DESIGN AND ANALYSIS OF ALGORITHMS LAB

NAME: Mohammed Saad Belgi

UID: 2021700005

BATCH: A

BRANCH: CSE DS

EXPT. NO.: 1B

AIM: Experiment on finding the running time of an algorithm.

ALGORITHM:

Selection sort:

SELECTION-SORT(ARR):

1. for i = 1 to ARR.LENGTH-1:
2.   min_idx ← i
3.   for j = i + 1 to ARR.LENGTH:
4.   if ARR[j] < ARR[min_idx]:
5.   min_idx = j
6.   if i != min_idx:
7.   temp ← ARR[min_idx]
8.   ARR[min_idx] ← ARR[i]
9.   ARR[i] ← temp

Insertion sort:

INSERTION-SORT(ARR):

1. for j = 2 to ARR.LENGTH:
2.   key ← ARR[j]
3.   i ← j – 1
4.   while i > 0 and ARR[i] > key:
5.   ARR[i+1] ← ARR[i]
6.   i ← i - 1
7.   ARR[i + 1] ← key

THEORY:

Selection and insertion sort algorithms have a worst-case time complexity of $O(n^2)$, n being the size of the input array to be sorted. Both algorithms require n(n-1)/2 comparisons in worst-case scenario. However, selection sort performs n(n-1)/2 comparisons in all cases and thus has quadratic time complexity for best and average cases as well, while insertion sort requires fewer comparisons in most cases. In best-case scenario, insertion sort requires n comparisons, and thus has a linear time complexity. This is why insertion sort outperforms selection sort in most cases.

Both these algorithms are in-place sorting algorithms, i.e., extra data structures are not required to sort an array. Number of temporary variables required is constant (independent of input size) for both these algorithms, and thus they have a constant space complexity (O(1)).

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>    // required to use clock function if executing
program on linux
#ifdef _WIN32        // to check if OS is windows (_WIN32 is a macro
defined on every windows based gcc compiler)
#include <windows.h> // required to use windows api for time
measurement if executing on windows
#endif

/* clock function of time.h on linux provides precision up to
microseconds,
but on windows precision up to only milliseconds is supported
on windows, the windows api has much better functions for measuring
time. */

// prototypes:
void selectionSort(int *, int);
void insertionSort(int *, int);
void deepCopy(int *, int *, int);
void generateRandomNumbers();
void LinuxMain();
void WindowsMain();

int main()
{
    generateRandomNumbers();
#ifdef _WIN32
    WindowsMain();
#else
    LinuxMain();
#endif
    return 0;
}

void selectionSort(int *arr, int len)
{
    int min_i, temp;
    for (int i = 0; i < len; i++)
    {
        min_i = i;
        for (int j = i + 1; j < len; j++)
```

```
        {
            if (arr[j] < arr[min_i])
                min_i = j;
        }
        if (i != min_i)
        {
            temp = arr[min_i];
            arr[min_i] = arr[i];
            arr[i] = temp;
        }
    }
}

void insertionSort(int *arr, int len)
{
    int key, pos;
    for (int i = 1; i < len; i++)
    {
        key = arr[i];
        pos = 0;
        for (int j = i - 1; j >= 0; j--)
        {
            if (arr[j] > key)
                arr[j + 1] = arr[j];
            else
            {
                pos = j + 1;
                break;
            }
        }
        arr[pos] = key;
    }
}

void deepCopy(int *source, int *dest, int len)
{
    for (int i = 0; i < len; i++)
        dest[i] = source[i];
}

void generateRandomNumbers()
{
    FILE *fptr = fopen("rand_num.txt", "w");
    time_t cur_time;
    srand((unsigned int)time(&cur_time));
    for (int i = 0; i < 100000; i++)
        fprintf(fptr, "%d\n", rand());
    fclose(fptr);
```

```
}

void WindowsMain()
{
    FILE *rand_num = fopen("rand_num.txt", "r");
    FILE *dest = fopen("output.txt", "w");
    fprintf(dest, "size   | selection-sort-time | insertion-sort-
time\n");
    double time1, time2;
    LARGE_INTEGER clock_freq, start, end;
    QueryPerformanceFrequency(&clock_freq);
    for (int size = 100; size <= 100000; size += 100)
    {
        int arr1[size];
        int arr2[size];
        for (int j = 0; j < size; j++)
            fscanf(rand_num, "%d", &arr1[j]);
        fseek(rand_num, 0, SEEK_SET);
        deepCopy(arr1, arr2, size);

