HEAP SORT

- Heap sort is a comparison based sorting algorithm.
- It is a special tree-based data structure.
- Heap sort processes the elements by creating the min-heap or max-heap using the elements of the given array
- Min-heap or max-heap represents the ordering of array in which the root elementrepresents the minimum or maximum element of the array
- Heap sort basically recursively performs two main operations
  - Build a heap H, using the elements of array.
  - Repeatedly delete the root element of the heap formed in 1st phase.

What is a heap?

- A heap is a complete binary tree, and the binary tree is a tree in which the node can have the utmost two children. A complete binary tree is a binary tree in which all the levels except the last level, i.e., leaf node, should be completely filled, and all the nodes should be left-justified.

Working of Heap sort Algorithm

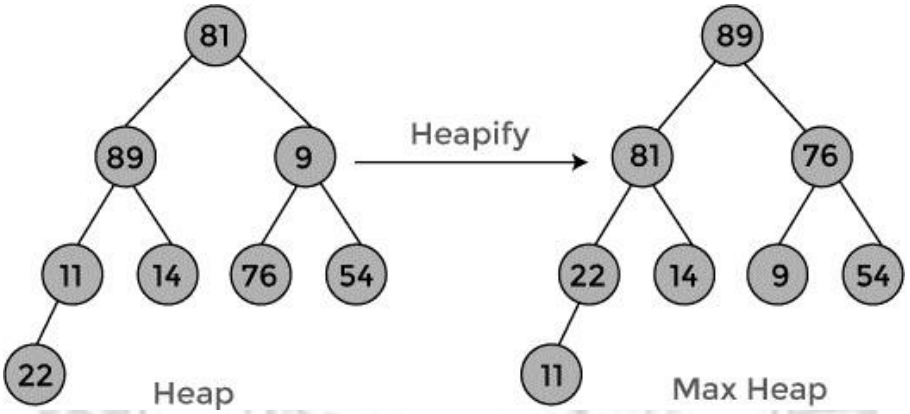
- In heap sort, basically, there are two phases involved in the sorting of elements. By using the heap sort algorithm, they are as follows:
- The first step includes the creation of a heap by adjusting the elements of the array.
- After the creation of heap, now remove the root element of the heap repeatedly by shifting it to the end of the array, and then store the heap structure with the remaining elements.

Consider an unsorted array as follows

81, 89, 9, 11, 14, 76, 54, 22

Given array is

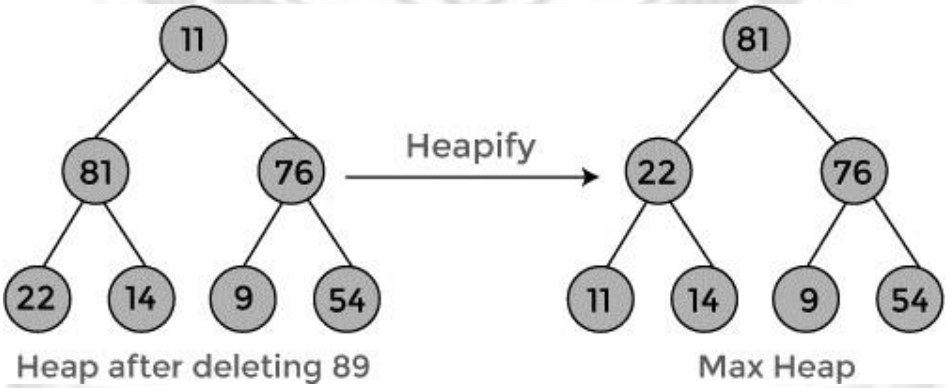
First, construct a heap from the given array and convert it into max heap



After converting the given heap into max heap, the array elements are

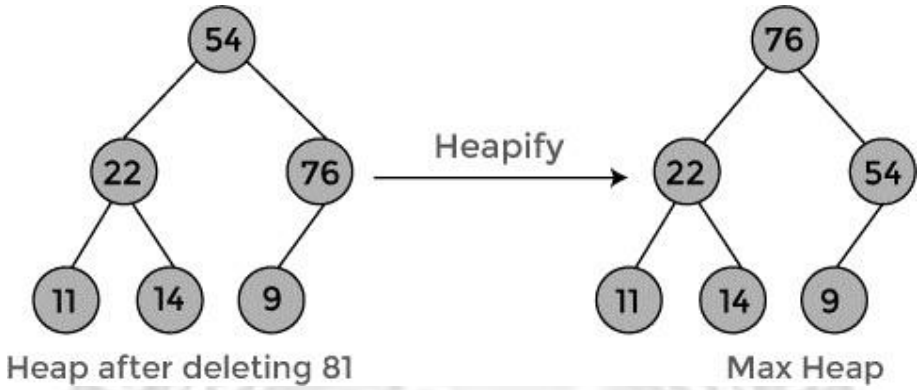
89	81	76	22	14	9	54	11

- Next step is to delete the root element (89) from the max heap. To delete this node, swap it with the last node, i.e. (11). After deleting the root element, again heapify it to convert it into max heap.

