Name: Rishi Tiku

Class: SE DS

Subject: DAA

Experiment: 2.1

UID: 2021700067

Aim: To analyze algorithms quicksort, merge sort, selection sort and insertion sort by counting number of swaps and comparisons.

Code

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

void printCur()
{
    time_t s, val = 1;
    struct tm* current_time;

    // time in seconds
    s = time(NULL);

    // to get current time
    current_time = localtime(&s);

    // print time in minutes,
    // hours and seconds
    printf("%02d:%02d:%02d\n",
           current_time->tm_hour,
           current_time->tm_min,
           current_time->tm_sec);

}

long long int swapm = 0;
long long int swapq = 0;
long long int swapi = 0;
```

```
long long int swaps = 0;

long long int compm = 0;
long long int compq = 0;
long long int compi = 0;
long long int comps = 0;


void InsertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key)
        {
            compi++;
            arr[j + 1] = arr[j];
            swapi++;
            j = j - 1;
        }
        arr[j + 1] = key;
        swapi++;
    }
}

void SelectionSort(int * A, int n)
{
    for(int i = 0; i<n; i++)
    {
        int min = A[i];
        int pos = i;
        int temp;
        for(int j = i; j<n; j++)
        {
            if(A[j]<min)
            {
                min = A[j];
                pos = j;
            }
```

```
            comps++;
        }
        temp = A[i];
        A[i] = min;
        A[pos] = temp;
        swaps++;
    }
}

void quickSort(int number[],int first,int last){
    int i, j, pivot, temp;
    if(first<last){
        pivot=first;
        i=first;
        j=last;
        while(i<j){
            while(number[i]<=number[pivot]&&i<last)
            {
                i++;
                compq++;
            }
            while(number[j]>number[pivot])
            {
                j--;
                compq++;
            }
            if(i<j){
                temp=number[i];
                number[i]=number[j];
                number[j]=temp;
                swapq++;
             }
        }
        temp=number[pivot];
        number[pivot]=number[j];
        number[j]=temp;
        swapq++;
        quickSort(number,first,j-1);
        quickSort(number,j+1,last);
    }
}
```

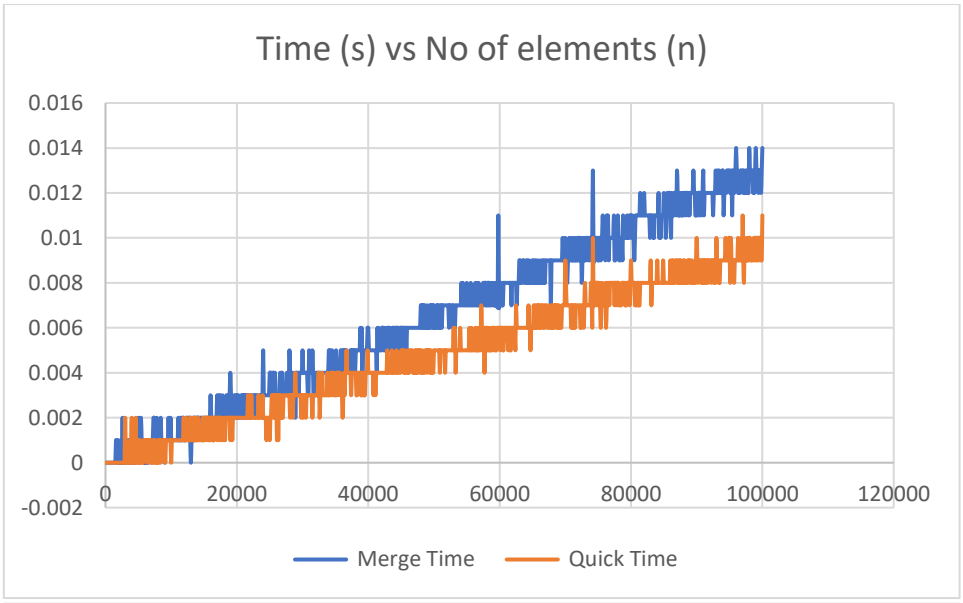
```c
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            compm++;
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        swapm++;
        k++;
    }

