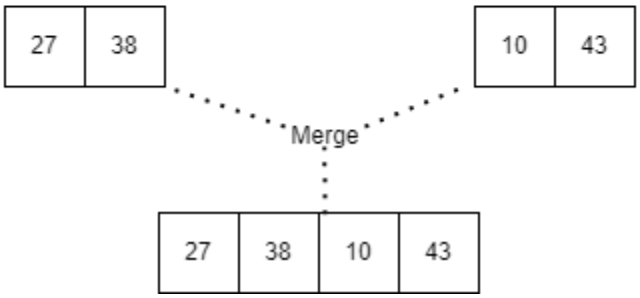
- These subarrays are further divided into two halves. Now they become arrays of unit length that can no longer be divided and arrays of unit length are always sorted.



- These sorted subarrays are merged together, and we get bigger sorted subarrays.



- This merging process is continued until the sorted array is built from the smaller subarrays.



Insertion Sort

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

Who invented the Insertion sort algorithm ?

Insertion sort was invented by John Mauchly as early as 1946,

Example

int arr[] = {12, 11, 13, 5, 6};

12	11	13	5	6

First Pass:

- Initially, the first two elements of the array are compared in insertion sort.

12	11	13	5	6

- Here, 12 is greater than 11 hence they are not in the ascending order and 12 is not at its correct position. Thus, swap 11 and 12.
- So, for now 11 is stored in a sorted sub-array.

11	12	13	5	6

Second Pass:

- Now, move to the next two elements and compare them.

11	12	13	5	6

- Here, 13 is greater than 12, thus both elements seem to be in ascending order, hence, no swapping will occur. 12 also stored in a sorted subarray along with 11

Third Pass:

- Now, two elements are present in the sorted subarray which are 11 and 12
- Moving forward to the next two elements which are 13 and 5

11	12	13	5	6

- Both 5 and 13 are not present at their correct place so swap them.

11	12	5	13	6

- After swapping, elements 12 and 5 are not sorted, thus swap again.

11	5	12	13	6

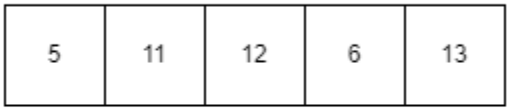
- Here, again 11 and 5 are not sorted, hence swap again

- Here, 5 is at its correct position

Fourth Pass:

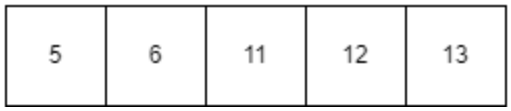
- Now, the elements which are present in the sorted subarray are 5, 11 and 12
- Moving to the next two elements 13 and 6

- Clearly, they are not sorted, thus perform swap between both



- Now, 6 is smaller than 12, hence, swap again

- Here, also swapping makes 11 and 6 unsorted hence, swap again



- Finally, the array is completely sorted.

QuickSort

QuickSort is a sorting algorithm based on the Divide and Conquer algorithm that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array.

Who invented the Quicksort algorithm ?

Quicksort algorithm was founded by Tony Hoare in 1960 in Moscow

Choice of Pivot:

There are many different choices for picking pivots.

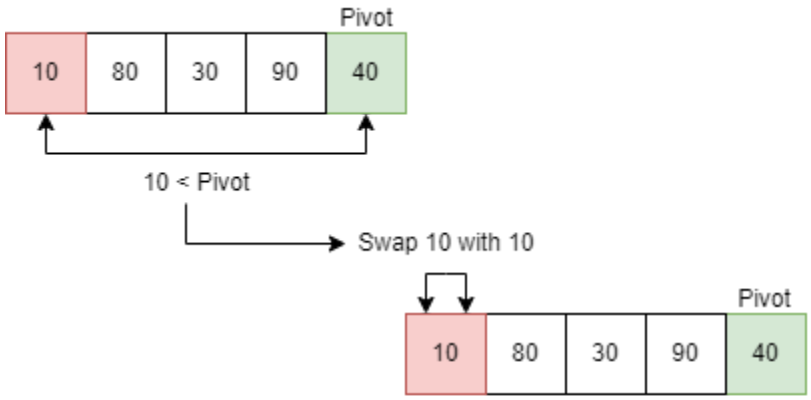
- Always pick the first element as a pivot.
- Always pick the last element as a pivot (implemented below)
- Pick a random element as a pivot.
- Pick the middle as the pivot.

Partition Algorithm:

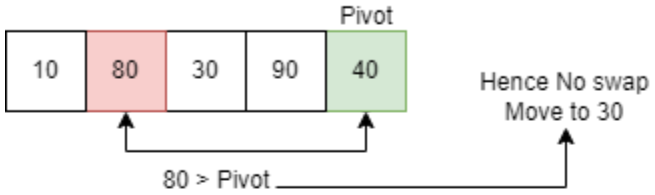
The logic is simple, we start from the leftmost element and keep track of the index of smaller (or equal) elements as i. While traversing, if we find a smaller element, we swap the current element with arr[i]. Otherwise, we ignore the current element.

Example
int arr[] = {10, 80, 30, 90, 40}.