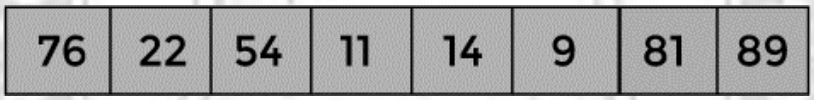


- After swapping the array element 89 with 11, and converting the heap into max-heap, the elements of array are

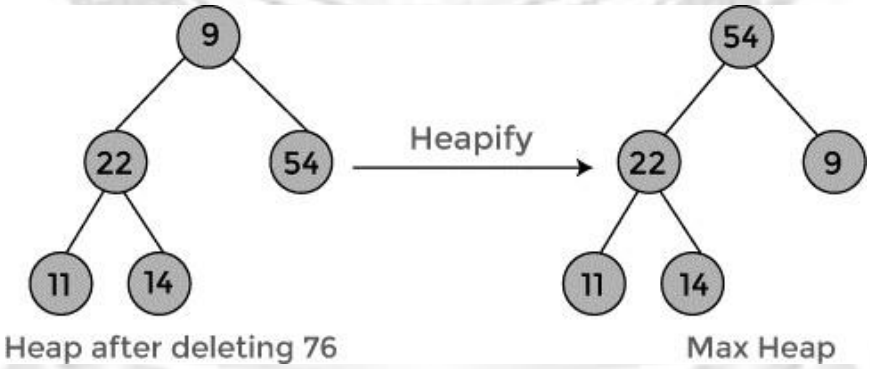
81	22	76	11	14	9	54	89



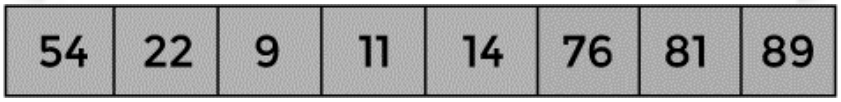
- After swapping the array element 81 with 54 and converting the heap into max-heap, the elements of array are



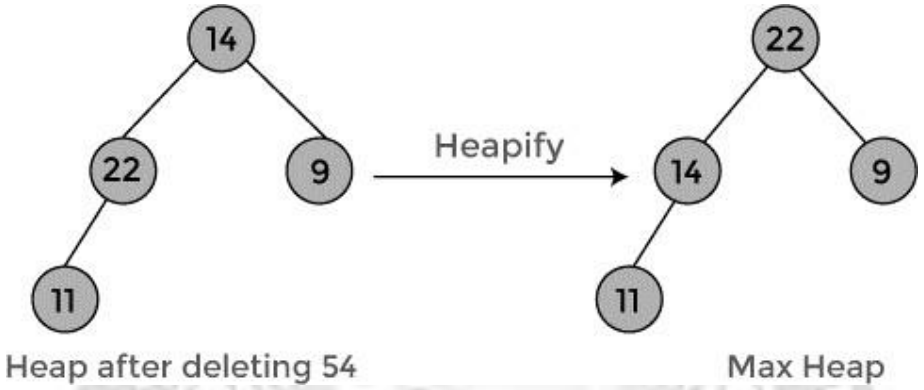
- In the next step, delete the root element (76) from the max heap again. To delete this node, swap it with the last node, i.e. (9). After deleting the root element, again heapify it to convert it into max heap.



- After swapping the array element 76 with 9 and converting the heap into max-heap, the elements of array are



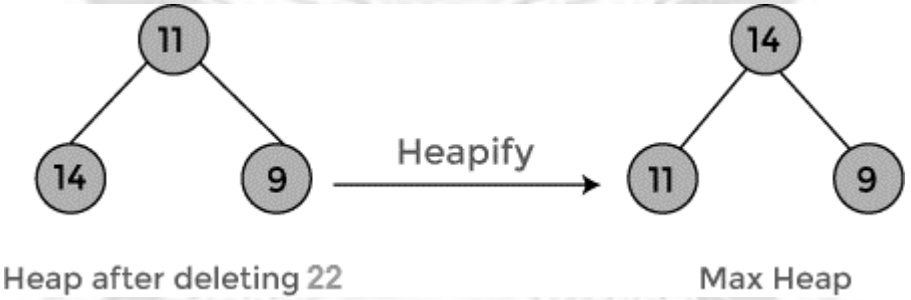
- In the next step, again delete the root element (54) from the max heap. To delete this node, swap it with the last node, i.e. (14). After deleting the root element, again heapify it to convert it into max heap.



