**INFO 6205**

**Program Structures & Algorithms**

**Summer Full 2018**

**Assignment 3**

This program is a benchmark program for sorting algorithms. It is able to run a benchmark on any sorting algorithm and display the average counts and time for each sort.

This assignment is divided into three parts :

1. To implement the run method
2. To implement the sorting algorithms (Insertion Sort, Selection Sort)
3. To implement the sorting with different array ordering situations i.e Sorted array, Partially sorted array , Reverse sorted array and Random array.

**Abbrevations:**

Some useful abbreviations:

* n – Size of array (n=10000)
* m – No of repetitions (m=100)
* array – Array to be sorted
* T – Generic input to the function f

**Approach**

1. I started with the second part of experiment by implementing Insertion Sort and Selection Sort algorithm and running them for small values of n with m being 1 just to check the working of algorithm.
2. Then I implemented the first part i.e implemented the run method to calculate the average time (Total time/ m) to sort the array using system clock System.nanos.
3. Algorithms were ran for different ordering situations of array in order to compare the two different algorithms by using Random class.

**Observations :**

Following are the results for various values of N by using doubling method (n=n\*2) while M= 100 to calculate average time.

For Random Array :

|  |  |  |  |
| --- | --- | --- | --- |
| Ordering of Array | N | Time for Insertion Sort(nanosecs) | Time for Selection Sort(nanosecs) |
| Random | 10000 | 6.97E+07 | 6.33E+07 |
| Random | 20000 | 7.38E+08 | 7.80E+08 |
| Random | 40000 | 3.57E+09 | 2.89E+09 |
| Random | 80000 | 6.51E+09 | 5.79E+09 |

Time taken by both the sorts for random array is almost same.

For Partially Sorted Array :

|  |  |  |  |
| --- | --- | --- | --- |
| Ordering of Array | N | Time for Insertion Sort (nanosecs) | Time for Selection Sort(nanosecs) |
| Partially Sorted | 10000 | 142306.27 | 1.85E+08 |
| Partially Sorted | 20000 | 179076.76 | 1.43E+09 |
| Partially Sorted | 40000 | 668062.83 | 6.31E+09 |
| Partially Sorted | 80000 | 694732.12 | 1.28E+10 |

Insertion sort works excellent with partially sorted array with almost linear time complexity while selection sort has poor performance for almost sorted array as it needs to find the minimum value by scanning the whole array with less number of swaps.

For Reverse Sorted Array

|  |  |  |  |
| --- | --- | --- | --- |
| Ordering of Array | N | Time for Insertion Sort (nanosecs) | Time for Selection Sort(nanosecs) |
| Reverse Sorted | 10000 | 2.03E+08 | 1.87E+08 |
| Reverse Sorted | 20000 | 1.73E+09 | 1.23E+09 |
| Reverse Sorted | 40000 | 6.96E+09 | 4.91E+09 |
| Reverse Sorted | 80000 | 1.40E+11 | 0.81E+11 |

Insertion sort shows poor performance for reverse sorted array as it needs to traverse till the end to swap the least element for the first place. Of all the ordering situations, Insertion sort takes maximum time only for reverse sorted arrays.

Selection sort works almost similar for reversed sorted array as for partially sorted but better than Insertion sort.

