Name: - Sakshi D Lonare
Roll no :- 2021300069

Aim:
Each student have to generate random 100000 numbers using rand() function and use this input as 10 blocks of 10000 integer numbers to Merge sort algo.

Algorithm:

1. Start
2. Create an array of length 100000.
3. Input 100000 random integers into both the arrays using : rand()%100000
4. Store the generated random numbers in a text file.
5. Perform insertion sort and selection sort of all the elements in groups of 10000, then 20000, ..so on till end.
6. Print the time taken for each sorting using clock() function.
7. Stop
8. Step 1: Start
   Step 2: Declare an array and left, right, mid variable
   Step 3: Perform merge function.
   mergesort(array,left,right)
   mergesort (array, left, right)
   if left > right
   return
   mid= (left+right)/2
   mergesort(array, left, mid)
   mergesort(array, mid+1, right)
   merge(array, left, mid, right)
   Step 4: Stop
9. QUICKSORT:-

Step 1 − Choose the highest index value has pivot
Step 2 − Take two variables to point left and right of the list excluding pivot
Step 3 − left points to the low index
Step 4 − right points to the high
Step 5 − while value at left is less than pivot move right
Step 6 − while value at right is greater than pivot move left
Step 7 − if both step 5 and step 6 does not match swap left and right
Step 8 − if left ≥ right, the point where they met is new pivot

Program:

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<time.h>
void merge(int arr[], int l, int m, int r,int* count)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
```

```
    int L[n1], R[n2];

    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
            *count+=1;
        }
        k++;
        *count+=1;
    }

    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r,int* count)
{
    if (l < r)
    {
        int m = l + (r - l) / 2;
```

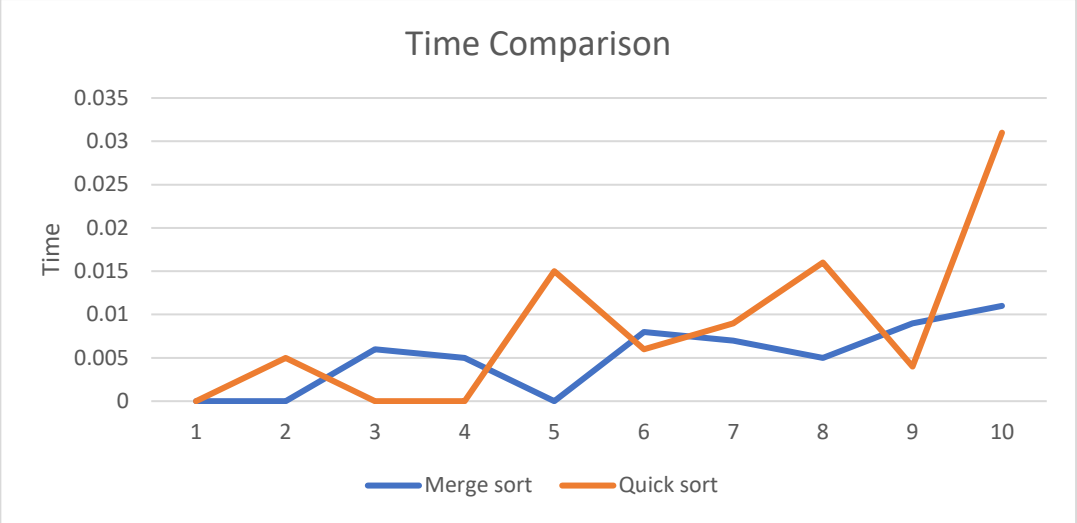
```
        mergeSort(arr, l, m,count);
        mergeSort(arr, m + 1, r,count);

        merge(arr, l, m, r,count);
    }
}

void swap(int* a,int* b)
{
    int temp=*a;
    *a=*b;
    *b=temp;
}

int partition(int arr[],int low,int high, int* cmp)
{
    int p=arr[high];
    int i=low-1;

    for(int j=low;j<=high-1;j++)
    {
        (*cmp)++;
        if(arr[j]<p)
        {
            i++;
            swap(&arr[i],&arr[j]);
        }
    }
    swap(&arr[i+1],&arr[high]);
    return (i+1);
}

