```
int findPartitionIndex(int * arr,int si,int ei)
{
	int count=0;
	for (int i=si;i<=ei;i++)
	{
		if (arr[i] < arr[si])
		{
			count++;
		}
	}

	// putting the element at si in the correct postiton
	int elementOfParition=arr[si];
	int indexOfPartition=si+count;

	int temp=arr[si];
	arr[si]=arr[si+count];
	arr[si+count]=temp;

	// placing all the elements greater than the si
	int i=si;
	int j=ei;

	while (i<=si+count && j>=si+count)
	{
		if (arr[i]<elementOfParition && arr[j]>elementOfParition)
		{
			i++;
			j--;
		}

		else if(arr[i]<elementOfParition)
		{
			i++;
		}

		else if (arr[j]>=elementOfParition)
		{
			j--;
		}

		else
		{
			int temp=arr[i];
			arr[i]=arr[j];
			arr[j]=temp;

			i++;
			j--;
		}
	}

	return si+count;
}

void quickSortHelper(int * arr,int si,int ei,int & count)
{
	// Base Case
```

```
      if (si>=ei)
      {
            return;
      }

      count++;

     // Our Calculation
      int indexOfPartition=findPartitionIndex(arr,si,ei);

      // Inuction Hypothesis
      quickSortHelper(arr,si,indexOfPartition-1,count);
      quickSortHelper(arr,indexOfPartition+1,ei,count);
}

void quickSort(int * arr,int n,int & count)
{
      quickSortHelper(arr,0,n-1,count);

      for (int i=0;i<n;i++)
      {
            cout<<arr[i]<<" ";
      }

      cout <<endl;
}

void heapSort(int * arr,int n)
{
  minPriorityQueue minPQ;
  for (int i=0;i<n;i++)
  {
     minPQ.insert(arr[i]);
  }

  while(!minPQ.isEmpty())
  {
    cout<<minPQ.removeMin()<<" ";
  }

  cout<<endl;
}
```