

|  |  |
| --- | --- |
| Name | Vedant Deshmukh |
| UID No. | 2021300025 |
| Class | COMPS A (B batch) |
| Experiment No. | 01 B |

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

Algorithm:

InsertionSort(A)

For i = 2 to A.length

key = A[i] j = i -1

while j > 0 and A[j] > key A[j + 1] = A[j]

j = j – 1 A[j + 1] = key

SelectionSort(A)

For i = 1 to A.length – 1 i\_min = i

for j = i +1 to A.length

if (A[j] < A[i\_min])

i\_min = j

if i != i\_min

swap(A[i], A[i\_min])



Code:

#include <stdio.h> #include <chrono> #include <cmath> #include <fstream> #include <iomanip> #include <iostream> using namespace std;

double insertion\_comparision = 0.0; double selection\_comparision = 0.0;

void insertion\_sort(int\* arr, int size) { for (int i = 1; i < size; i++) {

int current = arr[i]; int j = i - 1;

while (j >= 0 && current < arr[j]) { arr[j + 1] = arr[j];

j--;

insertion\_comparision++;

}

arr[j + 1] = current;

}

}

void swap(int& a, int& b) { int temp = a;

a = b;

b = temp;

}

void selection\_sort(int\* arr, int size) { for (int i = 0; i < size - 1; i++) {

int i\_min = i;

for (int j = i + 1; j < size; j++) { selection\_comparision++;

if (arr[j] < arr[i\_min]) { i\_min = j;

}

}

if (i != i\_min) {



swap(arr[i], arr[i\_min]);

}

}

}

int digits(int num) {

return num == 0 ? 1 : floor(log10(abs(num))) + 1;

}

int main() {

int arr\_ins[100000]; int arr\_sel[100000];

ifstream nums("random\_numbers.txt"); ofstream output("../csv/sort\_analysis.csv");

output << "block\_size,insertion,selection\n";

for (int i = 1; i <= 100000; i++) { int val = 0;

nums >> val; arr\_ins[i] = val; arr\_sel[i] = val;

}

// 1000 blocks of 100 numbers

for (int i = 1; i <= 1000; i++) {

// print 10 values at index 10000, 20000, ... int index = i \* 100;

if (index % 10000 == 0 && index != 100000) {

cout << "\nPrinting 10 values from index " << index << endl; for (int t = 0; t < 10; t++) {

cout << index + t << " : " << arr\_ins[index + t] << "\n";

}

}

// insertion

auto ins\_start = chrono::high\_resolution\_clock::now(); insertion\_sort(arr\_ins, i \* 100);

auto ins\_end = chrono::high\_resolution\_clock::now(); chrono::duration<double> ins\_time = (ins\_end - ins\_start);



// selection

auto sel\_start = chrono::high\_resolution\_clock::now(); selection\_sort(arr\_sel, i \* 100);

auto sel\_end = chrono::high\_resolution\_clock::now(); chrono::duration<double> sel\_time = (sel\_end - sel\_start);

output << i \* 100 << "," << ins\_time.count() << "," << sel\_time.count() << "\n";

}

cout << "Sorting completed !" << endl;

cout << "\nSmallest Number = " << arr\_ins[0] << "\tDigits = " << digits(arr\_ins[0]) << endl;

cout << "Largest Number = " << arr\_ins[99999] << "\tDigits = " << digits(arr\_ins[99999]) << endl;

printf("Insertion sort comparision count: %.0lf\n", insertion\_comparision);

printf("Selection sort comparision count: %.0lf\n", selection\_comparision);

return 0;

