NAME:	Vinit Madhyan
UID:	2021300070
SUBJECT	Design and Analysis of Algorithm
EXPERIMENT NO :	2
AIM:	Experiment on finding the running time of an quicksort and mergesort
Algorithm	**MERGE SORT ALGORITHM** 1. MERGE_SORT(arr, beg, end) 1. if beg < end 2. set mid = (beg + end)/2 3. MERGE_SORT(arr, beg, mid) 4. MERGE_SORT(arr, mid + 1, end) 5. MERGE (arr, beg, mid, end) 6. end of if 7. 8. END MERGE_SORT **QUICK SORT ALGORITHM** partition (arr[], low, high) { // pivot (Element to be placed at right position) pivot = arr[high]; i = (low – 1) // Index of smaller element and indicates the // right position of pivot found so far for (j = low; j <= high- 1; j++){ // If current element is smaller than the pivot if (arr[j] < pivot){ i++; // increment index of smaller element swap arr[i] and arr[j] } } swap arr[i + 1] and arr[high]) return (i + 1) }

	quickSort(arr[], low, high) { if (low < high) { /* pi is partitioning index, arr[pi] is now at right place */ pi = partition(arr, low, high); quickSort(arr, low, pi – 1); // Before pi quickSort(arr, pi + 1, high); // After pi }

PROGRAM:

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>

using namespace std;

int list[100000];

void read()
{
    ifstream fin("values.txt", ios::binary);
    for (long i = 0; i < 100000; i++)
    {
        fin.read((char *)&list[i], sizeof(int));
    }
    fin.close();
}

void merge(int arr[], int p, int q, int r)
{
    int n1 = q - p + 1;
    int n2 = r - q;

    int L[n1], M[n2];

    for (int i = 0; i < n1; i++)
        L[i] = arr[p + i];
    for (int j = 0; j < n2; j++)
```

	`` M[j] = arr[q + 1 + j]; int i, j, k; i = 0; j = 0; k = p; while (i < n1 && j < n2) { if (L[i] <= M[j]) { arr[k] = L[i]; i++; } else { arr[k] = M[j]; j++; } k++; } while (i < n1) { arr[k] = L[i]; i++; k++; } while (j < n2) { arr[k] = M[j]; j++; k++; } } void mergeSort(int arr[], int l, int r) { if (l < r) { int m = l + (r - l) / 2; mergeSort(arr, l, m); mergeSort(arr, m + 1, r); merge(arr, l, m, r); } } ```

```
long partition(long left, long right)
{
    int pivot_element = list[left];
    int lb = left, ub = right;
    int temp;

    while (left < right)
    {
        while (list[left] <= pivot_element)
            left++;
        while (list[right] > pivot_element)
            right--;
        if (left < right)
        {
            temp = list[left];
            list[left] = list[right];
            list[right] = temp;
        }
    }
    list[lb] = list[right];
    list[right] = pivot_element;
    return right;
}

void quickSort(long left, long right)
{
    if (left < right)
    {
        long pivot = partition(left, right);
        quickSort(left, pivot - 1);
        quickSort(pivot + 1, right);
    }
}

int main()
{
    clock_t t1, t2, t3, t4;
    read();
    int num = 100;
    for (int i = 0; i < 1000; i++)
    {
        t1 = clock();
        mergeSort(list, 0, num - 1);
        t2 = clock();
        t3 = clock();
        quickSort(0, num - 1);
        t4 = clock();
        double mergetime = double(t2 - t1) / double(CLOCKS_PER_SEC);
        double quicktime = double(t4 - t3) / double(CLOCKS_PER_SEC);
```

```
        cout << endl;
        cout << i + 1 << " " << fixed << mergetime << "\t";
        cout << fixed << quicktime;
        num += 100;
    }

    return 0;
}
```

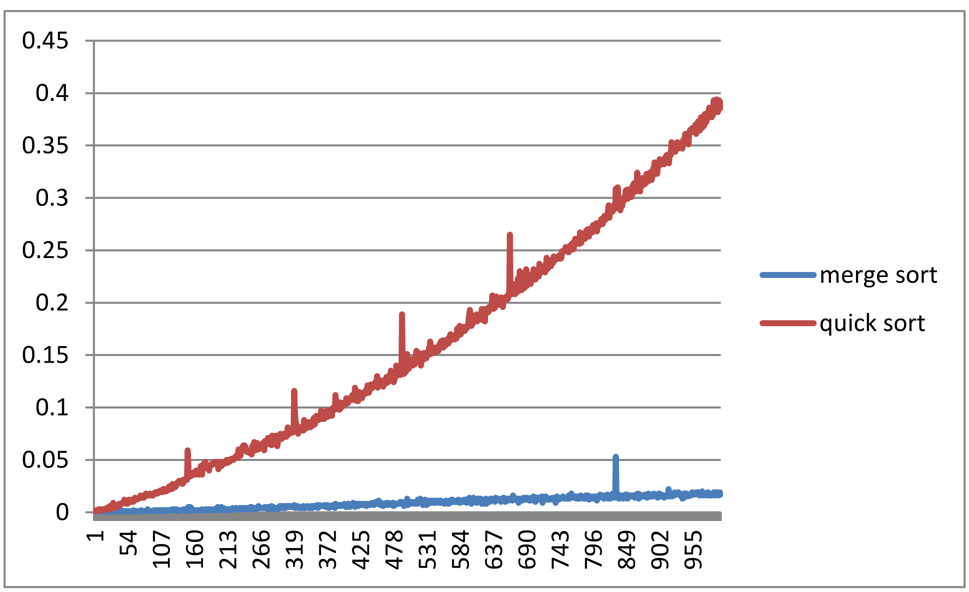
Values.cpp

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

int main()
{
    for (int i = 1; i <= 100000; i++)
    {
        cout << rand() << " ";
    }

    return 0;
}
```

Observation (SNAPSHOT)



After graphing the run times of both merge sort and quick sort, I concluded that merge sort outperforms quick sort for large arrays. This experiment deepened my understanding of the inner workings and practical application of both algorithms

Conclusion

Thus I have understood the Merge and Quick sort algorithm and their time complexities. I also understood how to calculate them and draw similar inferences.