        QueryPerformanceCounter(&start);
        selectionSort(arr1, size);
        QueryPerformanceCounter(&end);
        time1 = (double)(end.QuadPart - start.QuadPart) * 1.0 /
clock_freq.QuadPart;

        QueryPerformanceCounter(&start);
        insertionSort(arr2, size);
        QueryPerformanceCounter(&end);
        time2 = (double)(end.QuadPart - start.QuadPart) * 1.0 /
clock_freq.QuadPart;

        fprintf(dest, "%6d | %19f | %19f\n", size, time1, time2);
        printf("Size %d done!\n", size);
    }
    fclose(rand_num);
    fclose(dest);
}

void LinuxMain()
{
    FILE *rand_num = fopen("rand_num.txt", "r");
    FILE *dest = fopen("output.txt", "w");
    fprintf(dest, "size   | selection-sort-time | insertion-sort-
time\n");
    double time1, time2;
    clock_t start, end;
    for (int size = 100; size <= 100000; size += 100)
```

```
    {
        int arr1[size];
        int arr2[size];
        for (int j = 0; j < size; j++)
            fscanf(rand_num, "%d", &arr1[j]);
        fseek(rand_num, 0, SEEK_SET);
        deepCopy(arr1, arr2, size);

        start = clock();
        selectionSort(arr1, size);
        end = clock();
        time1 = (double)(end - start) * 1.0 / CLOCKS_PER_SEC;

        start = clock();
        insertionSort(arr2, size);
        end = clock();
        time2 = (double)(end - start) * 1.0 / CLOCKS_PER_SEC;

        fprintf(dest, "%6d | %19f | %19f\n", size, time1, time2);
        printf("Size %d done!\n", size);
    }
    fclose(rand_num);
    fclose(dest);
}
```

EXECUTION AND OUTPUT:

output.txt file (first and last 20 rows):

```
size   |  selection-sort-time | insertion-sort-time
   100 |             0.000023 |            0.000011
   200 |             0.000204 |            0.000073
   300 |             0.000177 |            0.000124
   400 |             0.000323 |            0.000156
   500 |             0.000496 |            0.000250
   600 |             0.000592 |            0.000320
   700 |             0.000892 |            0.000434
   800 |             0.001478 |            0.000683
   900 |             0.001898 |            0.000881
  1000 |             0.001959 |            0.000891
  1100 |             0.001798 |            0.000748
  1200 |             0.003412 |            0.001624
  1300 |             0.002960 |            0.001330
  1400 |             0.002545 |            0.001214
  1500 |             0.005046 |            0.002683
  1600 |             0.004671 |            0.002283
  1700 |             0.005758 |            0.003045
  1800 |             0.008210 |            0.005611
  1900 |             0.012246 |            0.006190
  2000 |             0.015206 |            0.007333
```

98100	15.536649	8.841853
98200	16.401716	8.817381
98300	15.988735	8.248043
98400	16.681684	8.592832
98500	15.905660	8.985641
98600	15.980266	8.267280
98700	13.343732	6.814934
98800	12.420786	6.034174
98900	12.052699	6.035503
99000	12.230165	5.986056
99100	11.993930	6.044758
99200	12.084464	5.991187
99300	12.025502	6.112879
99400	12.196849	6.054576
99500	12.058697	6.014215
99600	12.128793	6.151081
99700	12.118082	6.069470
99800	12.114044	6.154202
99900	12.119248	6.124878
100000	12.222517	6.362584

rand_num - Notepad

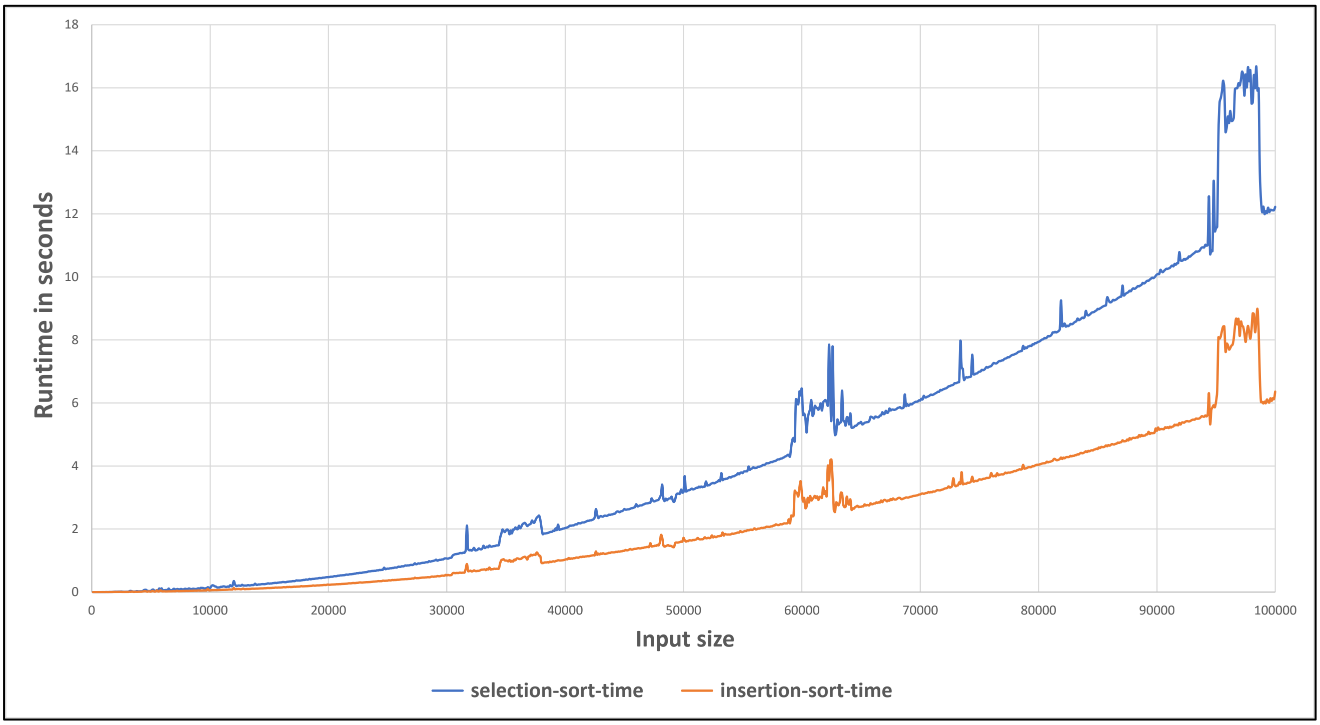
File Edit View