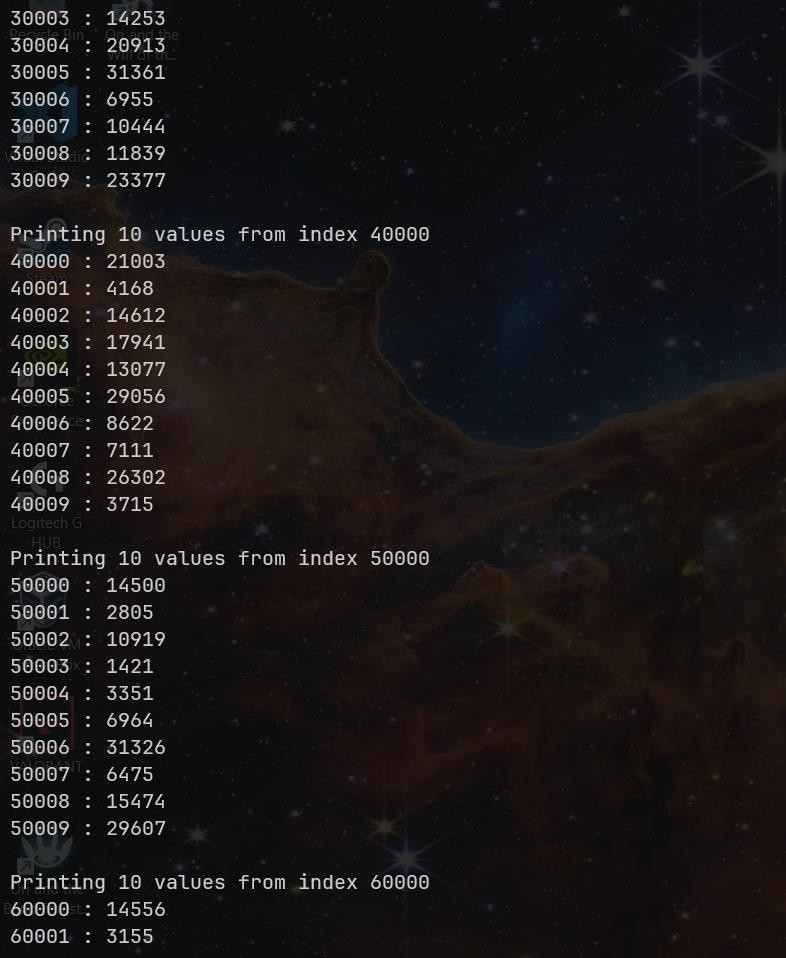
}



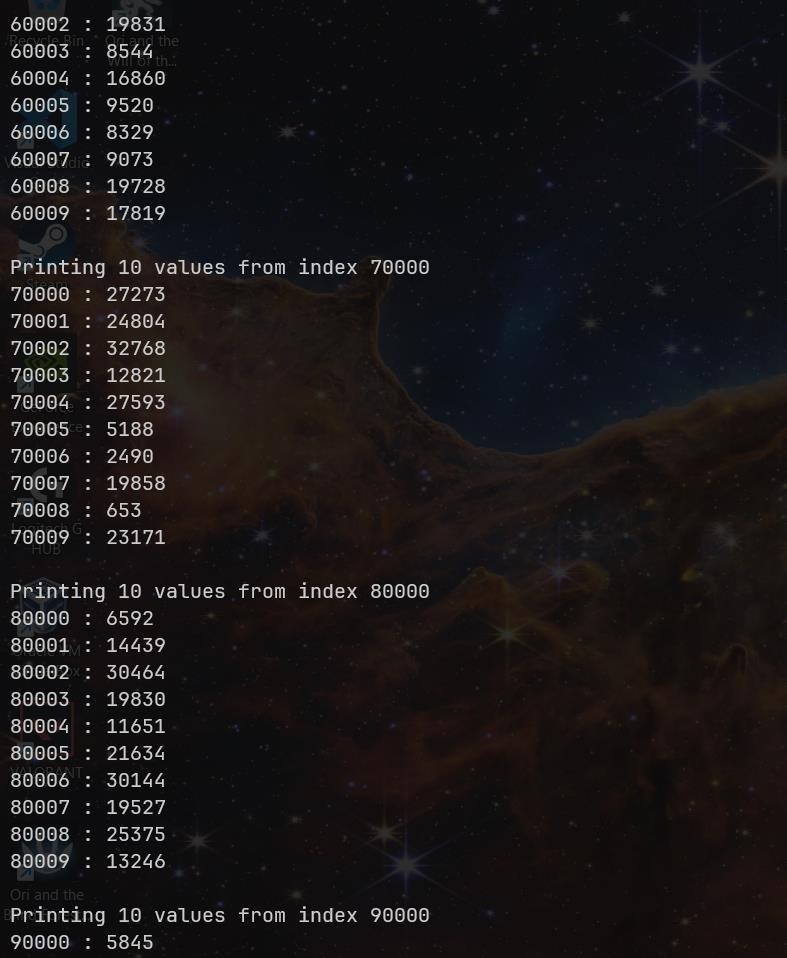
Output:



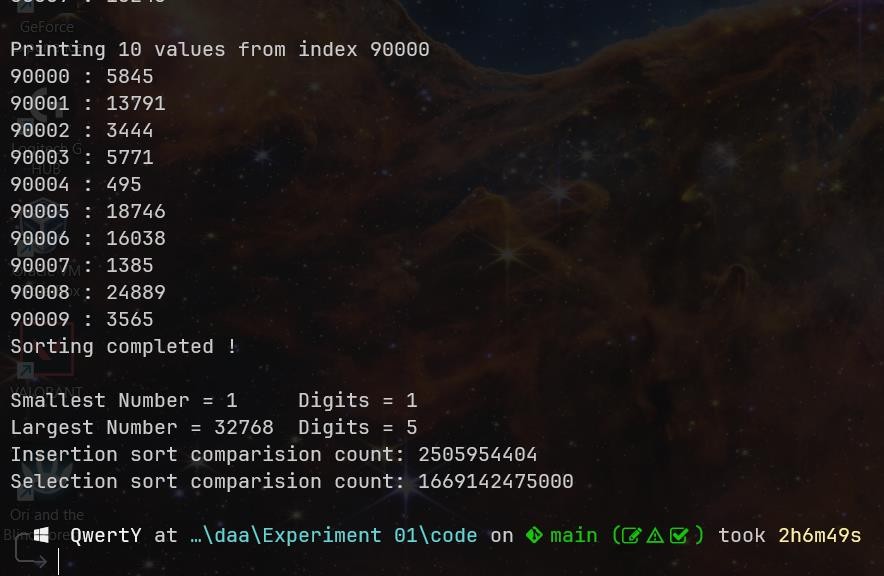


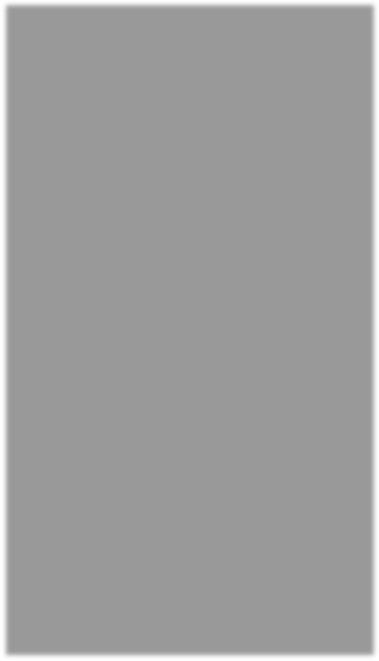












Time (sec)

100

2900

5700

8500

11300

14100

16900

19700

22500

25300

28100

30900

33700

36500

39300

42100

44900

47700

50500

53300

56100

58900

61700

64500

67300

70100

72900

75700

78500

81300

84100

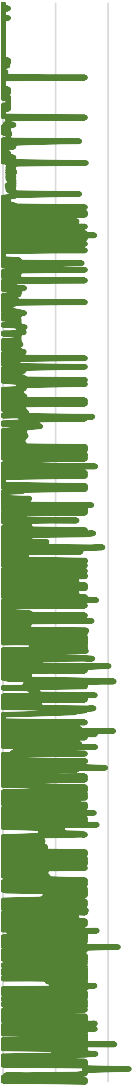
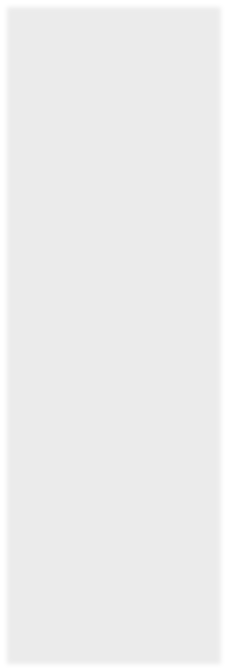
86900

89700

92500

95300

98100



Time (sec)

100

2900

5700

8500

11300

14100

16900

19700

22500

25300

28100

30900

33700

36500

39300

42100

44900

47700

50500

53300

56100

58900

61700

64500

67300

70100

72900

75700

78500

81300

84100

86900

89700

92500

95300

98100

**Bharatiya Vidya Bhavan’s**

**Sardar Patel Institute of Technology**

Bhavan’s Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India

(Autonomous College Affiliated to University of Mumbai)

Chart:

Comparision of Insertion Sort and Selection Sort

12

10

8

6

4

2

0

Size of array (n)

insertion

selection

Insertion Sort

0.03

0.02

0.01

0

Size of array (n)



**Bharatiya Vidya Bhavan’s**

# Sardar Patel Institute of Technology

Bhavan’s Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India

(Autonomous College Affiliated to University of Mumbai)

Observations:

1. Initially, both sorting techniques take the same time.
2. As the size of blocks increase, I observe that selection sort begins to take more time. However, insertion sort takes almost the same amount of time at every block
3. The graph observed for selection sort resembles a scaled down quadratic curve.
4. The graph observed for insertion sort resembles a line.
5. Insertion sort works by maintaining two subarrays, one is always sorted while other is not. This problem statement mimics the behavior of insertion sort as in the only the new 100 elements are completely random while the all the previous elements are partially sorted.
6. Selection sort works by finding the smallest element in the unsorted subarray and replacing it with current element. In order to determine if an element is truly the smallest element, it is necessary to check all elements.

## Conclusion:

* Insertion sort works by maintaining two subarrays, one of which always contain sorted elements. If an element from unsorted array finds its appropriate position in sorted array, we go on to the next iteration.
* The current problem statement, that goes on adding 100 unsorted elements on already sorted array, greatly favors in how insertion sort works.
* Selection sort works by again maintaining two subarrays. One is always sorted while other isn’t. However, in selection sort we find the smallest element in unsorted array and swap it with current element.
* In order to find the smallest element, it is necessary to traverse the entire unsorted subarray. Hence each addition of 100 elements as per problem statement, just increases the number of comparisons for selection sort.

After conducting this experiment and analyzing the time for both sorting techniques, I conclude that the nature of problem statement and initial state of data affects greatly on the runtime on algorithm even if they have the same worst case time complexity.