Previous (quick-sort.html) Next (comparison-of-sorting-methods.html)

# Heap Sort Algorithm

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.

## Step by Step Process

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.**

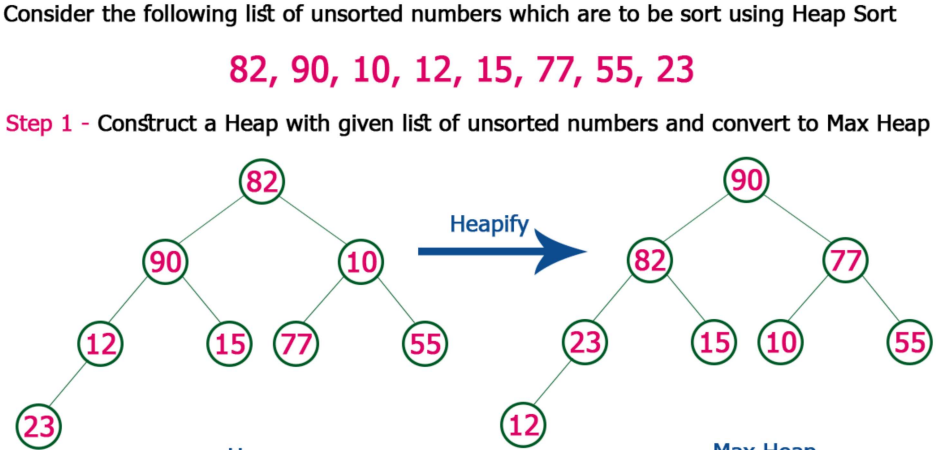
**Step 3 -** Delete the root element from Min Heap using **Heapify** method.

**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

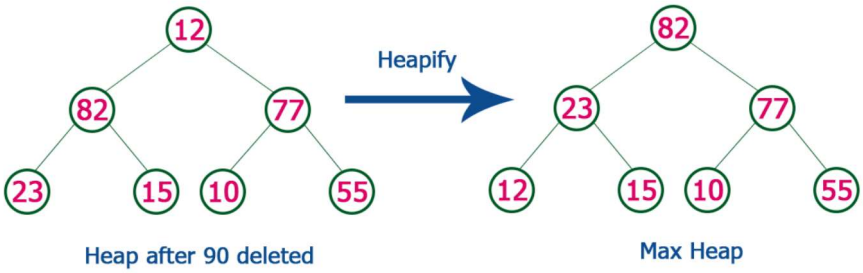


Heap Max Heap

list of numbers after heap converted to Max Heap

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

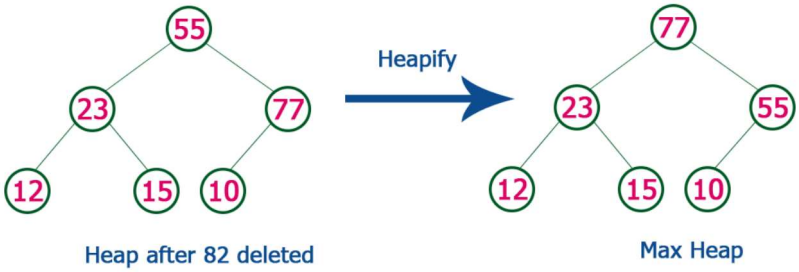
Step 2 - Delete root (**90**) from the Max Heap. To delete root node it needs to be swapped with last node (**12**). After delete tree needs to be heapify to make it Max Heap.




list of numbers after swapping 90 with 12.

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

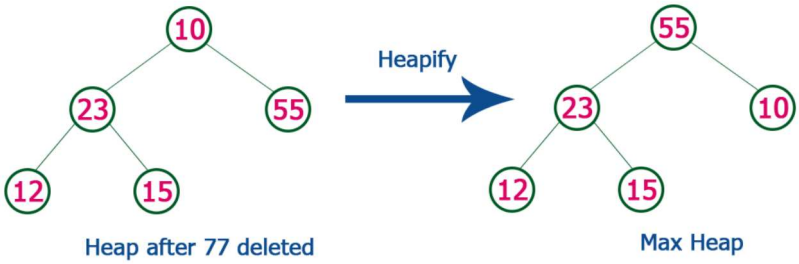
Step 3 - Delete root (**82**) from the Max Heap. To delete root node it needs to be swapped with last node (**55**). After delete tree needs to be heapify to make it Max Heap.




list of numbers after swapping 82 with 55.

