Name	Ronak Thakar
UID no.	2021700066
Experiment No.	2

AIM:	Experiment based on divide and conquer approach.

Program 1

PROBLEM STATEMENT:

Each student have to generate random 100000 numbers using rand() function and use this input as 1000 blocks of 100,200,300,...,100000 integer numbers to Quicksort algorithm.

ALGORITHM/ THEORY:

It picks an element called as pivot, and then it partitions the given array around the picked pivot element. It then arranges the entire array in two sub-array such that one array holds values that are smaller than the specified value (Pivot), and another array holds the values that are greater than the pivot. The Divide and Conquer steps of Quicksort perform following functions.

- Divide: In Divide, first pick a pivot element. After that, partition or rearrange the array into two sub-arrays such that each element in the left sub-array is less than or equal to the pivot element and each element in the right sub-array is larger than the pivot element.
- Conquer: Recursively, sort two subarrays with Quicksort.
- Combine: Combine the already sorted array.

Worst case complexity is O(n^2).

Algorithm:

Step 1: Declare an array and it's left and right index.

Step 2: Declare the leftmost element as the pivot element(X) and arrange the array such that elements to the left of X are smaller then X and the elements to the right of X are greater than X. Now X is at its perfect position.

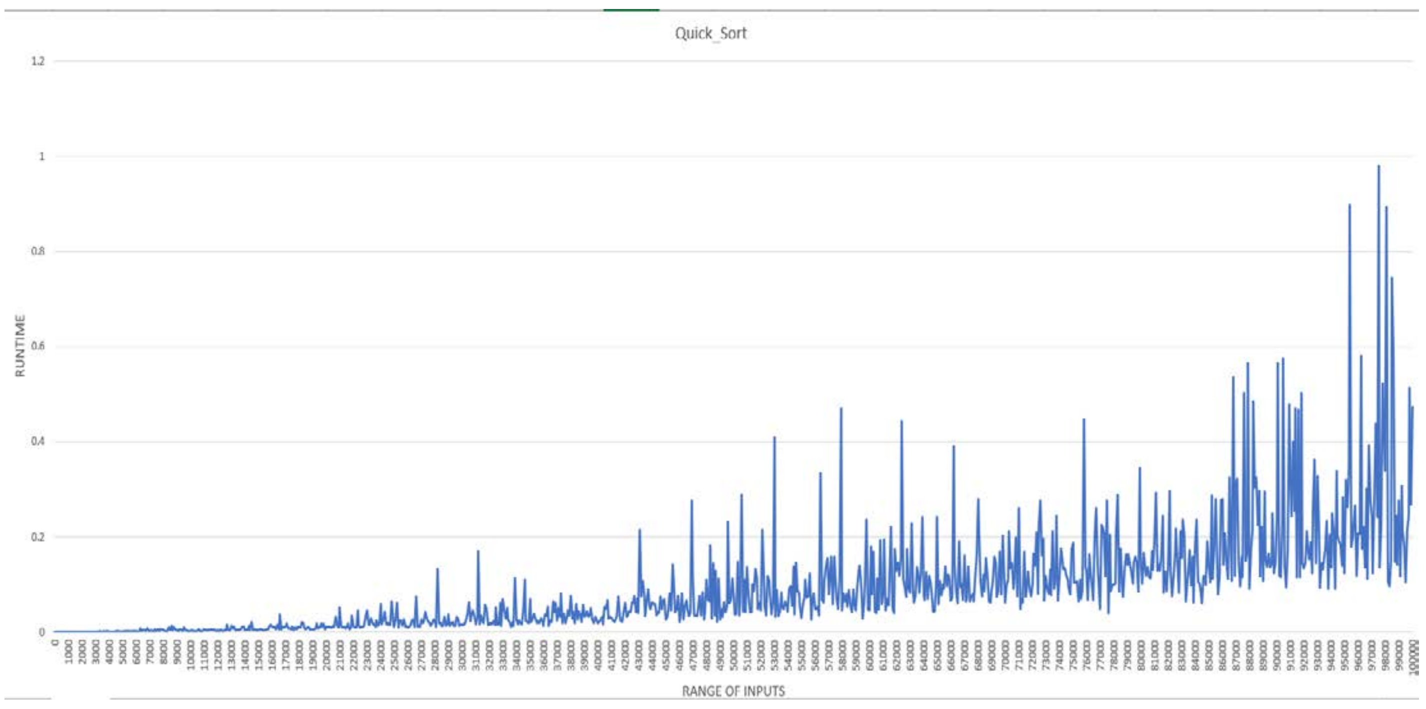
Step 3: use the index of X and call quick_sort() recursively and send the two unsorted parts of array to repeat the step 2 and 3.

