Name:-Aditya Harish Khuman

UID:-2021300061

SUBJECT:-Design and analysis of algorithm

Aim:-Exp to find on finding the running time for quick sort and merge sort

Algorithm

1. Merge Sort

Step 1: Find the middle index of the array.

Middle = 1 + (last – first)/2

Step 2: Divide the array from the middle.

Step 3: Call merge sort for the first half of the array

MergeSort(array, first, middle)

Step 4: Call merge sort for the second half of the array.

MergeSort(array, middle+1, last)

Step 5: Merge the two sorted halves into a single sorted array.

2. Quick Sort

Step 1 - Consider the first element of the list as pivot (i.e., Element at first position in the array).

Step 2 - Define two variables i and j. Set i and j to first and last elements of the list respectively.

Step 3 - Increment i until arr[i] > pivot then stop.

Step 4 - Decrement j until arr[j] < pivot then stop.

Step 5 - If i < j then exchange list[i] and list[j].

Step 6 - Repeat steps 3,4 & 5 until i > j.

Step 7 - Exchange the pivot element with arr[j] element..

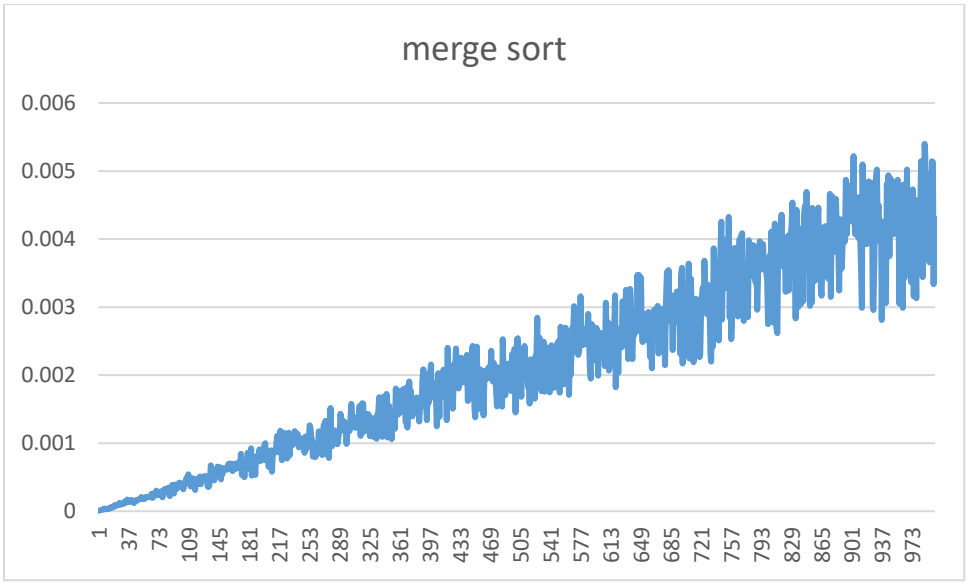
```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>

using namespace std;
```

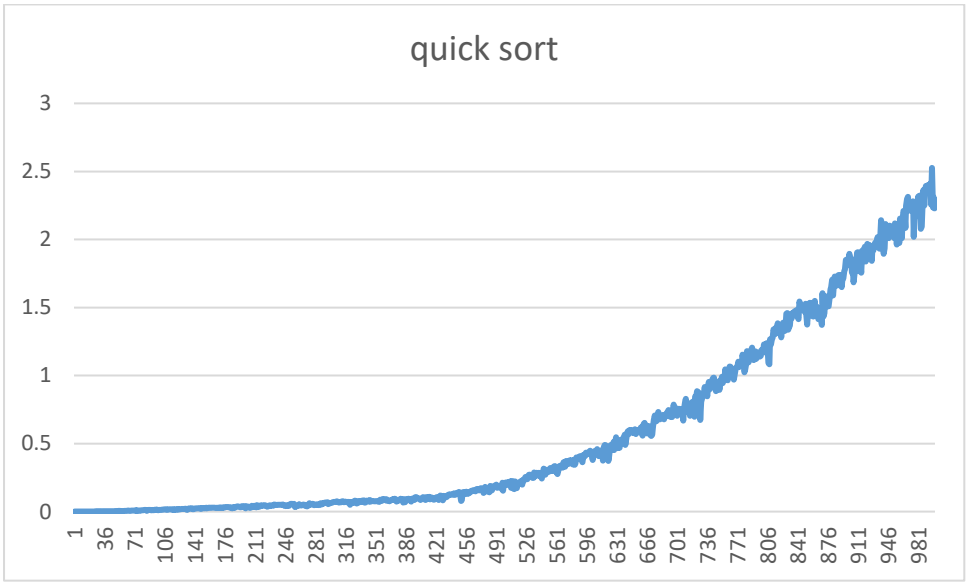
```
int arr[100000];

void read()
{
   ifstream fin("values.txt", ios::binary);
   for (long i = 0; i < 100000; i++)
   {
      fin.read((char *)&arr [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 = arr[left];
    int lb = left, ub = right;
    int temp;

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


merge sort
0.006
0.005
0.004
0.003
0.002
0.001
0
1
37
73
109
145
181
217
253
289
325
361
397
433
469
505
541
577
613
649
685
721
757
793
829
865
901
937
973


quick sort
3
2.5
2
1.5
1
0.5
0
1
36
71
106
141
176
211
246
281
316
351
386
421
456
491
526
561
596
631
666
701
736
771
806
841
876
911
946
981