void quickSort(int arr[],int low,int high,int* cmp)
{
    if(low<high)
    {
        //if pivot is at the right place
        int p=partition(arr,low,high,cmp);
        quickSort(arr,low,p-1,cmp);
        quickSort(arr,p+1,high,cmp);
    }
}

int main(){
FILE *f;
FILE *ans1;
FILE *time1;
FILE *ans2;
FILE *time2;
```

```
int count=0;
int cmp=0;

clock_t start, end, start1, end1;
f = fopen("demo.txt","w");
ans1 = fopen("ans.txt","w");
time1 = fopen("time.txt","w");
ans2 = fopen("ans1.txt","w");
time2 = fopen("time1.txt","w");

int arr[100000],arr1[100000];
for(int i=0;i<100000;i++){
int x=rand()%100000;
//printf("%d\n",x);
fprintf(f,"%d\n",x);
arr[i]=x;
arr1[i]=x;
}

for(int i=1;i<=10;i++)
{
start = clock();
mergeSort(arr,0,i*10000,&count);
end = clock();
double time_taken = (double)(end - start) / (double)(CLOCKS_PER_SEC);
printf("Time taken for %d elements to sort using mergesort:%fs \n",i*10000,time_taken);
fprintf(time1,"%f\n",time_taken);
}
printf("\nNo of comparisons:- %d\n",count);


for(int i=1;i<=10;i++)
{
   start = clock();
   quickSort(arr1,0,i*10000,&cmp);
   end= clock();
   double time_taken1 = (double)(end - start) / (double)(CLOCKS_PER_SEC);
   printf("Time taken for %d elements to sort using quicksort:%fs \n",i*10000,time_taken1);
   fprintf(time2,"%f\n",time_taken1);
}
printf("\nNo of comparisons:- %d\n",cmp);

for(int i=1;i<=10000;i++){
fprintf(ans1,"Sorted arr %d\n",i);
for(int j=0;j<i*10000;j++){
fprintf(ans1,"%d\n",arr[j]);
}
}
```

```
for(int i=1;i<=10000;i++){
fprintf(ans2,"Sorted arr %d\n",i);
for(int j=0;j<i*10000;j++){
fprintf(ans2,"%d\n",arr1[j]);
}
}

fclose(f);
fclose(ans1);
fclose(time1);
fclose(ans2);
fclose(time2);
return 0;
}
```

Graph & observation:

```
Time taken for 10000 elements to sort using mergesort:0.000000s
Time taken for 20000 elements to sort using mergesort:0.000000s
Time taken for 30000 elements to sort using mergesort:0.000000s
Time taken for 40000 elements to sort using mergesort:0.013000s
Time taken for 50000 elements to sort using mergesort:0.000000s
Time taken for 60000 elements to sort using mergesort:0.008000s
Time taken for 70000 elements to sort using mergesort:0.008000s
Time taken for 80000 elements to sort using mergesort:0.001000s
Time taken for 90000 elements to sort using mergesort:0.015000s
Time taken for 100000 elements to sort using mergesort:0.005000s

No of comparisons:- 6001708
Time taken for 10000 elements to sort using quicksort:0.000000s
Time taken for 20000 elements to sort using quicksort:0.000000s
Time taken for 30000 elements to sort using quicksort:0.011000s
Time taken for 40000 elements to sort using quicksort:0.000000s
Time taken for 50000 elements to sort using quicksort:0.011000s
Time taken for 60000 elements to sort using quicksort:0.008000s
Time taken for 70000 elements to sort using quicksort:0.010000s
Time taken for 80000 elements to sort using quicksort:0.010000s
Time taken for 90000 elements to sort using quicksort:0.010000s
Time taken for 100000 elements to sort using quicksort:0.031000s

No of comparisons:- 29312695
```



Here we can see that the running time of quicksort is less for inputs / sizes of larger values. Hence as compared to mergesort, Quicksort is useful as it is fast in such cases. We can also notice the count in the comparisons made while sorting , the merge sort is seen to have made the lowest comparisons

Conclusion: From the above experiment I learnt to about the running times of merge sort and quick sort , and how effectively Quicksort works with large input size.

.