Step 4: this process will end when all the array elements are sorted and when l is > or = h.

PROGRAM:

```
#include<stdio.h>
#include<time.h>
void swap(int arr[],int l,int h){
    int temp = arr[l];
    arr[l] = arr[h];
    arr[h] = temp;
}
int partition(int arr[], int l, int h){
    int X = arr[l],i = l;
    for(int j = l+1;j <= h;j++){
        if(arr[j]<X){
          i++;
          swap(arr,i,j);
        }
    }
    swap(arr,l,i);
    return(i);
}
void quick_sort(int arr[],int l,int h){
    if(l<h){
        int X = partition(arr,l,h);
        quick_sort(arr,l,X-1);
        quick_sort(arr,X+1,h);
    }
}
int main(){
    int arr[100000],arr2[100000],y=0;
    clock_t t1,t2;
    FILE *f;
    f = fopen("num.txt","r");
    for(int i = 0;i<100000;i++){
       fscanf(f,"%d",&arr[i]);
       arr2[i] = arr[i];
    }
    fclose(f);
    while(y < 100000){
        y +=100;
        t1 = clock();
        quick_sort(arr,0,y-1);
        t2 = clock();
        double total = ((double)(t2 - t1))/CLOCKS_PER_SEC;
        printf("%lf \n",total);
       for(int i=0;i<y;i++){
            arr[i] = arr2[i];
       }
    }
}
```

RESULT: The result of the above program is as follows:-



Program 2	
PROBLEM STATEMENT:	Each student have to generate random 100000 numbers using rand() function and use this input as 1000 blocks of 100,200,300,...,100000 integer numbers to Merge sorting algorithms.
ALGORITHM/ THEORY:	Merge sort is similar to the quick sort algorithm as it uses the divide and conquer approach to sort the elements. It divides the given list into two equal halves, calls itself for the two halves and then merges the two sorted halves. We have to define the merge() function to perform the merging. The sub-lists are divided again and again into halves until the list cannot be divided further. Then we combine the pair of one element lists into two element lists, sorting them in the process. The sorted two-element pairs are merged into the four-element lists, and so on until we get the sorted list. Worst case complexity is O(nlogn). Algorithm: Step 1: Declare an array, its left and right variables(indexes). Step 2: Calculate the mid index using the left and right. Step 3: If left > right Then return Divide the array into two halves using the mid index and left and right. And call the merge_sort() recursively. Step 4: Repeat step 3 till the moment that the function can no longer be divided and then start the merging of the small arrays into a big one.