Sorted Array

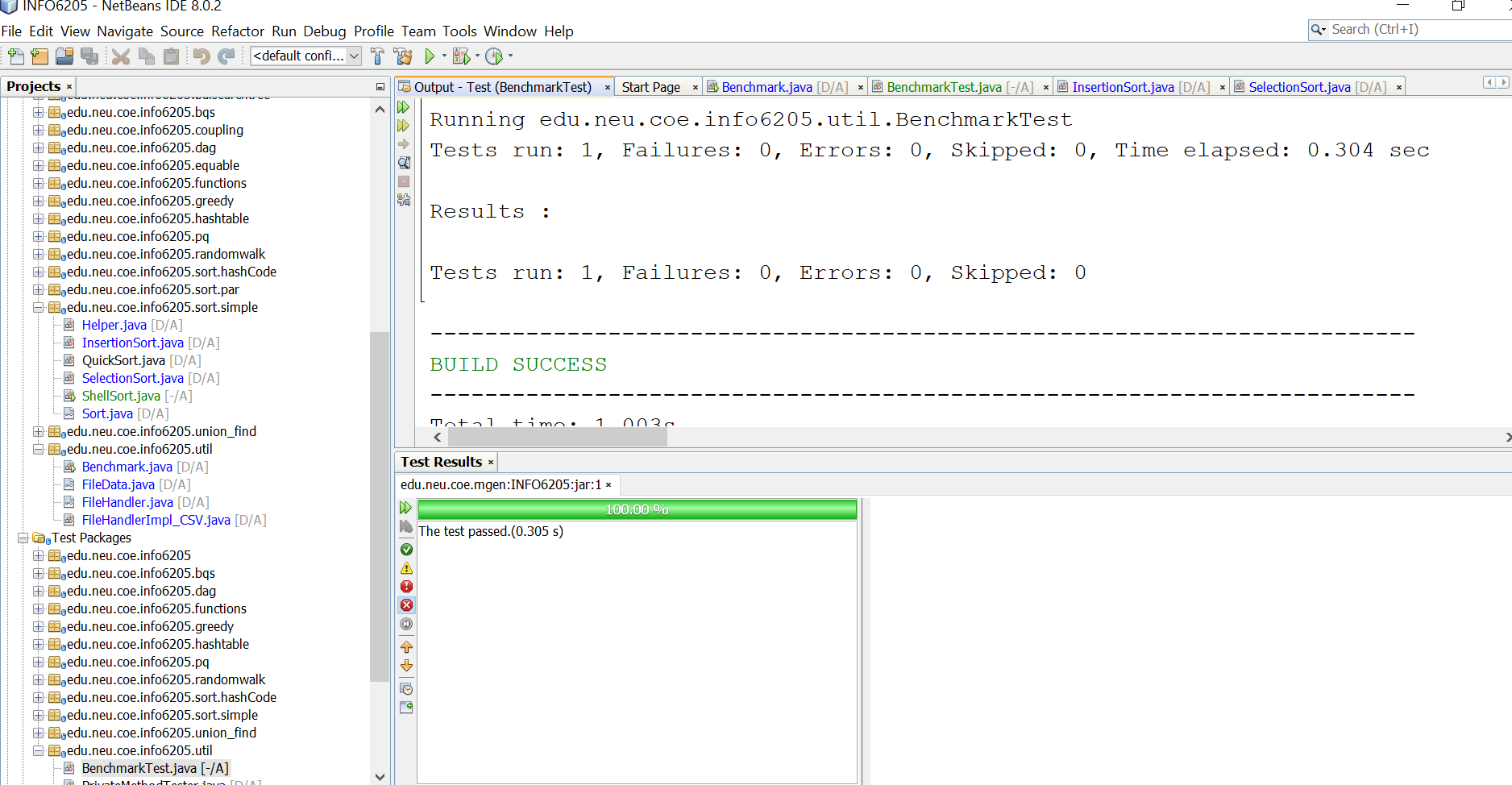
|  |  |  |  |
| --- | --- | --- | --- |
| Ordering of Array | N | Time for Insertion Sort (nanosecs) | Time for Selection Sort(nanosecs) |
| Sorted | 10000 | 41032 | 2.06E+08 |
| Sorted | 20000 | 104034.76 | 9.23E+08 |
| Sorted | 40000 | 368979.55 | 4.21E+09 |
| Sorted | 80000 | 446432.44 | 4.648E+09 |

For already sorted array, Insertion sort is best as it takes almost linear time O(N)

