NAME:	Shubham Solanki
UID:	2022301015
SUBJECT	Design and Analysis of Algorithm
EXPERIMENT NO :	1 (B)
AIM:	Experiment on finding the running time of an algorithm.

Algorithm

1. **Insertion sort-**

```
a. procedure insertionSort(A: list of sortable items)
b.    n = length(A)
c.    for i = 1 to n - 1 do
d.        j = i
e.        while j > 0 and A[j-1] > A[j] do
f.            swap(A[j], A[j-1])
g.            j = j - 1
h.        end while
i.    end for
j. end procedure
```

2. **Selection sort-**

```
a. Repeat Steps b and c for i = 0 to n-1
b. CALL SMALLEST(arr, i, n, pos)
c. [INITIALIZE] SET key = i
d. Repeat for j = i+1 to n
e. if (arr[key] > arr[j])
f. SET  key=j
g. [END OF if]
h. [END OF LOOP]
i. SWAP arr[i] with arr[pos]
j. [END OF LOOP]
k. EXIT
```

PROGRAM:

```
Insertion and Selection Sort

#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");
     ofstream output("./output.csv");
    output << "block_size,insertion,selection\n";
    ifstream fin(filename.c_str());
    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 <1000; 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;
        output << i+1 <<","<<fixed <<
insertiontime << setprecision(5) << '\t';
        output << fixed << ","  <<selectiontime <<
setprecision(5) <<"\n";
        cout << i+1 <<","<<fixed << insertiontime
<< setprecision(5) << '\t';
        cout << fixed << ","  <<selectiontime <<
setprecision(5) <<"\n";

        num += 100;
    }
    cout<<"\n Sorting done";
    fin.close();
    return 0;
}
```

RandomValues.cpp

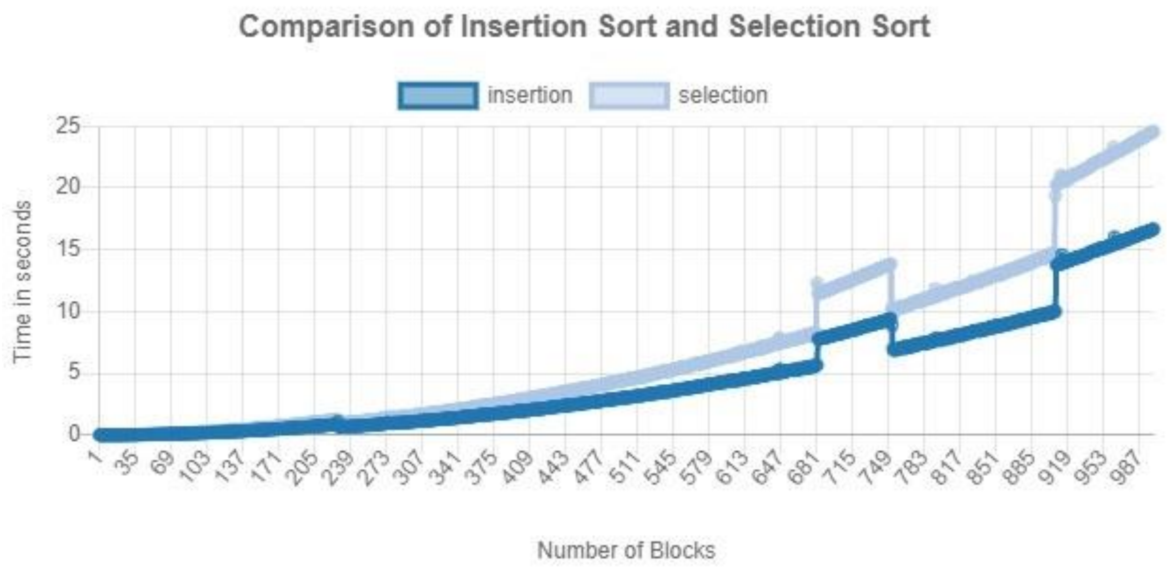
```
#include <iostream>
#include <cstdlib>

using namespace std;

int main() {

    for(int i = 0; i < 100000; i++) {
        cout<<"\n "<<rand();
    }
}
```

Observation (SNAPSHOT)



Observation

For insertion sort

- The insertion sort running time graph shows that the running time increases steadily and consistently as the size of the input data increases.
- The graph shows a recognisable increasing trend, with the slope of the curve steepening as the input data size grows.
- This demonstrates the quadratic nature of insertion sort's running time.

For selection sort

- The graph of the running time of selection sort shows that, like insertion sort, it increases with increasing input data size.
- The rate of rise is still significantly slower than insertion sort, despite the graph's less extreme slope.
- This demonstrates that selection sort is more efficient for small input sizes

Conclusion

The experiment on finding the running time of insertion sort and selection sort shows that the running time of both algorithms is dependent on the size of the input data. The running time of insertion sort is quadratic in nature, while that of selection sort is also quadratic but performs better on small inputs.