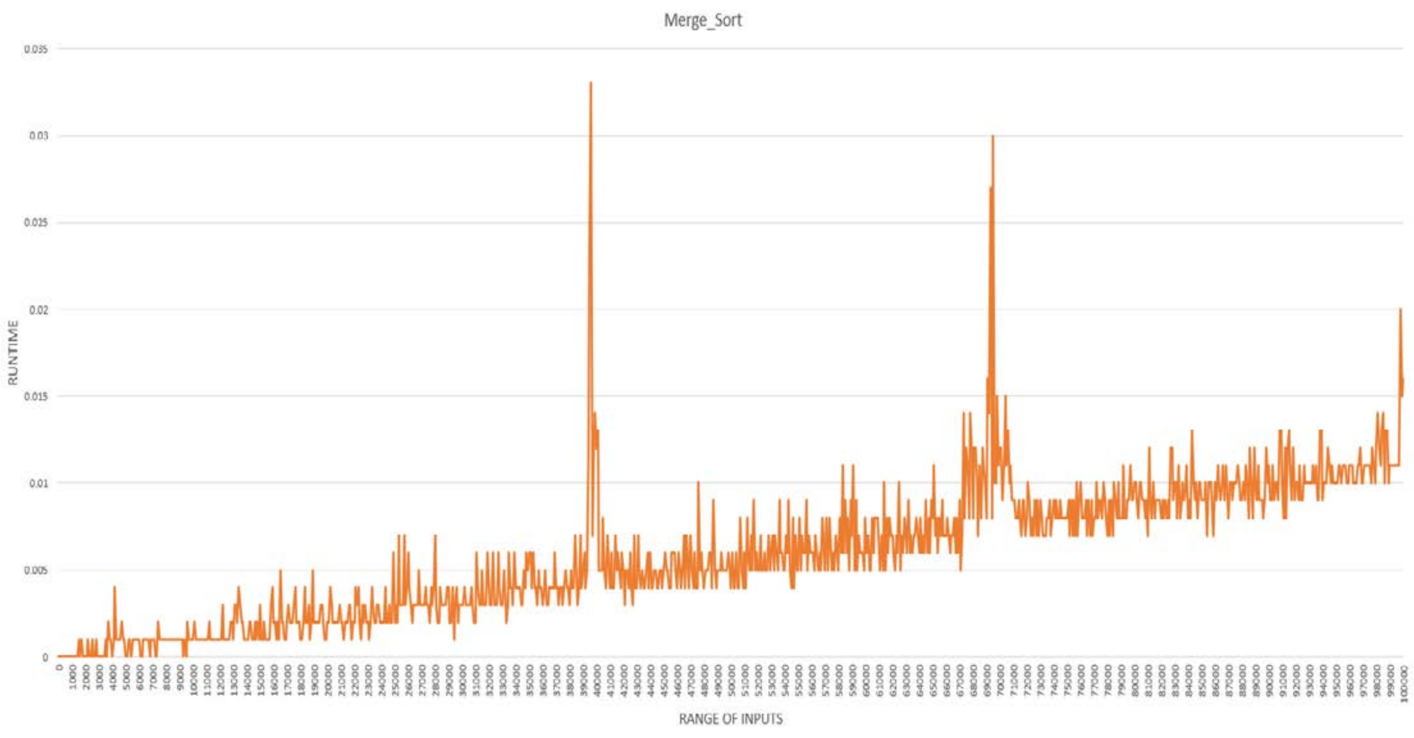
PROGRAM:	`#include<stdio.h>`
	`#include<time.h>`
	`int merge(int arr[],int l,int m,int r){`
	`    int i= 0,j = 0,k = 0,n1,n2;`
	`    n1 = m - l + 1;`