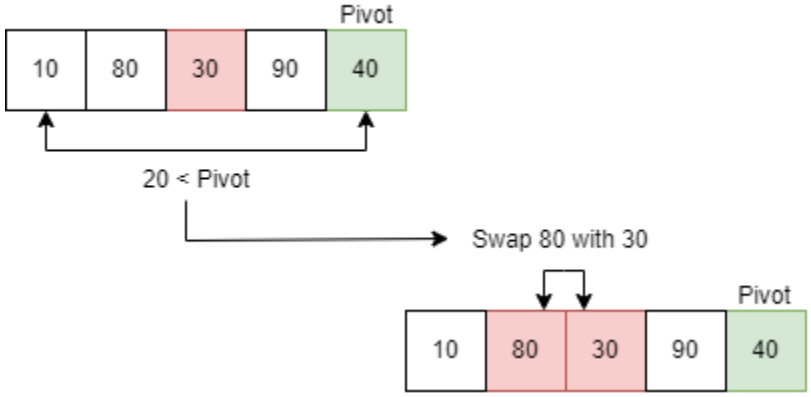
- Compare 10 with the pivot and as it is less than pivot arrange it accordingly.



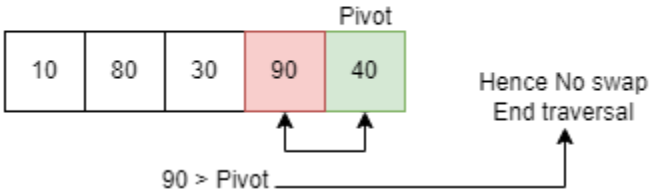
- Compare 80 with the pivot. It is greater than pivot.



- Compare 30 with pivot. It is less than pivot so arrange it accordingly.



- Compare 90 with the pivot. It is greater than the pivot.



- Arrange the pivot in its correct position.

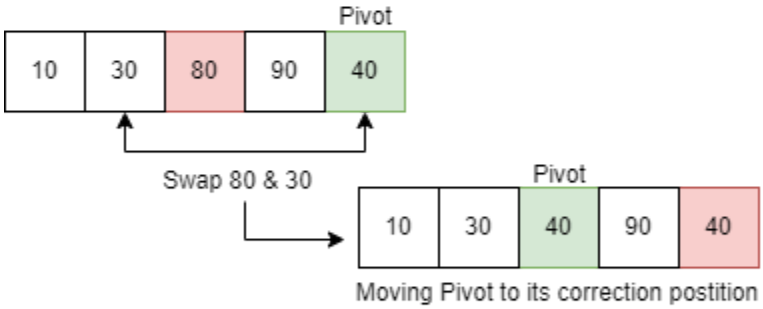
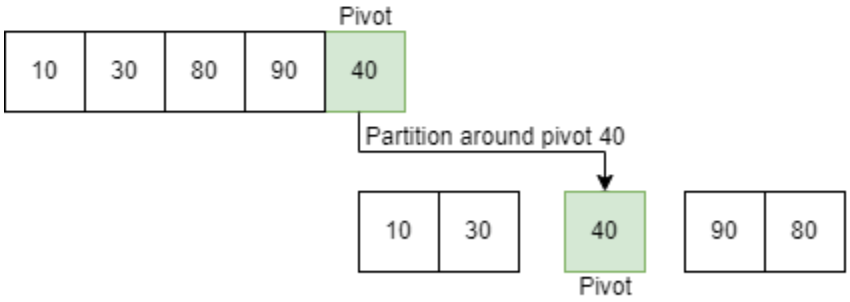


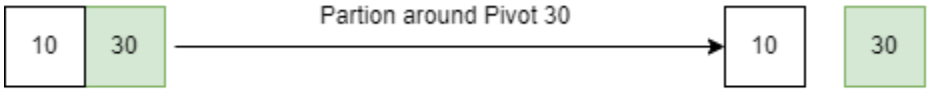
Illustration of Quicksort:
As the partition process is done recursively, it keeps on putting the pivot in its actual position in the sorted array. Repeatedly putting pivots in their actual position makes the array sorted.

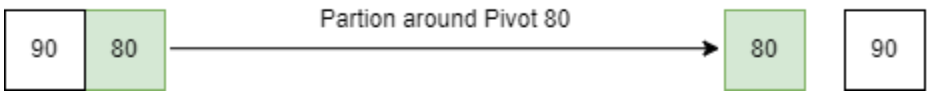
Follow the below images to understand how the recursive implementation of the partition algorithm helps to sort the array.

- Initial partition on the main array:



- Partitioning of the subarrays:



- 

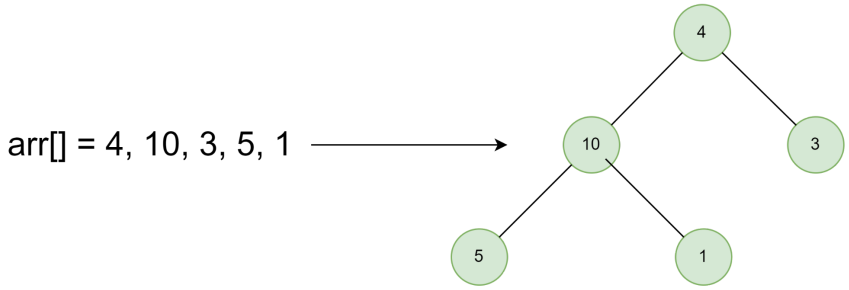
Heap Sort

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.

