NAME:	Ayush Kedare
UID:	2021300058
SUBJECT	Design and Analysis of Algorithm
EXPERIMENT NO :	2
AIM:	Experiment on finding the running time of an algorithm.
Algorithm	• **Insertion sort-** • If the element is the first element, assume that it is already sorted. Return 1. • Pick the next element, and store it separately in a **key.** • Now, compare the **key** with all elements in the sorted array. • If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right. • Insert the value. • Repeat until the array is sorted. • **Selection sort-** • Repeat Steps 2 **and** 3 **for** i = 0 to n-1 • CALL SMALLEST(arr, i, n, pos) • SWAP arr[i] with arr[pos] • [END OF LOOP] • EXIT • SMALLEST (arr, i, n, pos) • [INITIALIZE] SET SMALL = arr[i] • [INITIALIZE] SET pos = i • Repeat **for** j = i+1 to n • **if** (SMALL > arr[j]) • SET SMALL = arr[j] • SET pos = j • [END OF **if**] • [END OF LOOP] • RETURN pos

PROGRAM:

```
#include <bits/stdc++.h>
#include <fstream>
using namespace std;

void insertionsort(vector<int> arr, int num)
{
    int key, j;
    for (int i = 1; i < num; i++)
    {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

void selectionsort(vector<int> arr, int num)
{
    int key;
    for (int i = 0; i < num - 1; i++)
    {
        key = i;
        for (int j = i + 1; j < num; j++)
        {
            if (arr[j] < arr[key])
            {
                key = j;
            }
        }
        int temp;
        temp = arr[key];
        arr[key] = arr[i];
        arr[i] = temp;
```

```
    }
}

int main()
{
    vector<int> arr;
    clock_t t1, t2, t3, t4;
    string filename("values.txt");
    ifstream fin(filename);
    if (!fin.is_open())
    {
        cerr << "Could not open the file - '"
             << filename << "'" << endl;
        return EXIT_FAILURE;
    }

    while (!fin.eof())
    {
        int tmp;
        fin >> tmp;
        arr.push_back(tmp);
    }

    int num = 100;
    for (int i = 0; i < 150; i++)
    {
        t1 = clock();
        insertionsort(arr, num);
        t2 = clock();
        t3 = clock();
        selectionsort(arr, num);
        t4 = clock();
        double insertiontime = double(t2 - t1) /
double(CLOCKS_PER_SEC);
        double selectiontime = double(t4 - t3) /
double(CLOCKS_PER_SEC);
        cout << endl;
```

```
        cout << i + 1 << "  " << fixed <<
insertiontime << setprecision(5) << '\t';
        cout << fixed << selectiontime <<
setprecision(5);

        num += 100;
    }
    fin.close();
    return 0;
}
```

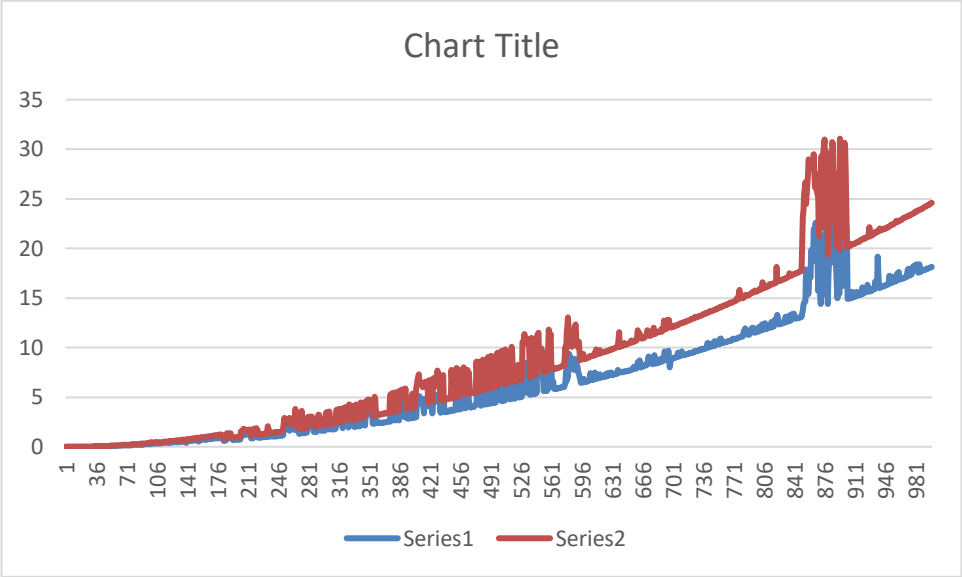
Values.cpp

```
#include <iostream>
#include <cstdlib>
#include <cstdio>
using namespace std;

int main()
{
    for (int i = 1; i <= 100000; i++)
    {
        cout << rand() << " ";
    }

    return 0;
}
```

Observation (SNAPSHOT)



Observation

For insertion sort as the size of the input data increases, the running time also increases.

For selection sort also it increases as the size of the input data increases, similar to insertion sort, but the rate of increase is relatively slower.

Conclusion

By performing the above experiment I understood the working of insertion and selection sort and also calculated their time complexities and found out that it is dependent on the size of the input data.