Name: Rishi Tiku

Class: SE DS

UID: 202170067

Subject: Design & Analysis of Algorithms

Experiment: 1-b

AIM: To find the running time of Insertion & Selection Sort Algorithms.

Introduction

Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list. The algorithm repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first element of the unsorted portion.

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

Algorithm

Selection Sort

1) START
2) Initialize minimum value(min_idx) to location 0.
3) Traverse the array to find the minimum element in the array.
4) While traversing if any element smaller than min_idx is found then swap both the values.
5) Then, increment min_idx to point to the next element.
6) Repeat until the array is sorted.
7) STOP

Insertion Sort:

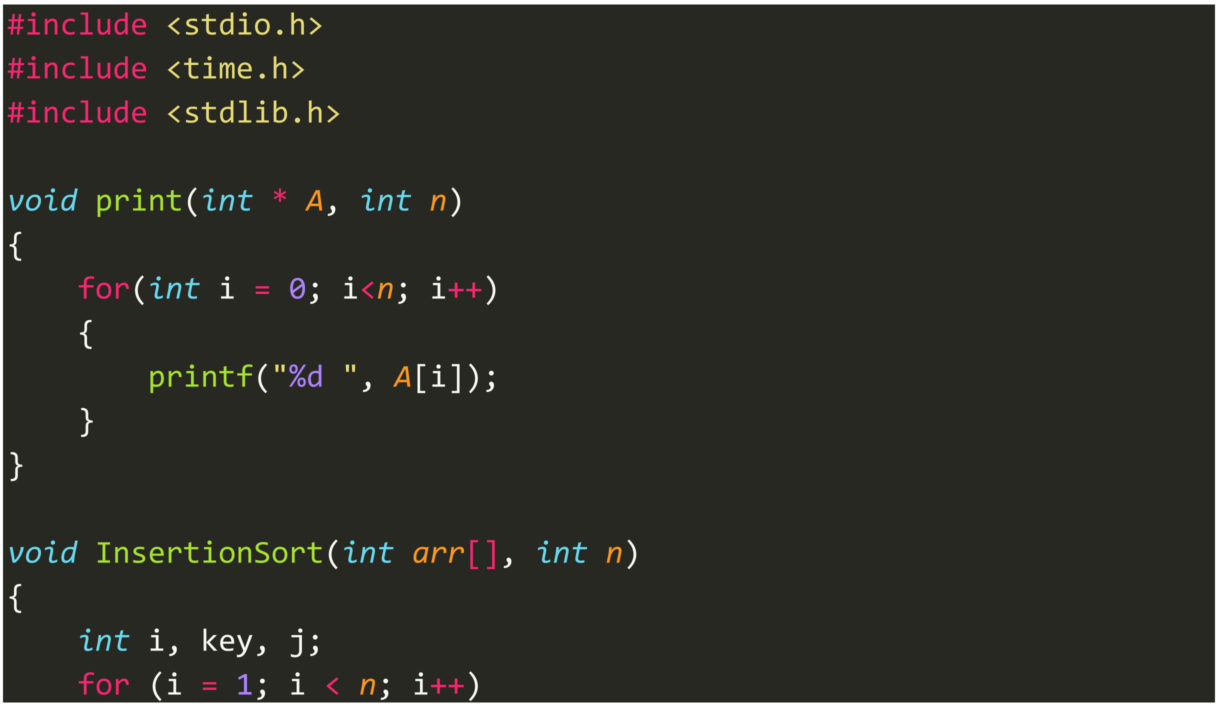
1) START
2) n = length(A)

```
3)  for i = 1 to n - 1 do
4)      j = i
5)      while j > 0 and A[j-1] > A[j] do
6)          swap(A[j], A[j-1])
7)              j = j - 1
8)      end while
9)   end for
10)  STOP
```

