AOA PRACTICAL LAB 1

Name: Brendan Lucas, Roll No:8953, Div: SE Comp B

Source Code:

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

int partition(int *arr,int l,int r)
{
	//Declaring variables
	int pivot,low,high,temp,sort=0,rev_sort=0,i;

	//Select first element as pivot element
	pivot=arr[l];

	//set low as left index index
	low=l;

	//set high as right index
	high=r;

	//while low index is less than high
	while(low<high)
	{
		//while greater element than pivot is not found
		while(arr[low]<=pivot&&low<r)
		{
			//check if array is already sorted
			if(arr[low]<=arr[low+1])
			{
				sort++;
			}
			//check if array is sorted in reverse order
			else if(arr[low]>=arr[low+1])
			{
				rev_sort++;
			}
			//increment to next index
			low++;
		}
		//while smaller element than pivot is not found
		while(arr[high]>pivot&&high>=l)
		{
			//check if array is already sorted
			if(arr[high-1]<=arr[high])
			{
				sort++;
			}
			//check if array is sorted in reverse order
			else if(arr[high-1]>=arr[high])
			{
				rev_sort++;
			}
			//decrement to next index
			high--;
		}
		//two elements of are to be swapped
		if(low<high)
		{
			temp=arr[low];
			arr[low]=arr[high];
			arr[high]=temp;
		}
	}
	//if array is already sorted
	if(sort==r-l+1)
	{
		return -1;
	}

	//if array is sorted in reverse order
	if(rev_sort==r-l)
	{
```

```
                        //Reverse the array
                        for(i=0;i<=(r-l)/2;i++)
                        {
                                temp=arr[l+i];
                                arr[l+i]=arr[r-i];
                                arr[r-i]=temp;
                        }
                        return -1;
                }
                //place pivot element in proper position
                //cout<<r<<" "<<l<<"\n";
                arr[l]=arr[high];
                arr[high]=pivot;
                return high;
}

/int*/void quicksort(int *arr,int l,int r)
{
                //Declaring variables
                int m;

                //if partition has one or more than one elements
                if(r>l)
                {
                        //Take index of partition
                        m=partition(arr, l, r);

                        //if array is sorted, return
                        if(m==-1)
                        {
                                return;
                        }

                        //Recursive call for first partition
                        quicksort(arr,l,m-1);

                        //Recursive call for second partition
                        quicksort(arr,m+1,r);
                }

//              return arr;
}

int main(void)
{

                //Declering variables
                int n,*arr,i;

                //Asking for No. of elements in array
                cout<<"Enter the no of elements in array:-\n";
                cin>>n;

                //Creating array of given size using malloc
                arr=(int*)malloc(sizeof(int)*n);

                //Taking input of elements of array
                cout<<"Enter the array\n";
                for(i=0;i<n;i++)
                {
                        cin>>arr[i];
                }

                //Sorting the array
                //bubblesort(arr, n);
                //insertionsort(arr, n);
                quicksort(arr,0, n-1);

                //Displaying the final result
                cout<<"\nThe Sorted array is:- \n";
                for(i=0;i<n;i++)
                {
                        cout<<arr[i]<<" ";
                }
                return 0;
}
```

Page No.
Date

Quick sort:-

code:-

```
int partition (int *arr, int l, int r)
{
    pivot = arr[l];
    low = l;
    high = r;
    while (low < high)
    {
        while (arr[low] <= pivot && low < r)
        {
            if (arr[low] <= arr[low +
            low++;
        }
        while (arr[high] > pivot && high > l)
        {
            high--;
        }
        if (low < high)
        {
            temp = arr[low];
            arr[low] = arr[high];
            arr[high] = temp;
        }
    }
    arr[l] = arr[high];
    arr[high] = pivot;
    return high;
}
```

```
void quicksort(int *arr, int l, int r)
{
    int m;
    if(r > l)
    {
        m = partition(arr, l, r);
        quicksort(arr, l, m-1);
        quicksort(arr, m+1, r);
    }
}
```

Time Analysis :-

(1) Best Case :-

The best case takes place when the pivot element is placed in the center of the array ~~and~~ partition giving us two arrays of size $\frac{n}{2}$.

The recurrence eqn is

$$T(n) = T(i) + T(n-1-i) + n.$$

for $(i = \frac{n}{2})$

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + n.$$

$$= 2T\left(\frac{n}{2}\right) + n.$$

Here, $a = 2$ & $b = 2$.

$\therefore n^{\log_2 2} = n^1 = n$.

$\therefore$ Case 2 is applicable.

$T(n) = O(n \log_2 n)$.

(i) Worst case :-

The worst case takes place when data is already sorted in ascending or descending order. This gives us two arrays of size 0 & (n-1)

$$T(n) = T(i) + T(n-1-i) + n$$
$$= T(0) + (T(0) + T(n-2) + (n-1)) + n$$
$$\vdots$$
$$T(n) = 1 + 2 + \cdots\cdots (n-1) + n$$

Similarly for descending

$$T(n) = (n-1) + (n-2) + \cdots\cdots + 2 + 1$$

$$\therefore \text{Total complexity} = \frac{n(n-1)}{2}$$
$$= 0.5n^2 - 0.5n$$
$$\therefore T(n) = O(n^2)$$