Who invented the Heapsort Algorithm ?
Heapsort was invented by J. W. J. Williams in 1964.

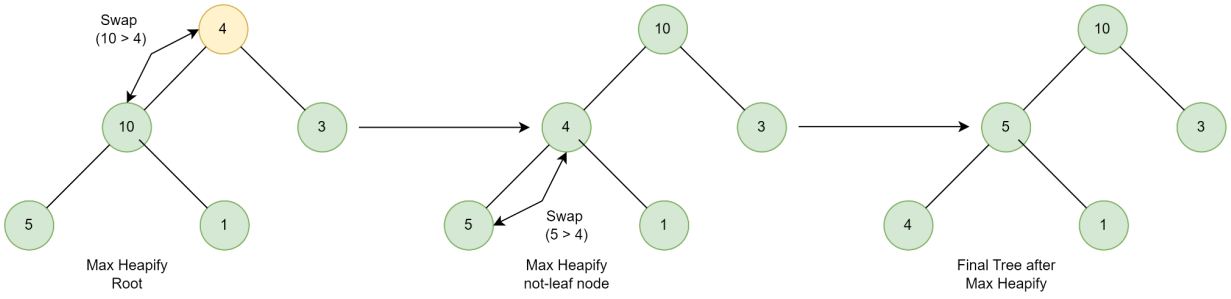
Example

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



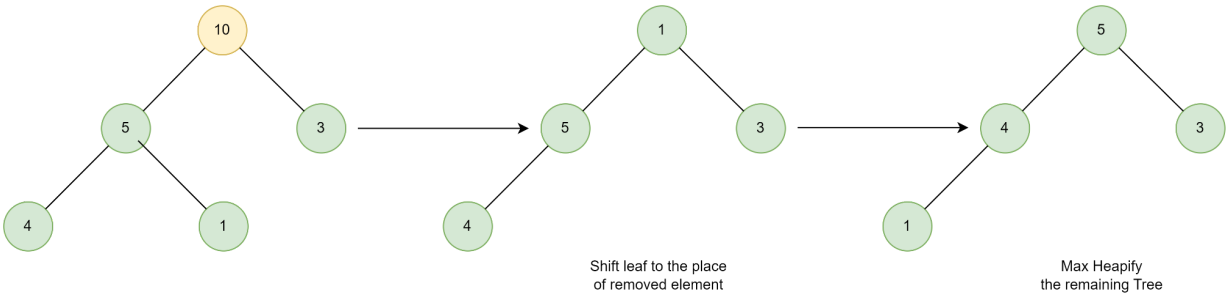
Transform into max heap: After that, the task is to construct a tree from that unsorted array and try to convert it into max heap.

- 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.
- Now, 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:

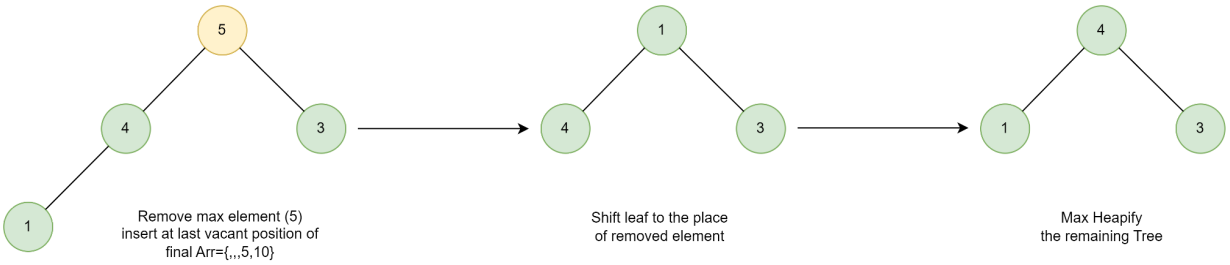


Perform heap sort: Remove the maximum element in each step (i.e., move it to the end position and remove that) and then consider the remaining elements and transform it into a max heap.

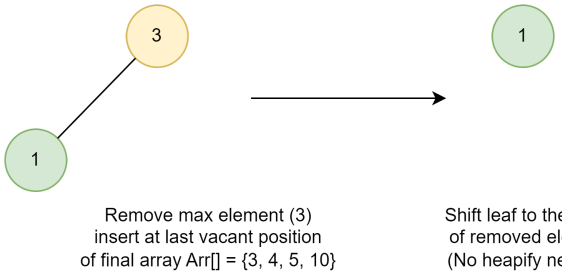
- Delete the root element (10) from the max heap. In order to delete this node, try to swap it with the last node, i.e. (1). After removing the root element, again heapify it to convert it into max heap.
- Resulted heap and array should look like this:



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



- Now remove the root (i.e. 3) again 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}.