    /* Copy the remaining elements of L[], if there
    are any */
    while (i < n1) {
        arr[k] = L[i];
        swapm++;
        i++;
        k++;
    }
```

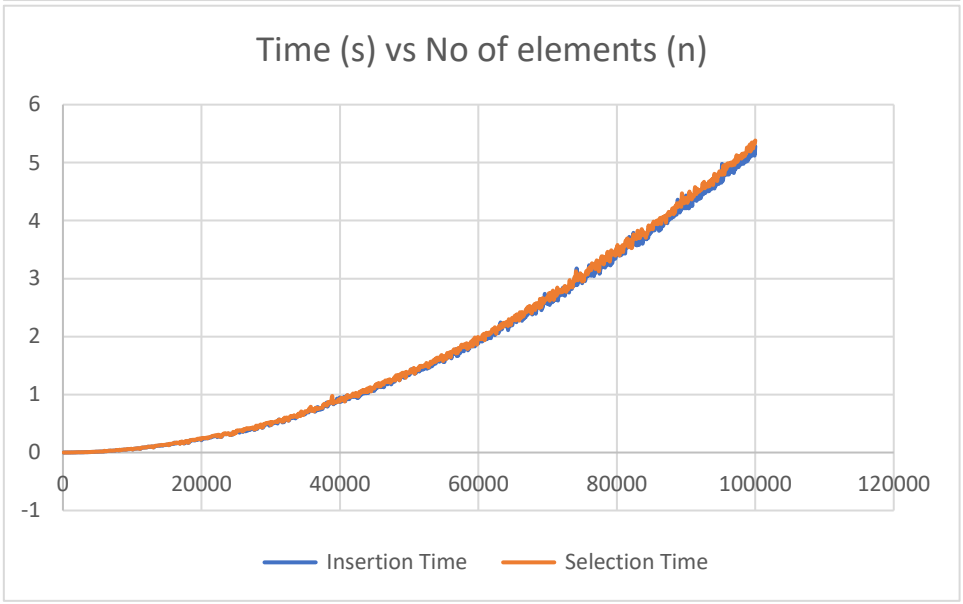
```
    /* Copy the remaining elements of R[], if there
    are any */
    while (j < n2) {
        arr[k] = R[j];
        swapm++;
        j++;
        k++;
    }
}

/* l is for left index and r is right index of the
sub-array of arr to be sorted */
void mergeSort(int arr[], int l, int r)
{
    if (l < r) {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l + (r - l) / 2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}


int genFile()
{
    FILE * fptr;
    if(!(fptr = fopen("RandomNumbers.txt", "w")))
    {
        return 1;
    }

    for(int i = 0; i<100000; i++)
    {
        fprintf(fptr, "%d\n", rand());
    }

    fclose(fptr);
```

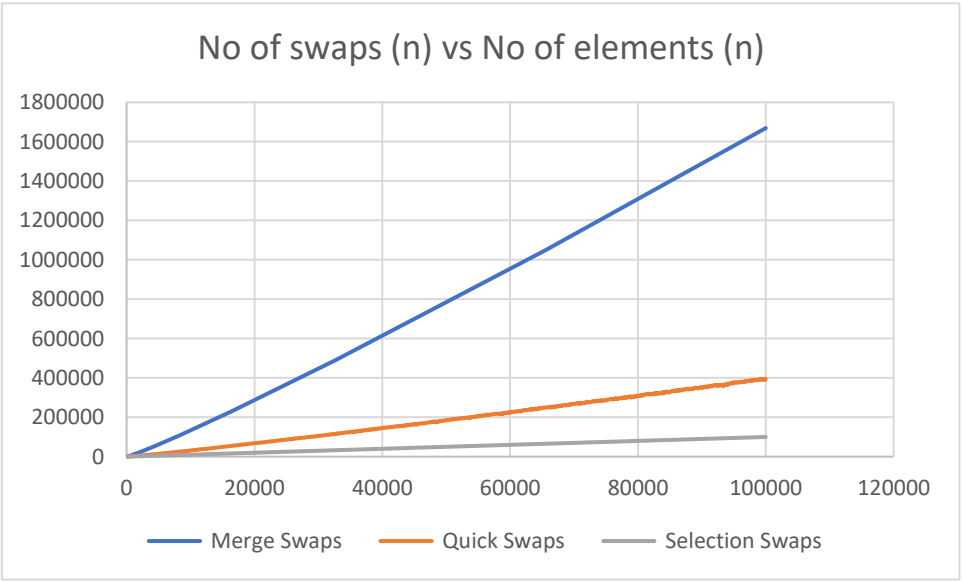
```
    return 0;
}


int main()
{
    genFile();
    FILE * fptrs, * fptrd;
    char ch;
    fptrd = fopen("CompsSwaps.csv", "w");
    if(!fptrd)
        return 0;
    fprintf(fptrd, "Cases, Merge Time, Merge Swaps, Merge Comps,
Quick Time, Quick Swaps, Quick Comps, ");
    fprintf(fptrd, "Insertion Time, Insertion Swaps, Insertion
Comps, Selection Time, Selection Swaps, Selection Comps\n");