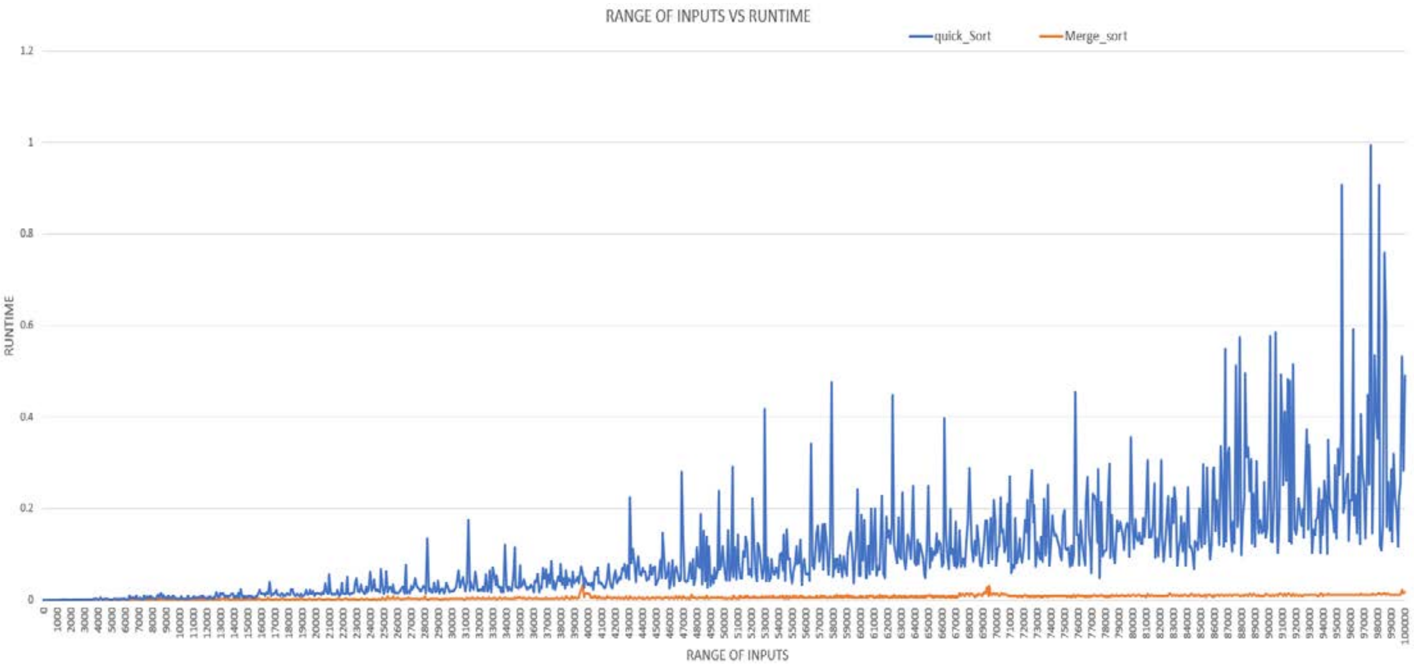
```
    n2 = r - m;
    int left[n1],right[n2];
    for(int x = 0;x<n1;x++)
      left[x] = arr[l + x];
    for(int x = 0;x<n2;x++)
      right[x] = arr[m + 1 + x];
    k = l;
    while(i<n1 && j<n2){
      if(left[i]<=right[j]){
       arr[k] = left[i];
       i++;
      }
      else{
       arr[k] = right[j];
       j++;
      }
      k++;
    }
    while(i<n1){
       arr[k] = left[i];
       i++;
       k++;
    }
    while(j<n2){
       arr[k] = right[j];
       j++;
       k++;
    }
}
void merge_sort(int arr[],int l,int r){
    if(l<r){
     int m = l + (r-l)/2;
     merge_sort(arr,l,m);
     merge_sort(arr,m+1,r);

     merge(arr,l,m,r);
    }
}
int main(){
    int arr[100000],arr2[100000],y=0;
    clock_t t1,t2;
    FILE *f;
    f = fopen("num.txt","r");
    for(int i = 0;i<100000;i++){
      fscanf(f,"%d",&arr[i]);
      arr2[i] = arr[i];
    }
    fclose(f);
    while(y < 100000){
       y +=100;
       t1 = clock();
       merge_sort(arr,0,y-1);
       t2 = clock();
```

```
        double total = ((double)(t2 - t1))/CLOCKS_PER_SEC;
        printf("%lf \n",total);
        for(int i=0;i<y;i++){
            arr[i] = arr2[i];
        }
    }
}
```

RESULT: The result of the above program is as follows:-





From the above picture we observe that quick sort algorithm has a higher runtime then merge sort algorithm. The runtimes shown here in the graph may differ from one computer to another as the type of processor also affects the runtime. To sum it all merge sort is better than quick sort as it takes less time to execute.

We see from the above pictures that there are some spikes in the graph. There are few reasons to it. They are as follows:-
1] Due to excessive heating of your computer.
2] Due to the number of tasks running while execution of the program.
3] Due to the random numbers.
4] Due to the arrangement of numbers before sorting.

CONCLUSION:	We saw two sorting algorithms quick and merge sort and the time they take to sort a large input (here 100000). We saw that quick sort is a bit slower as compared to merge sort.