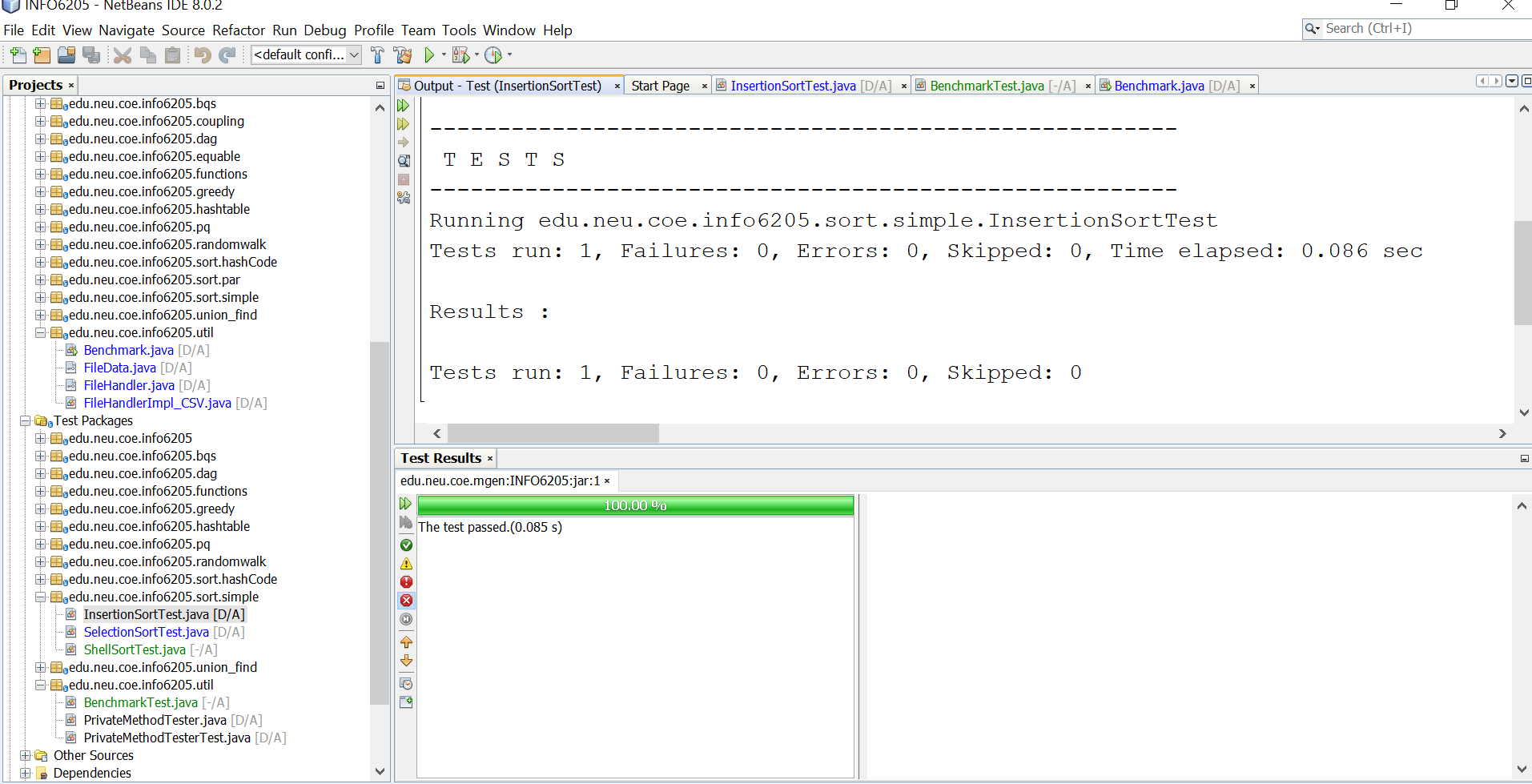
to sort an array of N elements while at the same time selection sort shows poor performance as compared to Insertion sort.

