BUBBLE SORT:

1. Bubble sort is an algorithm for sorting a given list of items.

2. It works by repeatedly looping through the list, comparing adjacent items and swapping them if they are in the wrong order.

3.This process continues until no more swaps are needed, at which point the list is sorted.

EXAMPLE:

Input {6,3,0,5}

Bubble sort start with very first two elements, comparing them to check which one is greater

(6,3,0,5) → (3 6 0 5) compare the first two elements and swap since 6>3.

(3 6 0 5) →(3 0 6 5) swap since 6>0

(3 0 6 5) →(3 0 5 6) swap since 6>5

SECOND CASE:

(3 0 6 5) → (0 3 5 6) swap since 3>0

(0 3 5 6)→(0 3 5 6) no changes as 5>3

THIRD CASE:

Now the array is already sorted but our algorithm does not know if it is completed.

The algorithm needs one whole pass without any swap to know it is sorted.

(0 3 5 6) →(0 3 5 6) no change as 3>0

Array is now sorted and no more pass will happen.

INSERTION SORT:

It works by taking elements from the list one by one and inserting them in their correct position into a new sorted list.

1. Start with an empty sorted list.

2. Take the first element from the unsorted list and insert it into the sorted list.

3. Compare the next element from the unsorted list with the elements in the sorted list, and insert it into the correct position.

4. Continue this process until all elements are inserted into the sorted list.

EXAMPLE:

STEP 1:

Input = [8 3 5 1 4 2]

key = 3

Here `key` will be compared with the previous elements.

In this case, `key` is compared with 8. since 8 > 3, move the element 8
to the next position and insert `key` to the previous position.

Result: [3 8 5 1 4 2]

STEP 2:

key = 5 //2nd index

8 > 5 //move 8 to 2nd index and insert 5 to the 1st index.

Result: [3 5 8 1 4 2]

STEP 3:

key = 1 //3rd index

8 > 1 => [3 5 1 8 4 2]

5 > 1 => [3 1 5 8 4 2]

3 > 1 => [1 3 5 8 4 2]

Result: [1 3 5 8 4 2]

STEP 4:

key = 4 //4th index

8 > 4 => [1 3 5 4 8 2]

5 > 4 => [1 3 4 5 8 2]

3 > 4 ≠> stop

Result: [1 3 4 5 8 2]

STEP 5:

C key = 2 //5th index

8 > 2 => [1 3 4 5 2 8]

5 > 2 => [1 3 4 2 5 8]

4 > 2 => [1 3 2 4 5 8]

3 > 2 => [1 2 3 4 5 8]

1 > 2 ≠> stop

Result: [1 2 3 4 5 8]

HEAP SORT:

Heap sort is one of the sorting algorithms used to arrange a list of elements in order. Heapsort algorithm uses one of the tree concepts called **Heap Tree**. In this sorting algorithm, we use **Max Heap** to arrange list of elements in Descending order and **Min Heap** to arrange list elements in Ascending order.

The Heap sort algorithm to arrange a list of elements in ascending order is performed using following steps...

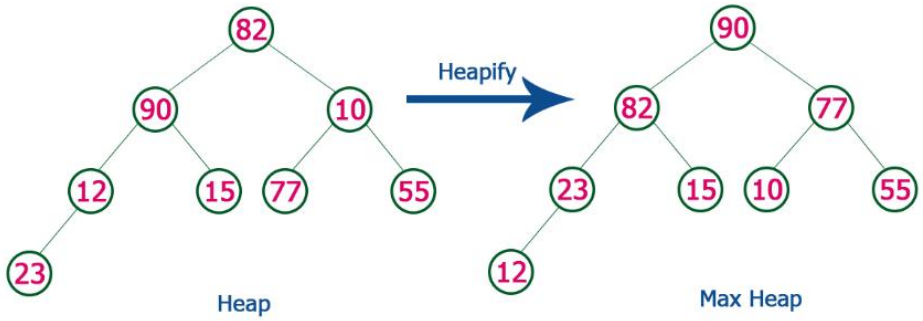
- **Step 1 -** Construct a **Binary Tree** with given list of Elements.
- **Step 2 -** Transform the Binary Tree into **Min Heap.**
- **Step 3 -** Delete the root element from Min Heap using **Heapify** method.
- **Step 4 -** Put the deleted element into the Sorted list.
- **Step 5 -** Repeat the same until Min Heap becomes empty.
- **Step 6 -** Display the sorted list.

EXAMPLE:

Consider the following list of unsorted numbers which are to be sort using Heap Sort

82, 90, 10, 12, 15, 77, 55, 23

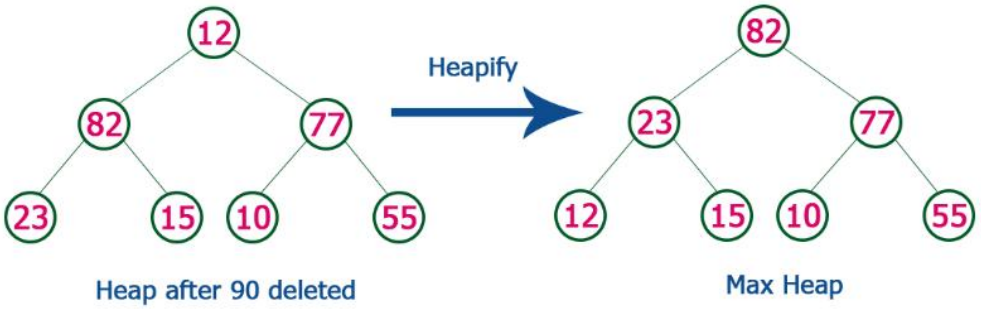
Step 1 - Construct a Heap with given list of unsorted numbers and convert to Max Heap



