NAME:	Atul Sharma
UID:	2021700060
SUBJECT	DAA
EXPERIMENT NO :	1-B
AIM:	**Experiment on finding the running time of an algorithm.**
Quick sort and merge sort.	
PROGRAM:	

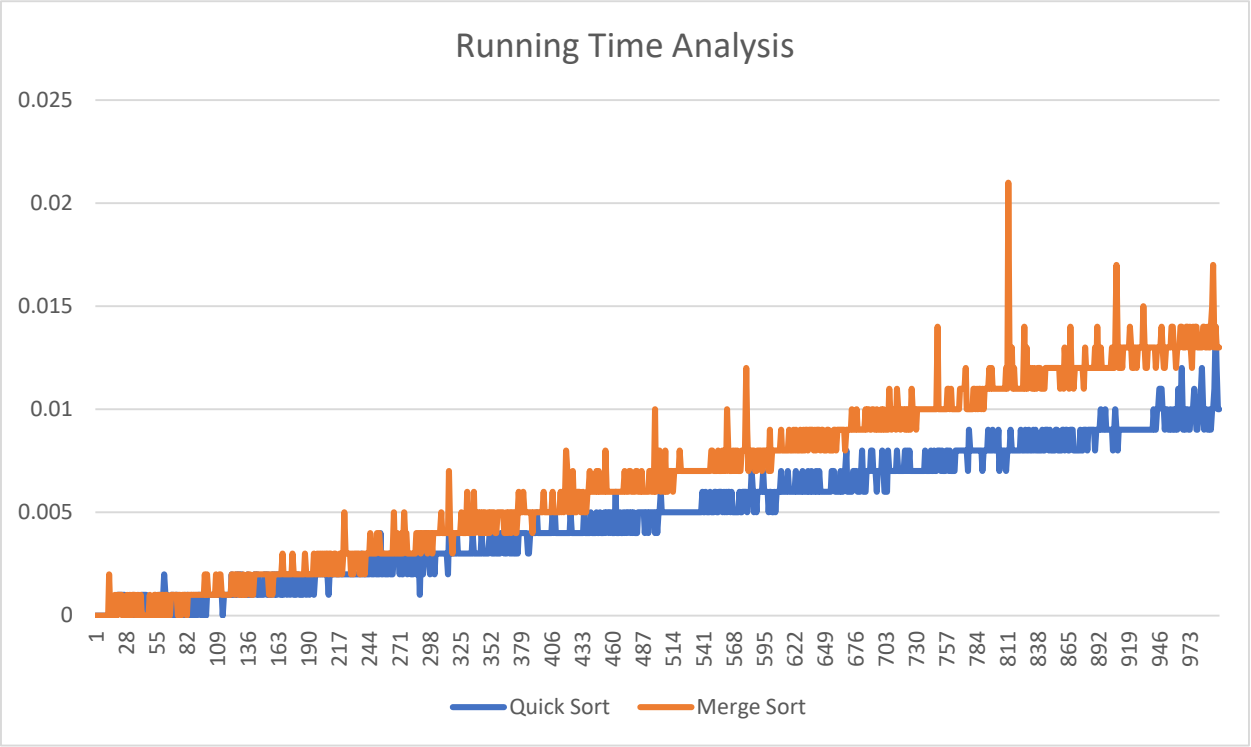
```c
#include <stdio.h>
#include <stdlib.h>
#include <stdlib.h>
#include <time.h>

int i = 0;

void swap(int arr[], int a, int b){

    int temp = arr[a];
    arr[a]= arr[b];
    arr[b] = temp;
}

void getInput()
{
    FILE *fp;
    fp = fopen("input.text", "w");
    for (int i = 0; i < 100000; i++)
        fprintf(fp, "%d ", rand() % 100000);
    fclose(fp);
}

void readfile(int arr[])
{
    FILE *fp;
    fp = fopen("input.text", "r");
    for (i; i % 100 == 0 && i != 0; i++)
    {
```

```
        fscanf(fp, "%d", &arr[i]);
    }
    fclose(fp);
}

void merge(int a[], int low, int mid, int high)
{
    int n1 = mid - low + 1;
    int n2 = high - mid;
    int left[n1], right[n2];
    for (int i = 0; i < n1; i++)
        left[i] = a[low + i];
    for (int j = 0; j < n2; j++){
        right[j] = a[mid + j + 1];
    }
    int i = 0, j = 0, k = low;
    while (i < n1 && j < n2)
    {
        if (left[i] <= right[j])
        {
            a[k] = left[i];
            i++;
            k++;
        }
        else
        {
            a[k] = right[j];
            k++;
            j++;
        }
    }
    while (i < n1)
    {
        a[k] = left[i];
        i++;
        k++;
    }
    while (j < n2)
    {
        a[k] = right[j];
        j++;
        k++;
    }
}
```

```
void mergesort(int arr[], int left, int right)
{

    if (right > left)
    {
        int middle = left + (right - left) / 2; // finding the
middle index of the array
        mergesort(arr, left, middle);        // recursively
calling for left half
        mergesort(arr, middle + 1, right);   // recursively
calling for right half
        merge(arr, left, middle, right); // again merging the
sublists
    }
}

int hpartition(int arr[], int low, int high){

    int pivot = arr[low];
    int i=low-1, j=high+1;
    while(1){
    do{
        i++;
    }while(arr[i]<pivot);
    do{
        j--;
    }while(arr[j]>pivot);

    if(i>=j)
    return j;
    swap(arr,i,j);
    }
}

void qSort(int arr[], int l, int h){
    if(l<h){
        int p = hpartition(arr,l,h);
        qSort(arr,l,p);
        qSort(arr, p+1, h);
    }
}

int main(){
```

```
    getInput();
    FILE *fp, *Wptr;
    int index=99;
    int arrNums[100000];
    clock_t t;
    fp = fopen("input.text", "r");
    Wptr = fopen("mTimes.txt", "w");
    for(int i=0; i<999; i++){
        for(int j=0; j<=index; j++){
            fscanf(fp, "%d", &arrNums[j]);
        }
        t = clock();
        mergesort(arrNums, 0, index);
        t = clock() - t;
        double time_taken = ((double)t)/CLOCKS_PER_SEC;
        fprintf(Wptr, "time taken for %d iteration is %Lf\n",
(i+1), time_taken);
        printf("%lf\n", time_taken);
        index = index + 100;
        fseek(fp, 0, SEEK_SET);
    }
    printf("\n\n");
    fclose(Wptr);
    Wptr = fopen("QTimes.txt", "w");
    index=99;
    for(int i=0; i<999; i++){
        for(int j=0; j<=index; j++){
            fscanf(fp, "%d", &arrNums[j]);
        }
        t = clock();
        qSort(arrNums, 0, index);
        t = clock() - t;
        double time_taken = ((double)t)/CLOCKS_PER_SEC;
        fprintf(Wptr, "time taken for %d iteration is %Lf\n",
(i+1), time_taken);
        printf("%lf\n",time_taken);
        index = index + 100;
        fseek(fp, 0, SEEK_SET);
    }
    fclose(Wptr);
    fclose(fp);
    return 0;
}
```

RESULT (SNAPSHOT)



I observed that quick sort is better than merge sort as it takes more time than quick sort as we increase the no. the inputs.

Time complexity:	
CONCLUSION:	Through this experiment, I understood the concept of time complexity of merge sort and quick sort . I also obsevered that when we increase the no. of inputs quick sort sort is better than merge sort .