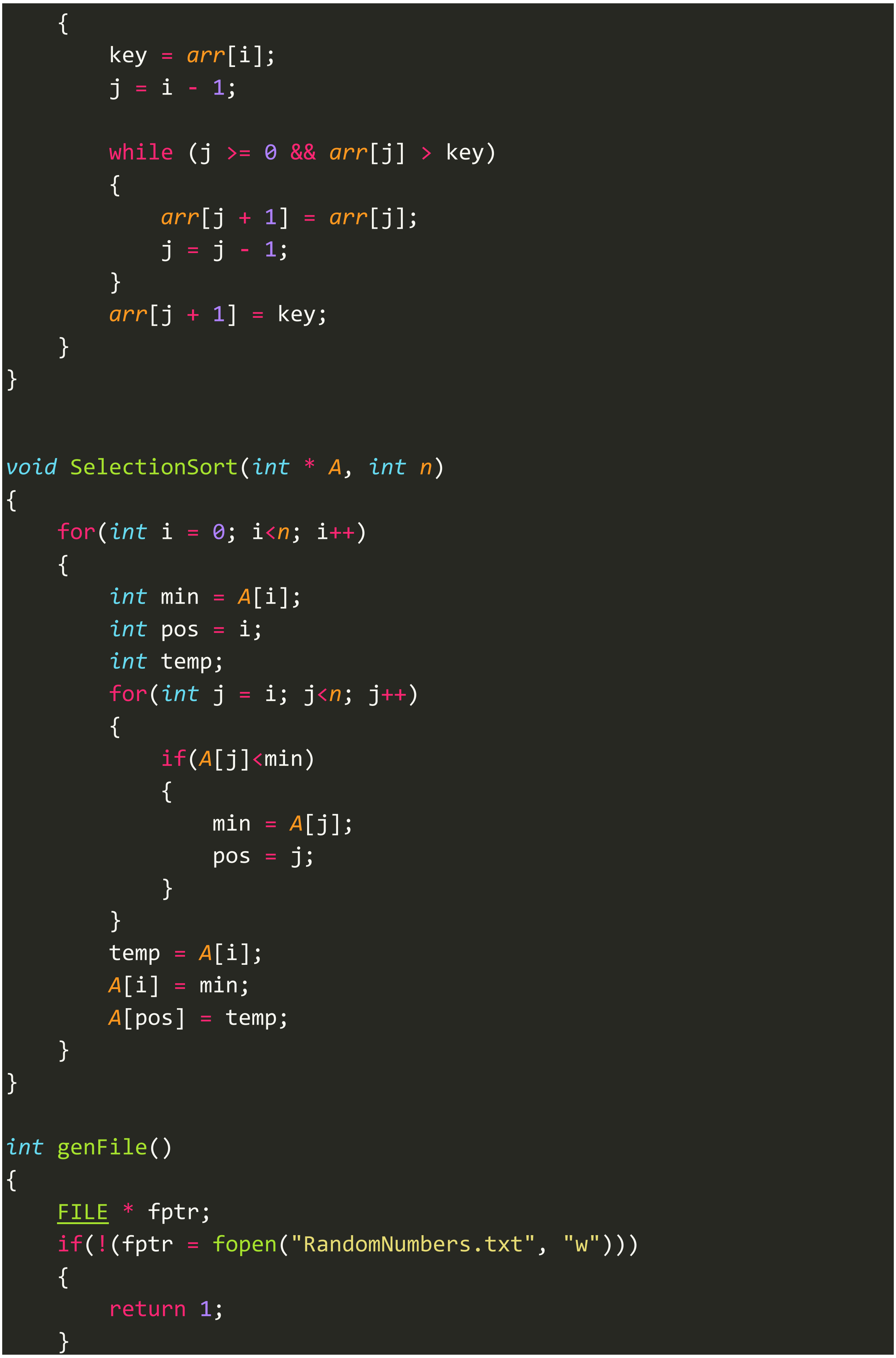
Code:

1) START
2) Create a file containing 100000 random numbers (unsorted).
3) Load the first 100, first 200, first 300 (… and so on) elements of the file into an array A of the same size as the number of elements.
4) Duplicate A to get B.
5) Calculate the time taken to run Selection Sort on A and Insertion sort on B for x number of elements in array.
6) Output time taken for both algorithms along with number of elements present to a file, preferably .csv for easy import into Excel.
7) Run Step 5 till x = 100000, incrementing x in steps of 100
8) STOP

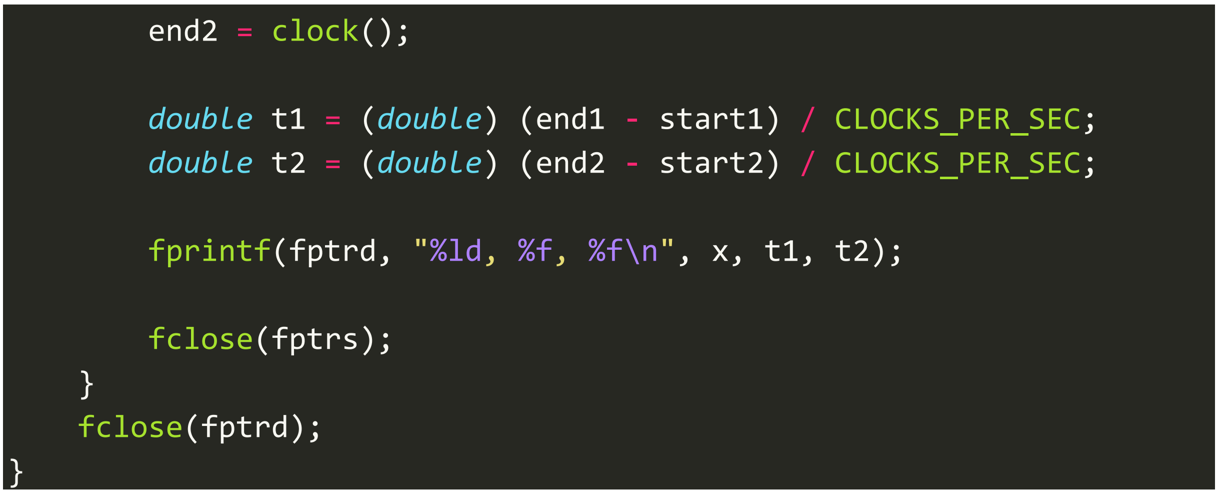
Code:



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

void print(int * A, int n)
{
    for(int i = 0; i<n; i++)
    {
        printf("%d ", A[i]);
    }
}

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

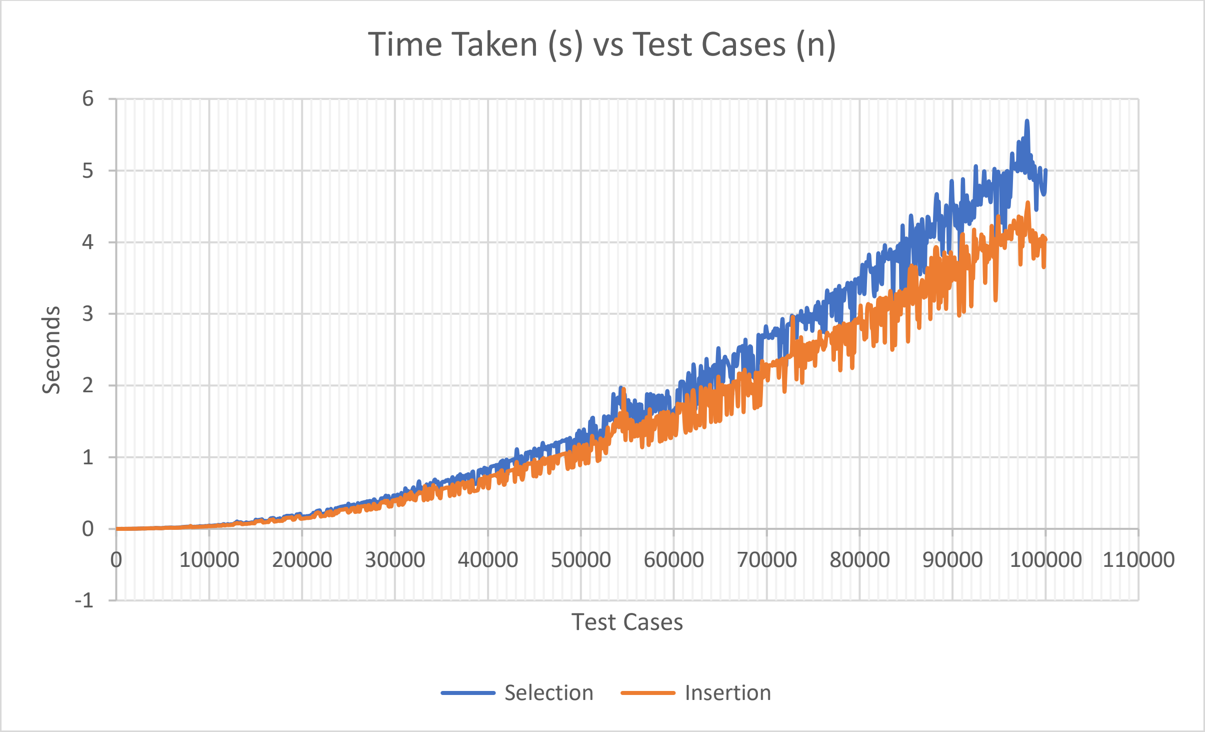


```
    {
        key = arr[i];
        j = i - 1;

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


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;
            }
        }
        temp = A[i];
        A[i] = min;
        A[pos] = temp;
    }
}

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;
    fptrd = fopen("times.csv", "w");
    if(!fptrd)
        return 0;
    fprintf(fptrd, "Cases, Selection, Insertion\n");

    for(long int x = 100; x<=100000; x+=100)
    {
        fptrs = fopen("RandomNumbers.txt", "r");
        int A[x], B[x];
        clock_t start1, end1, start2, end2;
        if(x % 1000 == 0)
        {
            printf("x = %ld.\n", x);
        }

        for(long int i = 0; i<x; i++)
        {
            fscanf(fptrs, "%d\n", &A[i]);
            B[i] = A[i];
        }

        start1 = clock();
        SelectionSort(A, x);
        end1 = clock();

        start2 = clock();
        InsertionSort(B, x);
```



```
        end2 = clock();

        double t1 = (double) (end1 - start1) / CLOCKS_PER_SEC;
        double t2 = (double) (end2 - start2) / CLOCKS_PER_SEC;

        fprintf(fptrd, "%ld, %f, %f\n", x, t1, t2);

        fclose(fptrs);
    }
    fclose(fptrd);
}
```

Graph



Observations

Complexity	Insertion	Selection
Time	Worst: $O(N^2)$ Best: $O(N)$	$O(N^2)$ $O(N^2)$
Space	O(1)	O(1)

The time complexity of Selection Sort is $O(N^2)$ as there are two nested loops:

- One loop to select an element of Array one by one = O(N)
- Another loop to compare that element with every other Array element = O(N)

Insertion sort takes maximum time to sort if elements are sorted in reverse order. And it takes minimum time (Order of n) when elements are already sorted.

Space Complexity of both algorithms is O(1) because they need some extra memory space for swapping operation.

Conclusion

Time & Space Complexity of Insertion & Selection Sort is investigated.