# Exercise 10 Sorting Methods

**10. a) Write a C++ Program for sorting integer numbers using Merge sort**

**b) Write a C++ program for bottom up heap with arrays using Heap sort.**

**Objective:** The objective of this exercise help you to perform sorting of integer numbers using Merge sort and Bottom up heap with arrays using Heap sort method.

**Procedure and description:**

**a) Merge Sort:**

Merge sort is a comparison-based sorting algorithm. Implementations uses divide and conquer algorithm. Merge Sort breaks the data into small data sets, sorts those small sets, and then merges the resulting sorted lists together.

**Algorithm:**

A: Array elements
p: index value of array starting element
r: index value of array ending element

The steps for this exercise are given below:

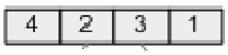
```
Step 1: Merge Sort (A, p, r)
Step 2: If p<r
Step 3: Then q<--[(p+r)/2]    //dividing array into two parts
Step 4: Merge Sort (A, p,q)
Step 5: Merge Sort (A, q+1, r)
Step 6: Merge (A, p, q, r)
```

**Expected Output:**

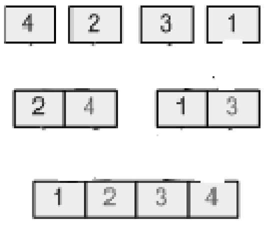
**Input:** After executing program, enter input following things.
Enter size of array, enter integer numbers.

**Example:**

Enter the size of array: 4
Enter integer numbers:



**Output:** Merge Sort breaks the data into small data sets, sorts those small sets, and then merges the resulting sorted lists together. For better understanding see below pictorial representation.



**b) Heap Sort**

**Procedure and description:**

Heap sort often uses for very large arrays which are in unsorted state. The largest or smallest element of the list is determined and then placed at the end or beginning of the list. This process continues with the rest of the list, and is accomplished by using a data structure called a heap, which is a special type of binary tree.

It consists of two types of sorting

1. Top down heap
2. Bottom up heap

**Algorithm:** The steps for bottom up heap are given below:
H – Heap value declaration
[N/2] - Give the heap of n nodes, what’s the index of the last parent
K – The root of the sub tree to be heapified (more extreme (greater) than or equal to the keys of its children)
V – The key of the root
J – Index
Step 1: start
Step 2: declare bottom up heap value (H[1…n])
Step 3: declare for loop for last parent value like down to 1, heapify the sub tree rooted at **I** **//heapify-**(a group of elements placed or thrown one on another)

```
        For i←[n/2]
Step 4:   condition k← I; v← H[k] heap ← false
Step 5: apply while loop for not heap case
          2*k ≤ n means do
          J ← 2*k
          If j < n
          If H[j] < H[j +1] j ← j+1
          If v ≥ H[ j ]
          Heap ← true
Step 6: exchange the key with the key of the larger child using
             else statement
            Else H[ k ] ← H [ j ];
             K ← j
Step 7: check the condition the sub tree to be heapified and key is in root.
          H [ k ] ← v
Step 8: stop
```

**Expected Output:**

```
Enter the elements for heap
 4   1   3   2   16   19   10   14   8   7

After heap the values in bottom up method
16   14   10   8   7   9   3   2   4   1
```