    for(long int x = 100; x<=100000; x+=100)
    {
        fptrs = fopen("RandomNumbers.txt", "r");
        int A[x], B[x], C[x], D[x];
        clock_t start1, end1, start2, end2, start3, end3, start4,
end4;
        if(x % 1000 == 0)
        {
                printf("x = %ld. Time = ", x);
                printCur();
        }
        for(long int i = 0; i<x; i++)
        {
            fscanf(fptrs, "%d\n", &A[i]);
            D[i] = C[i] = B[i] = A[i];
        }

        start1 = clock();
        mergeSort(A, 0, x-1);
        end1 = clock();

        start2 = clock();
        quickSort(B, 0, x-1);
        end2 = clock();
```

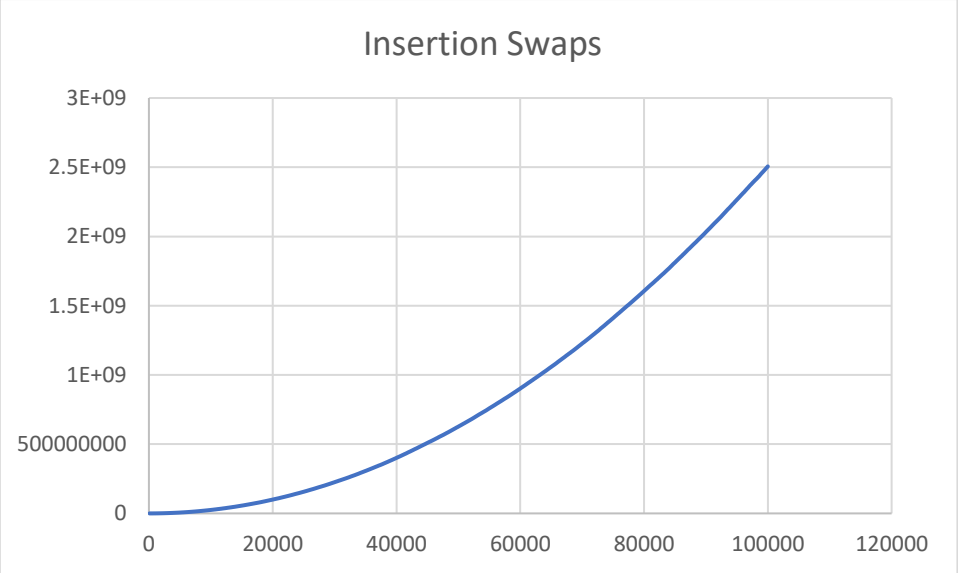
```
        start3 = clock();
        InsertionSort(C, x);
        end3 = clock();

        start4 = clock();
        SelectionSort(D, x);
        end4 = clock();

        double t1 = (double) (end1 - start1) / CLOCKS_PER_SEC;
        double t2 = (double) (end2 - start2) / CLOCKS_PER_SEC;
        double t3 = (double) (end3 - start3) / CLOCKS_PER_SEC;
        double t4 = (double) (end4 - start4) / CLOCKS_PER_SEC;

        fprintf(fptrd, "%ld, %f, %lld, %lld, %f, %lld, %lld, ", x,
t1, swapm, compm, t2, swapq, compq);
        fprintf(fptrd, "%f, %lld, %lld, %f, %lld, %lld\n", t3,
swapi, compi, t4, swaps, comps);
        // printf("%ld, %lld, %lld, %lld, %lld\n", x, swapm, compm,
swapq, compq);
        // scanf("%c", &ch);