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

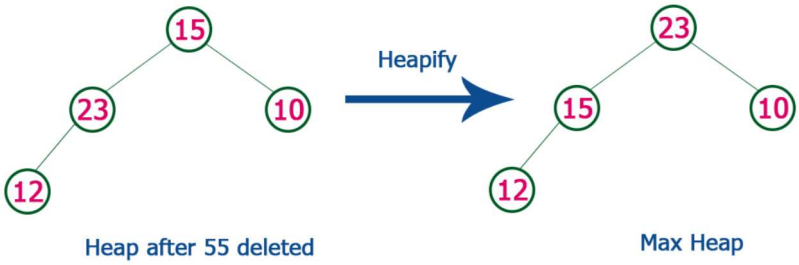
Step 4 - Delete root (**77**) from the Max Heap. To delete root node it needs to be swapped with last node (**10**). After delete tree needs to be heapify to make it Max Heap.




list of numbers after swapping 77 with 10.

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

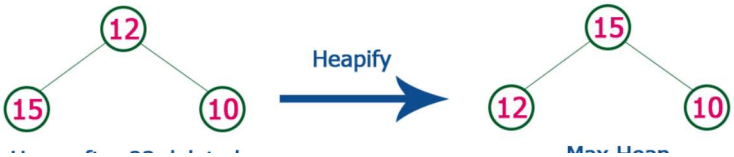
Step 5 - Delete root (**55**) from the Max Heap. To delete root node it needs to be swapped with last node (**15**). After delete tree needs to be heapify to make it Max Heap.




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.

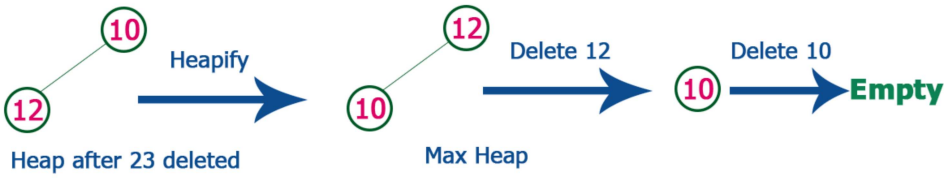


Heap after 23 deleted Max Heap

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.




Heap after 23 deleted 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

## Complexity of the Heap Sort Algorithm

To sort an unsorted list with 'n' number of elements, following are the complexities...

Worst Case : O(n log n)

Best Case : O(n log n)

Average Case : O(n log n)

