6. Write C++ programs for implementing the following sorting methods:

a) Merge sort

b) Heap sort

a) MERGE SORT:

```
#include <iostream>
using namespace std;

// Function to merge two halves of the array
void merge(int arr[], int left, int mid, int right) {
  int n1 = mid - left + 1;
  int n2 = right - mid;

  int leftArr[n1], rightArr[n2];

  // Copy data to temporary arrays
  for (int i = 0; i < n1; i++)
    leftArr[i] = arr[left + i];
  for (int i = 0; i < n2; i++)
    rightArr[i] = arr[mid + 1 + i];

  // Merge the temp arrays back into arr[l..r]
  int i = 0; // Initial index of first subarray
  int j = 0; // Initial index of second subarray
  int k = left; // Initial index of merged subarray
  while (i < n1 && j < n2) {
    if (leftArr[i] <= rightArr[j]) {
      arr[k] = leftArr[i];
      i++;
    } else {
      arr[k] = rightArr[j];
```

```
            j++;
        }
        k++;
    }

    // Copy the remaining elements of leftArr[], if any
    while (i < n1) {
        arr[k] = leftArr[i];
        i++;
        k++;
    }

    // Copy the remaining elements of rightArr[], if any
    while (j < n2) {
        arr[k] = rightArr[j];
        j++;
        k++;
    }
}

// Function that recursively sorts the array
void mergeSort(int arr[], int left, int right) {
    if (left >= right)
        return;

    int mid = left + (right - left) / 2;

    mergeSort(arr, left, mid);    // Sort first half
    mergeSort(arr, mid + 1, right); // Sort second half
    merge(arr, left, mid, right);  // Merge the sorted halves
}
```

```
int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Given array: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;

    mergeSort(arr, 0, n - 1);

    cout << "Sorted array: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;

    return 0;
}
```

b)HEAP SORT:

```
#include <iostream>
using namespace std;

// Function to heapify a subtree rooted with node i
void heapify(int arr[], int n, int i) {
    int largest = i;  // Initialize largest as root
    int left = 2 * i + 1;  // left = 2*i + 1
    int right = 2 * i + 2; // right = 2*i + 2

    // If left child is larger than root
    if (left < n && arr[left] > arr[largest])
        largest = left;

    // If right child is larger than largest so far
    if (right < n && arr[right] > arr[largest])
        largest = right;

    // If largest is not root
    if (largest != i) {
        swap(arr[i], arr[largest]);

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}

// Main function to do heap sort
void heapSort(int arr[], int n) {
    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // One by one extract elements from heap
    for (int i = n - 1; i > 0; i--) {
        // Move current root to end
        swap(arr[0], arr[i]);
        // call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Given array: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;

    heapSort(arr, n);

    cout << "Sorted array: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;

    return 0;
}
```