        fclose(fptrs);
        // if(ch == 'e')
        //     return 0;
        swapm = 0;
        compm = 0;
        swapq = 0;
        compq = 0;
        swapi = 0;
        compi = 0;
        swaps = 0;
        comps = 0;
    }
    //fclose(fptrd);
    return 0;
}
```

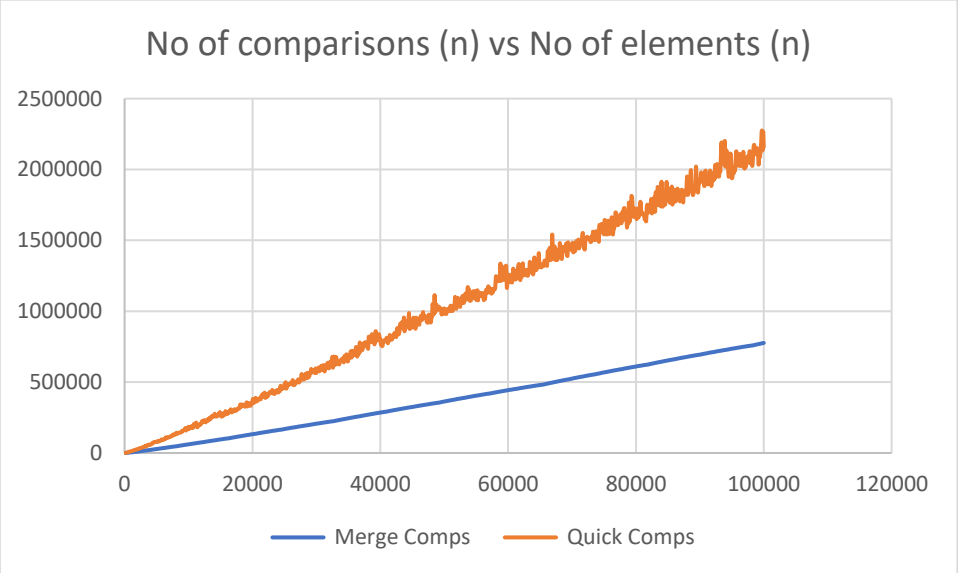
Graphs

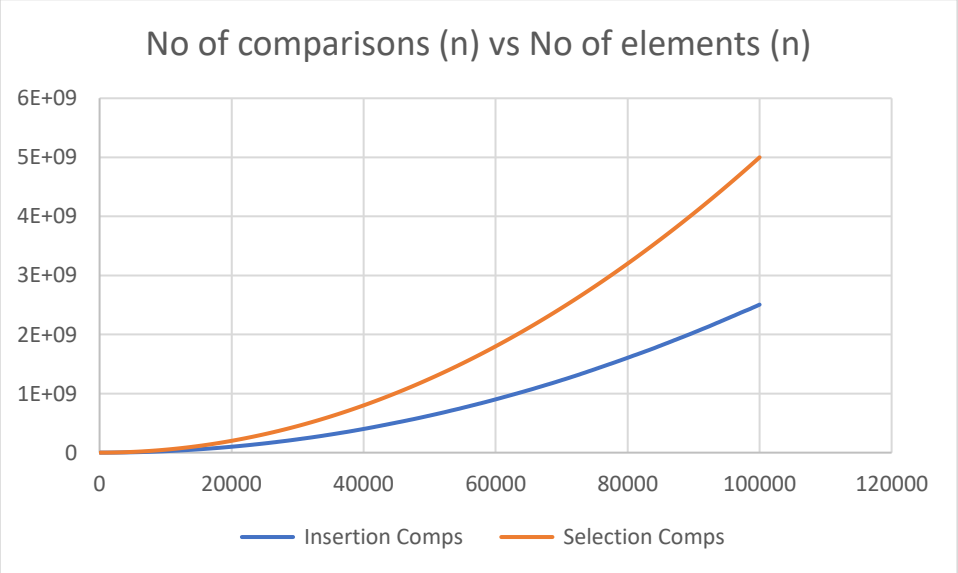

Time (s) vs No of elements (n)
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
-0.002
0
20000
40000
60000
80000
100000
120000
Merge Time
Quick Time


Time (s) vs No of elements (n)
6
5
4
3
2
1
0
-1
0
20000
40000
60000
80000
100000
120000
Insertion Time
Selection Time


No of swaps (n) vs No of elements (n)
1800000
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
0
20000
40000
60000
80000
100000
120000
Merge Swaps
Quick Swaps
Selection Swaps


Insertion Swaps
3E+09
2.5E+09
2E+09
1.5E+09
1E+09
500000000
0
0
20000
40000
60000
80000
100000
120000


No of comparisons (n) vs No of elements (n)
2500000
2000000
1500000
1000000
500000
0
0
20000
40000
60000
80000
100000
120000
Merge Comps
Quick Comps


No of comparisons (n) vs No of elements (n)
6E+09
5E+09
4E+09
3E+09
2E+09
1E+09
0
0
20000
40000
60000
80000
100000
120000
Insertion Comps
Selection Comps

Observations

- Selection Sort has the least number of swaps. Maximum of n^2 and only once per iteration
- Quicksort has quadruple times less swaps than Merge sort, but almost thrice the number of comparisons.
- Insertion sort has the most swaps as each array element between the current and sorted position of the element has to move to make space.
- Insertion sort comparisons & Selection sort comparisons are significantly higher than those of merge sort and quick sort
- Selection Sort and Insertion sort have relatively similar time taken and similar growth, non-linear, while Quicksort and Merge sort have relatively low and linear times.
- Dominant operations:
    - In Selection sort: Comparisons
    - In Insertion sort: Swaps
    - In Quicksort: Comparisons
    - In Merge sort: Swaps

Result

Analysis of algorithms is done by counting number of comparisons and swaps and finding the dominant operation.