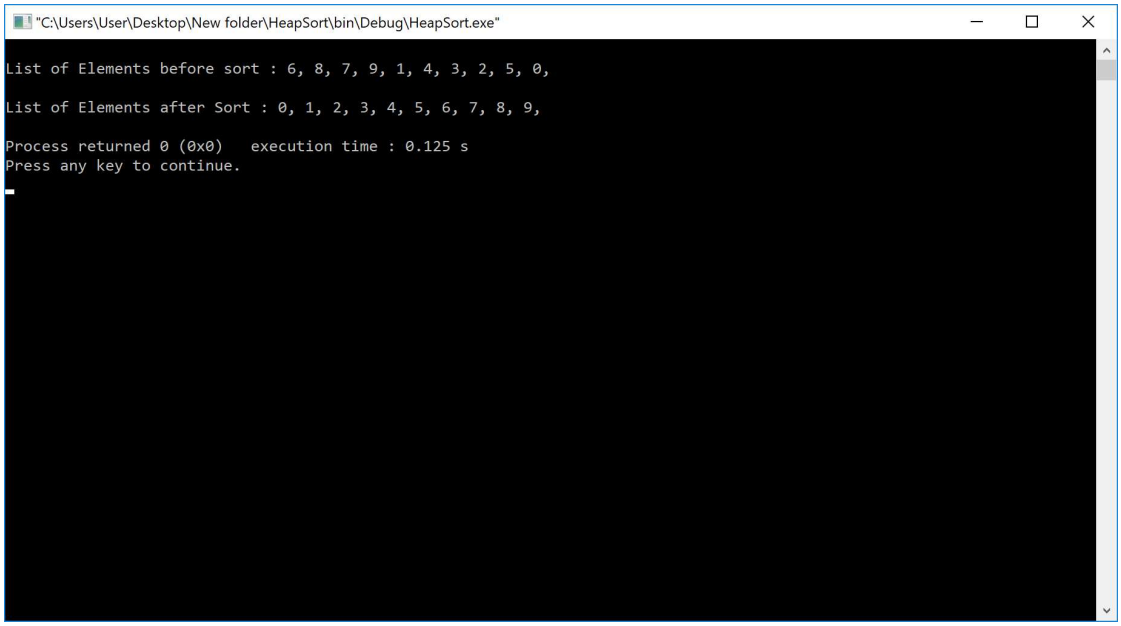
## Implementaion of Heap Sort Algorithm using C++ Programming Language

```
#include<iostream>
#include<cmath>

using namespace std;

int find_min_element_index(int a[] , int element , int left_child , int right_child){
        if(a[element] <= a[left_child] && a[element] <= a[right_child])
                return element;

        if(a[left_child] <= a[element] && a[left_child] <= a[right_child])
                return left_child;

        if(a[right_child] <= a[element] && a[right_child] <= a[left_child])
                return right_child;
}

int check_heap(int a[] , int n){

        int second_last_level = (int)log2(n);

        int check_elements_upto_index = pow( 2 , second_last_level+1)-2;
        for(int i=0; i<=check_elements_upto_index; i++){

                if(2*i+1 <= n-1 || 2*i+2 <= n-1){
                                if(2*i+1 <= n-1 && 2*i+2 <= n-1){
                                        if(a[i] <= a[2*i+1] && a[i] <= a[2*i+2])
                                                continue;
                                        else
                                                return 0;
                                }
                                else if(2*i+1 <= n-1){
                                                if(a[i] <= a[2*i+1])
                                                        continue;
                                                else
                                                        return 0;
                                }
                                else{
                                                if(a[i] <= a[2*i+2])
                                                        continue;
                                                else
                                                        return 0;
                                }
                        }
                        else{
                           break;
                        }
        }
        return 1;
}

void heapify(int a[] , int n){
        int heap_flag = 0;
        while(heap_flag == 0){
                int level = (int)log2(n);
                int ending_index_at_corresponding_level = pow(2 ,level+1)-2;
                for(int i = ending_index_at_corresponding_level;i >= 0 ;i--){
                                        if(2*i+1 <= n-1 || 2*i+2 <= n-1){
```

```
                                        if(2*i+1 <= n-1 && 2*i+2 <= n-1){
                                            int min_index = find_min_element_index
(a , i , 2*i+1 , 2*i+2);
                                                        if(min_index == i)
{
                                                                continue;
                                                        }
                                                        else{
                                                                int temp =
a[min_index];
                                                                a[min_inde
x] = a[i];
                                                                a[i] = tem
p;
                                                        }
                                                }
                                                else if( 2*i+1 <= n-1){
                                                        if(a[i] <= a[2*i+1])
                                                                continue;
                                                        else{
                                                                int temp =
a[i];
                                                                a[i] = a[2
*i+1];
                                                                a[2*i+1] =
temp;
                                                        }
                                                }
                                                else{
                                                        if(a[i] <= a[2*i+2])
                                                                continue;
                                                        else{
                                                                int temp =
a[i];
                                                                a[i] = a[2
*i+2];
                                                                a[2*i+2] =
temp;
                                                        }
                                                }
                                        }
                        }

                        heap_flag = check_heap(a , n);
                        if(heap_flag == 1){
                                cout << a[0] <<", ";
                                int temp = a[0];
                                a[0] = a[n-1];
                                a[n-1] = temp;
                                n = n-1;
                        }
                        if(n==1){
                                cout <<a[0] <<", ";
                                break;
                        }

                        heap_flag = check_heap(a , n);
                }
```

```
}


int main(){
        int list[] = {6,8,7,9,1,4,3,2,5,0};
        cout<<endl<<"List of Elements before sort: ";
        for(int i = 0; i<10; i++)
        cout<<list[i]<<", ";
        cout <<endl<<endl<<"Sorted Output as Follows : ";
        heapify(list , 10);
        cout<<endl;
}
```

## Output




Previous (quick-sort.html) Next (comparison-of-sorting-methods.html)