Question:-Implement Quick Sort, Merge Sort and Heap Sort

```
#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

class Arrange

{

   public:

   int size;

   vector<int>v;

   Arrange(int size)

   {

      this->size=size;

      cout<<"Enter the array ";

      v.resize(size);//this line is important

      for(int i=0;i<size;i++)

      {

         int temp;

         cin>>temp;

         v[i]=temp;

      }

   }

   int QuickSort(int i,int j,int pivotindex,int pivot)

   {

      int size=j;
```

```
    while(i<j)

    {

        while(v[i]<=pivot&&i<j)

        {

            i++;

        }

        while(v[j]>pivot)

        {

            j--;

        }

        if(i<j)

        {

            swap(v[i],v[j]);

        }

    }

    swap(v[j],v[pivotindex]);

    return j;

}

void qsort(int lowerbound,int upperbound)

{

    if(lowerbound<upperbound)

    {

        int index=QuickSort(lowerbound,upperbound,lowerbound,v[lowerbound]);

        qsort(lowerbound,index-1);

        qsort(index+1,upperbound);
    }

}

void Mergesort(vector<int>&v,int start,int end,int mid)

{

    int size1=mid-start+1;

    int size2=end-mid;

    int arr[size1];

    int arr1[size2];

    int s=start;

    int m=mid+1;

    int i=0,j=0;

    //here i copy array

    for(int i=0;i<size1;i++)

    {

        arr[i]=v[s];

        s++;

    }

    for(int i=0;i<size2;i++)

    {

        arr1[i]=v[m];

        m++;

    }

    i=0;

    j=0;

    s=start;
```

```
//here i checking array and arrange in sorted manner
while(i<size1&&j<size2)
{
   if(arr[i]<arr1[j])
   {
      v[s]=arr[i];
      i++;
      s++;
   }
   else
   {
      v[s]=arr1[j];
      j++;
      s++;
   }
}
//if any array left than traversing than we traverse that and arrange thoes no's in sorted manner
while(i<size1)
{
   v[s]=arr[i];
   i++;
   s++;
}
while(j<size2)
{
        v[s]=arr1[j];

        j++;

        s++;

    }

}

void msort(int start,int end)

{

    if(start<end)

    {

        int mid=(start+end)/2;

        msort(start,mid);

        msort(mid+1,end);

        Mergesort(v,start,end,mid);

    }


}

void Hdelete(int start,int end)

{

    int e=end;

    int s=start;

    swap(v[start],v[end]);

    while(2*s<end)

    {

        int lchild=v[2*s];

        int index=0;
        if((2*s)+1<end)
        {
            int rchild=v[(2*s)+1];
            int m=max(lchild,rchild);
            if(m==lchild)
            {
                index=2*s;
            }
            else
            {
                index=(2*s)+1;
            }
            swap(v[index],v[s]);
            s=index;
        }
        else{
            index=2*s;
            if(v[s],v[2*s])
            {
                swap(v[index],v[s]);
                s=index;
            }
        }
    }
}
```

```
void hepify(int start,int end)
{
    int s=start;
    int e=end;
    while(e>1)
    {
        int end1=e;
        while(v[end1]>v[end1/2]&&end1>1)
        {
            if(v[end1]>v[end1/2])
            {
                swap(v[end1],v[end1/2]);
            }
            end1=end1/2;
        }
        e--;
    }
}
void Hsort(int start,int end)
{
    start=start+1;
    hepify(start,end);
    while(end>=2)
    {
        Hdelete(start,end);
```

```
            end--;
        }
    }
    void print()
    {
        cout<<"Sorted array is "<<endl;
        for(int i=0;i<size;i++)
        {
            cout<<v[i]<<" ";
        }
    }
};
int main()
{
    int size;
    cout<<"Enter the size "<<endl;
    cin>>size;
    Arrange a(size);
    int lowerbound=0;
    int upperbound=size-1;
    cout<<"Enter your choice ";
    int choice=0;
    cout<<"Enter 1 for quick sort "<<endl;
    cout<<"Enter 2 for merge sort "<<endl;
    cout<<"Enter 3 for Heap sort "<<endl;
    cin>>choice;
    switch(choice)
    {
        case 1:
        {
            a.qsort(lowerbound,upperbound);
            a.print();
            break;
        }
        case 2:
        {
            a.msort(lowerbound,upperbound);
            a.print();
            break;
        }
        case 3:
        {
            a.Hsort(lowerbound,upperbound);
            a.print();
            break;
        }
    }
}
```

Input:-size=11

//using quick sort

```
1 3 5 4 6 13 10 9 8 15 17

1 3 4 5 6 8 9 10 13 15 17

Input:-size=11

//using merge sort

1 3 5 4 6 13 10 9 8 15 17

1 3 4 5 6 8 9 10 13 15 17

Input:-size=11

//using Heap sort

-1 1 3 5 4 6 13 10 9 8 15 17

-1 1 3 4 5 6 8 9 10 13 15 17
```