**Test Case Results :**

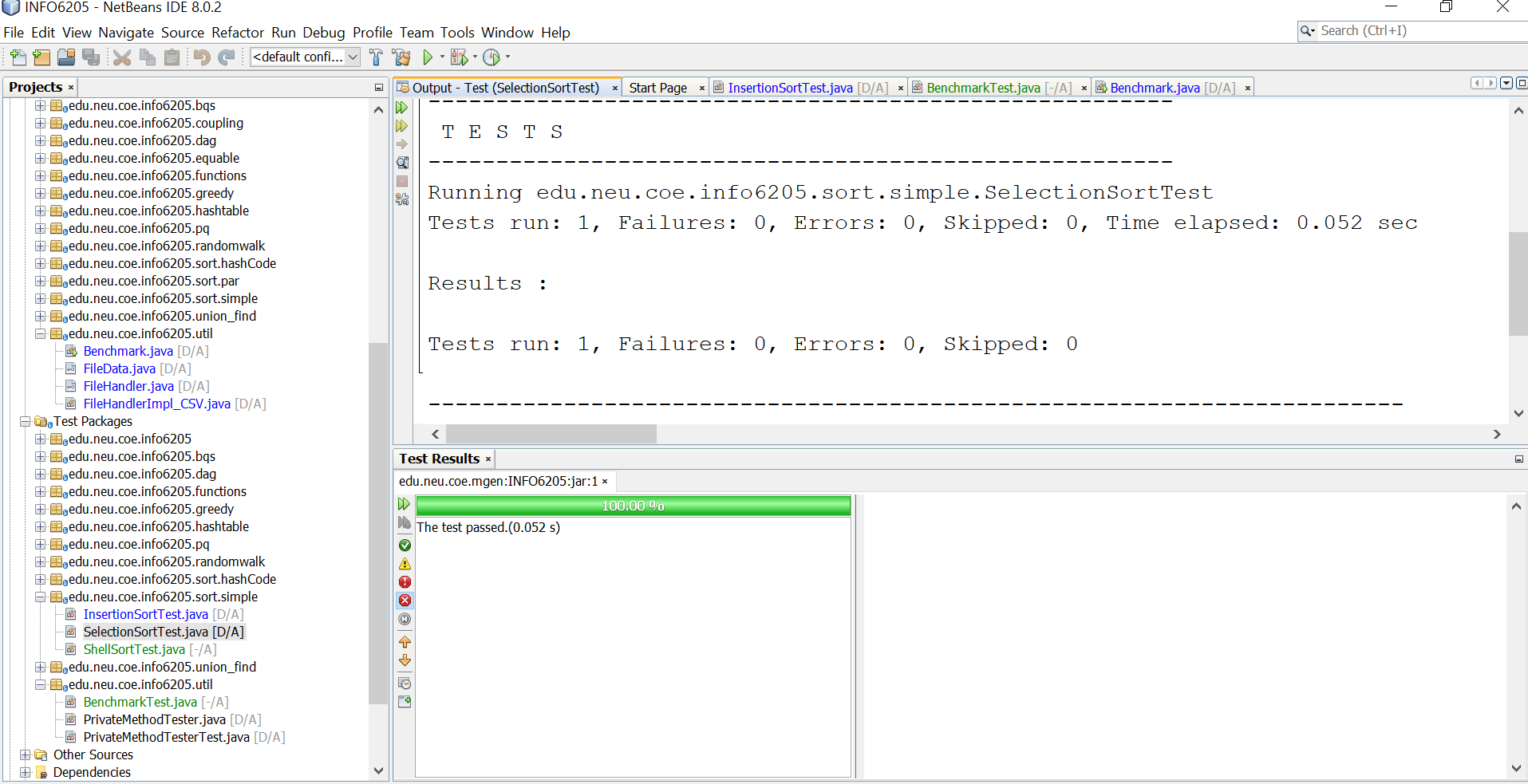
Benchmark.Test



InsertionSort Test File :



Selection Sort Test File :



**Conclusion :**

Hence, Insertion sort works better as compared to Selection sort except for the Reverse sorted array.