- After swapping the array element 54 with 14 and converting the heap into max-heap, the elements of array are

22	14	9	11	54	76	81	89

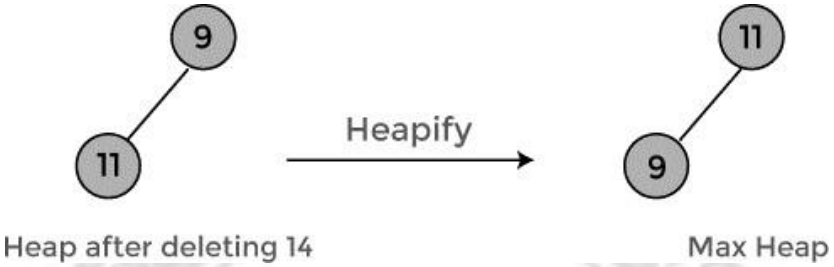
- In the next step, again delete the root element (22) from the max heap. To delete this node, swap it with the last node, i.e. (11). After deleting the root element, again heapify it to convert it into max heap.



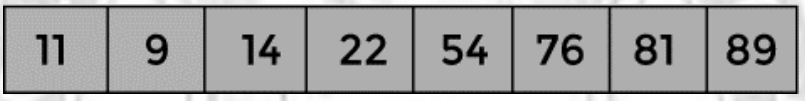
- After swapping the array element 22 with 11 and converting the heap into max-heap, the elements of array are

14	11	9	22	54	76	81	89

- In the next step, again delete the root element (14) from the max heap. To delete this node, swap it with the last node, i.e. (9). After deleting the root element, again heapify it to convert it into max heap.

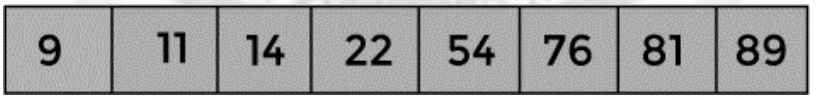


- After swapping the array element 14 with 9 and converting the heap into max-heap, the elements of array are

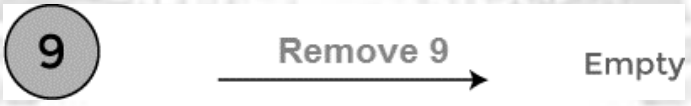


- In the next step, again delete the root element (11) from the max heap. To delete this node, swap it with the last node, i.e. (9). After deleting the root element, again heapify it to convert it into max heap.

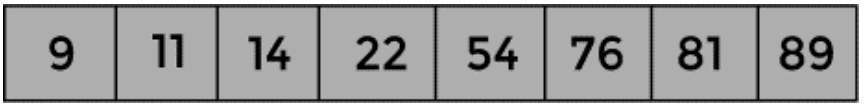
- After swapping the array element 11 with 9, the elements of array are



- Now, heap has only one element left. After deleting it, heap will be empty.



- After completion of sorting, the array elements are



Now, the array is completely sorted

Heapsort Complexity:

Time Complexity	
Best	O(n log n)
Worst	O(n log n)
Average	O(n log n)
Space Complexity	O(1)
Stability	No

Heap Sort Applications:

- Systems concerned with security and embedded systems such as Linux Kernel use Heap Sort.
- Because of the O(n log n) upper bound on Heapsort's running time and constant O(1) upper bound on its auxiliary storage.
- Although Heap Sort has O(n log n) time complexity even for the worst case, it doesn't have more applications (compared to other sorting algorithms like QuickSort, Merge Sort).

Example Program 5.3: Program for implementing Heap Sort#include <stdio.h>

```
/* function to heapify a subtree. Here 'i' is the
index of root node in array a[], and 'n' is the size of heap. */
void heapify(int a[], int n, int i)
{
  int largest = i; // Initialize largest as root
  int left = 2 * i + 1; // left child
  int right = 2 * i + 2; // right child
  // If left child is larger than root
  if (left < n && a[left] > a[largest])
    largest = left;
```

```
    // If right child is larger than root
    if (right < n && a[right] > a[largest])
        largest = right;
    // If root is not largest
    if (largest != i) {
        // swap a[i] with a[largest]
        int temp = a[i];
        a[i] = a[largest];
        a[largest] = temp;

        heapify(a, n, largest);
    }
}
/*Function to implement the heap sort*/
void heapSort(int a[], int n)
{
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(a, n, i);
    // One by one extract an element from heap
    for (int i = n - 1; i >= 0; i--) {
        /* Move current root element to end*/
        // swap a[0] with a[i]
        int temp = a[0];
        a[0] = a[i];
        a[i] = temp;
        heapify(a, i, 0);
    }
}
/* function to print the array elements */
```

```
void printArr(int arr[], int n)
{
   for (int i = 0; i < n; ++i)
   {
      printf("%d", arr[i]);
      printf(" ");
   }
}
int main()
{
   int a[] = {42, 8, 26, 39, 28, 23, 7};
   int n = sizeof(a) / sizeof(a[0]);
   printf("Before sorting array elements are -
   \n");printArr(a, n);
   heapSort(a, n);
   printf("\nAfter sorting array elements are -
   \n");printArr(a, n);
   return 0;
}
```

Output

```
Before sorting array elements are
42, 8, 26, 39, 28, 23, 7
After sorting array elements are
7, 8, 23, 26, 28, 39, 42
```