```
8184
21874
32211
22458
2198
21309
20677
8229
30925
32365
21237
10383
25564
7316
5568
22195
2851
20967
13201
1564
7056
6375
30970
23784
16219
23873
9220
24011
30707
15858
11715
23383
21560
15022
21037
12410
22250
20611
18988
25160
8545
```

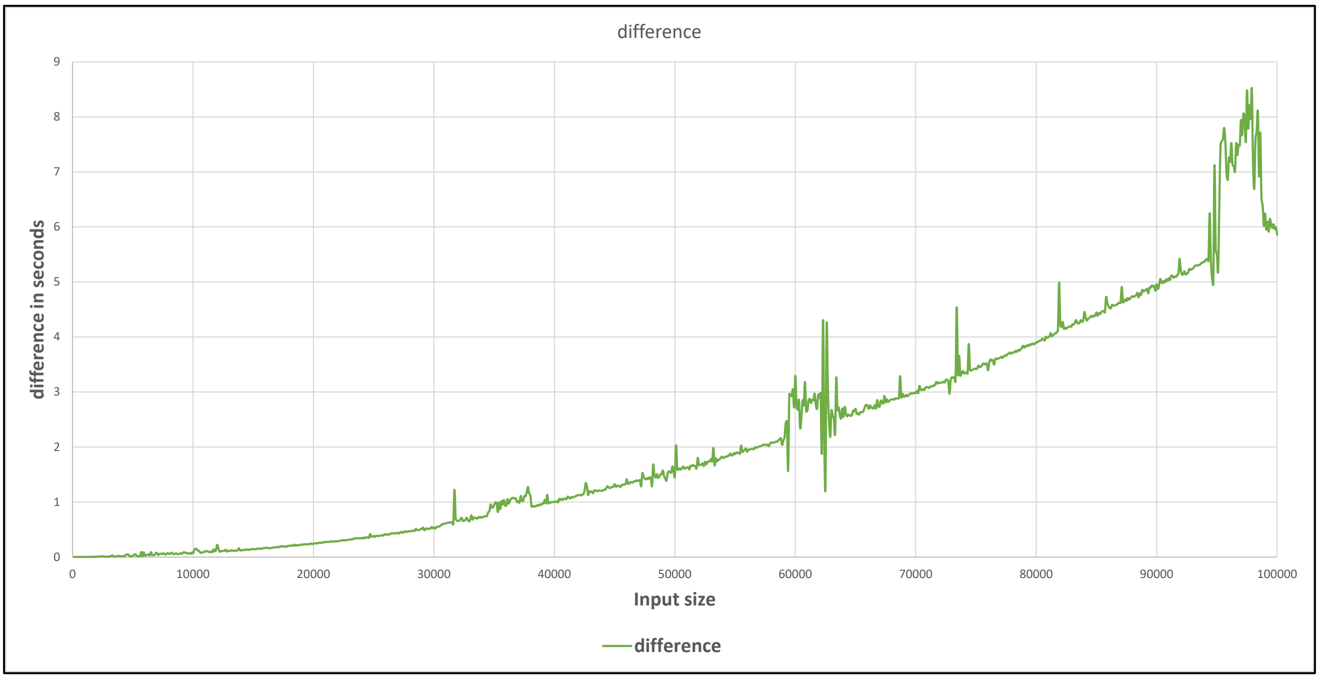
Ln 100000, Col 1

PLOTTING THIS DATA IN EXCEL:

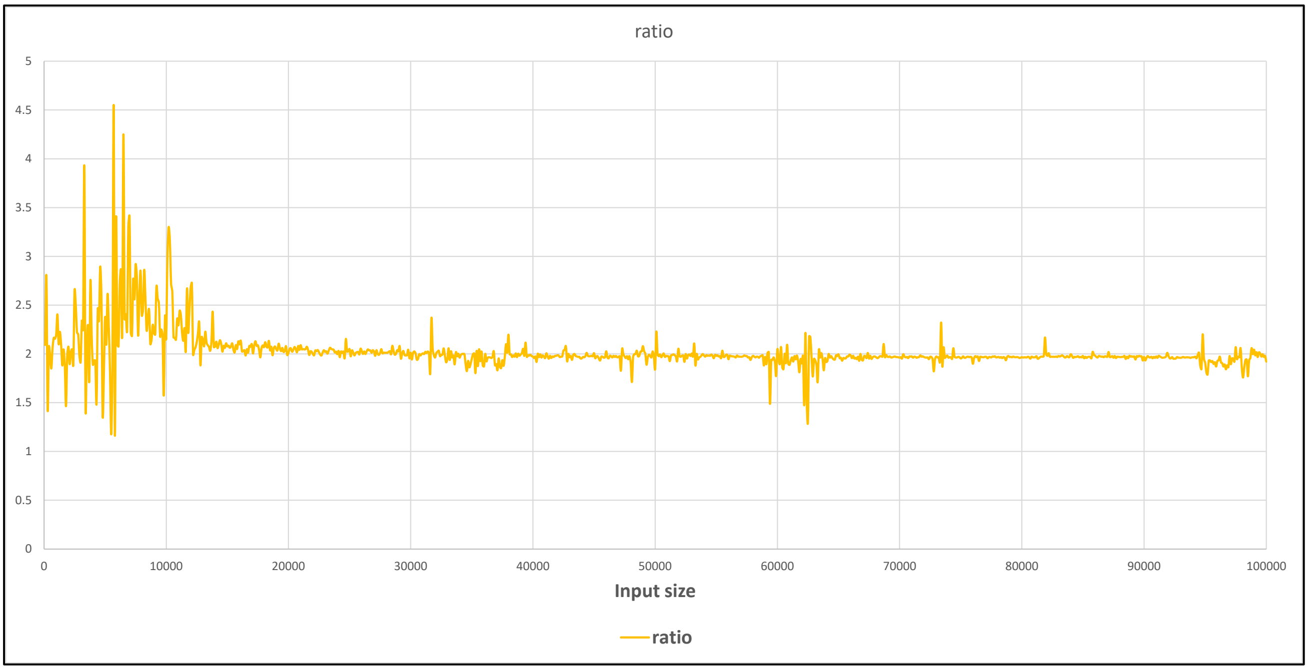
Graph of running time of selection sort and insertion sort vs input size:



Graph of difference between running time of selection sort and insertion sort vs input size:



Graph of ratio of running time of selection sort to that of insertion sort vs input size:



EMPIRICAL OBSERVATIONS AND RESULT ANALYSIS:

Insertion sort beats selection sort for every input size.

Difference between running time of selection and insertion sort also increases along with input size. This difference was observed to be more than 8 seconds for some cases (with large input size).

There are large fluctuations in the ratio of running time of the two algorithms (selection/insertion sort) for smaller values of input size (n < 15000). For n>15000, this ratio stabilises around an approximate value of 2. From empirical observation, running time of selection sort was found to be roughly twice of that of insertion sort for the same input.

CONCLUSION:

Insertion sort outperforms selection sort for same input most of the times, despite having the same worst-case time complexity of $O(n^2)$.