list of numbers after heap converted to Max Heap

90, 82, 77, 23, 15, 10, 55, 12

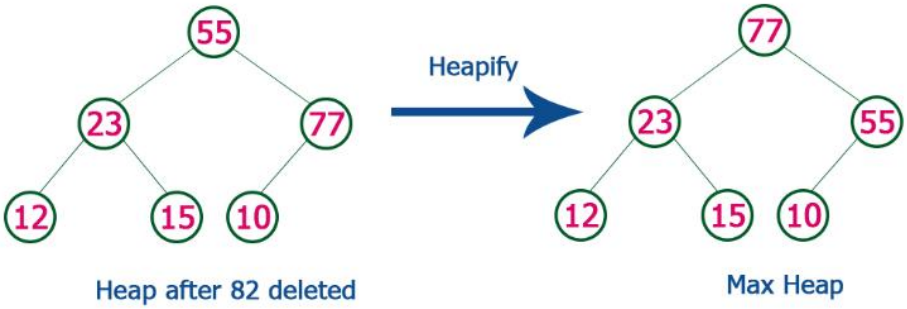
Step 2 - Delete root (**90**) from the Max Heap. To delete root node it needs to be swapped with last node (**12**). After delete tree needs to be heapify to make it Max Heap.



list of numbers after swapping 90 with 12.

12, 82, 77, 23, 15, 10, 55, 90

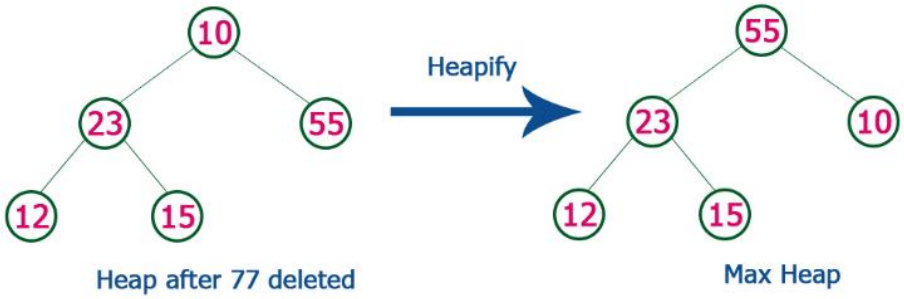
Step 3 - Delete root (**82**) from the Max Heap. To delete root node it needs to be swapped with last node (**55**). After delete tree needs to be heapify to make it Max Heap.



list of numbers after swapping 82 with 55.

12, 55, 77, 23, 15, 10, 82, 90

Step 4 - Delete root (**77**) from the Max Heap. To delete root node it needs to be swapped with last node (**10**). After delete tree needs to be heapify to make it Max Heap.



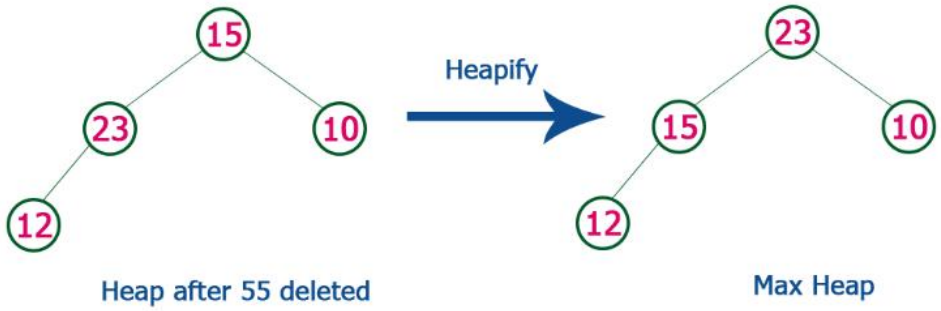
Heap after 77 deleted

Max Heap

list of numbers after swapping 77 with 10.

12, 55, 10, 23, 15, **77**, **82**, **90**

Step 5 - Delete root (**55**) from the Max Heap. To delete root node it needs to be swapped with last node (**15**). After delete tree needs to be heapify to make it Max Heap.



list of numbers after swapping 55 with 15.

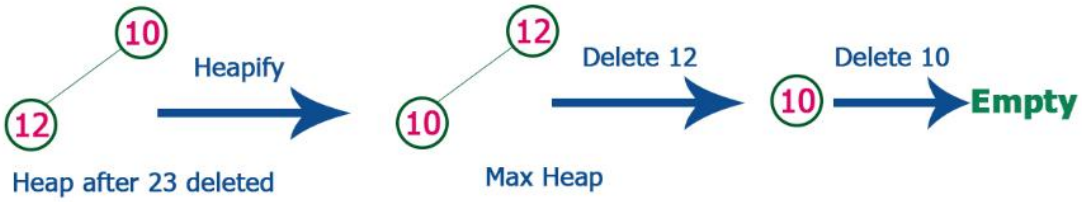
12, 15, 10, 23, 55, 77, 82, 90

Step 6 - Delete root (**23**) from the Max Heap. To delete root node it needs to be swapped with last node (**12**). After delete tree needs to be heapify to make it Max Heap.

list of numbers after swapping 23 with 12.

12, 15, 10, 23, 55, 77, 82, 90

Step 7 - Delete root (**15**) from the Max Heap. To delete root node it needs to be swapped with last node (**10**). After delete tree needs to be heapify to make it Max Heap.



list of numbers after Deleting 15, 12 & 10 from the Max Heap.

10, 12, 15, 23, 55, 77, 82, 90

Whenever Max Heap becomes Empty, the list get sorted in Ascending order