|  |  |
| --- | --- |
| **NAME:** | Dewyani Anil Deshmukh |
| **UID:** | 2021300024 |
| **CLASS:** | SE Comps-A |
| **EXPERIMENT NO:** | 2 |

**AIM**: Finding the running time of mergesort and quicksort algorithms.

**CODE:**

**Generating 100000 numbers:**

#include <stdio.h>

#include <stdlib.h>

int main()

{

    int num,offset=0;

    int i,j;

    FILE \*fptr;

    fptr=fopen("numbers.txt","w");

    for(i=0; i<1000; i++)

    {

        for (j= 0; j<100; j++)

        {

           num = rand() % 100 + offset;

           fprintf(fptr," %d ",num);

        }

        offset+=100;

    }

    fclose(fptr);

    return 0;

}

**Sorting algorithms:**

#include <stdio.h>

#include <time.h>

int partition(int arr[], int lb, int ub)

{

    int temp, temp1;

    int pivot = arr[ub];

    int i=lb-1;

        for(int k=lb;k<ub;k++)

        {

            if(arr[k]<pivot)

            {

            i++;

            temp = arr[i];

            arr[i] = arr[k];

            arr[k] = temp;

            }

          temp1 = arr[i+1];

          arr[i+1] = arr[ub];

          arr[ub] = temp1;

        }

        return (i+1);

    }

void quicksort(int arr[], int lb, int ub)

{

    if (lb < ub)

    {

        int loc = partition(arr, lb, ub);

        quicksort(arr, lb, loc - 1);

        quicksort(arr, loc + 1, ub);

    }

}

void enter\_quick(int a[], int size)

{

    quicksort(a,0,size-1);

}

void merge(int a[], int lb, int mid, int ub)

{

    int i, j, k;

    int n1 = mid - lb + 1;

    int n2 = ub - mid;

    int l[n1], r[n2];

    for (int i = 0; i < n1; i++)

    {

        l[i] = a[lb + i];

    }

    for (int j = 0; j < n2; j++)

    {

        r[j] = a[mid + 1 + j];

    }

    i = 0;

    j = 0;

    k = lb;

    while (i < n1 && j < n2)

    {

        if (l[i] <= r[j])

        {

            a[k] = l[i];

            i++;

        }

        else

        {

            a[k] = r[j];

            j++;

        }

        k++;

    }

    while (i < n1)

    {

        a[k] = l[i];

        i++;

        k++;

    }

    while (j < n2)

    {

        a[k] = r[j];

        j++;

        k++;

    }

}

void mergesort(int a[], int lb, int ub)

{

    if (lb < ub)

    {

       int mid = (lb + ub) / 2;

        mergesort(a, lb, mid);

        mergesort(a, (mid + 1), ub);

        merge(a, lb, mid, ub);

    }

}

void enter\_merge(int a[],int size)

{

   mergesort(a,0,size-1);

}

void timecalc()

{

      int data, len;

    FILE \*fptr;

    fptr = fopen("random.txt", "r");

   int a[100000];

   for(int k=0;k<100000;k++)

   {

    fscanf(fptr,"%d\n",&a[k]);

   }

    //fclose(fptr);

    int a1[100000];

    int a2[100000];

    for (int i = 0; i <1000; i++)

    {

        int size = 100\*(i+1);

        for (int j = 0; j < size; j++)

        {

            a1[j] = a[j];

            a2[j] = a[j];

        }

        clock\_t start = clock();

        enter\_quick(a1,size);

        double diff = (double)((clock()- start) / CLOCKS\_PER\_SEC);

        clock\_t start1 = clock();

        enter\_merge(a,size);

        double diff1 =(double) ((clock()- start1) / CLOCKS\_PER\_SEC);

        printf("%d\t%lf\t%lf",(i+1),diff,diff1);

        printf("\n");

}

}

int main()

{

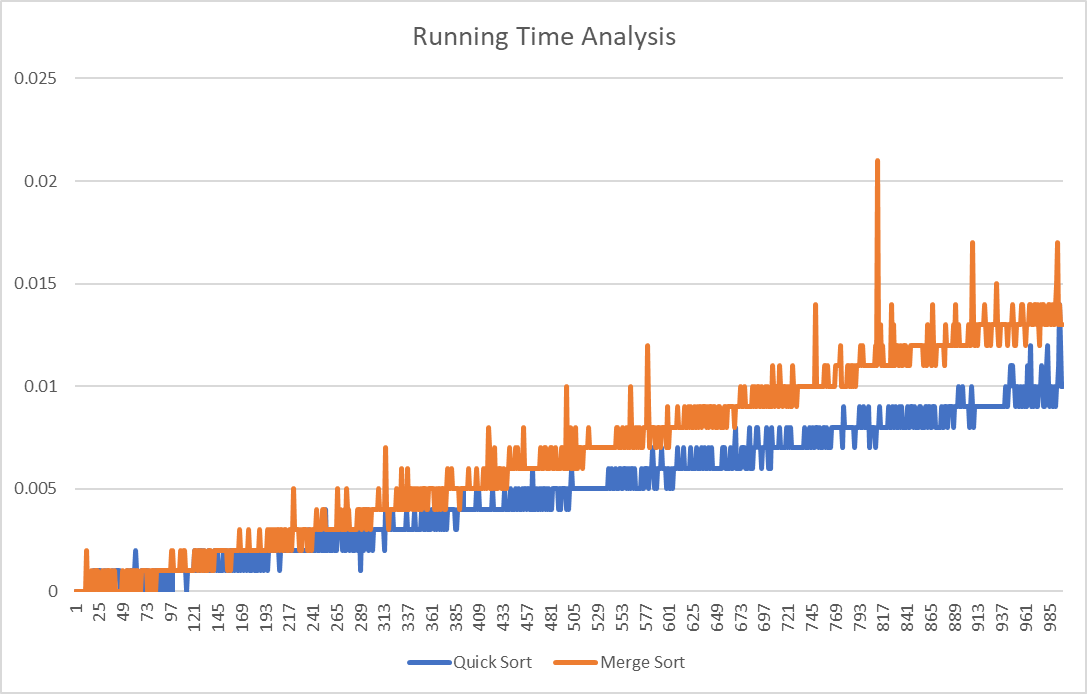
    timecalc();

    return 0;

}

**GRAPHICAL**

**REPRESENTATION:**

****

**OBSERVATIONS:**

1. From the graph it can be observed that quick sort is faster than mergesort.
2. The time complexity for both the algorithms is O(nlgn).
3. The space complexity for both the algorithms is same i.e O(n).

**CONCLUSION:** I have successfully implemented both the sorting algorithms and have analysed their time and space complexity.