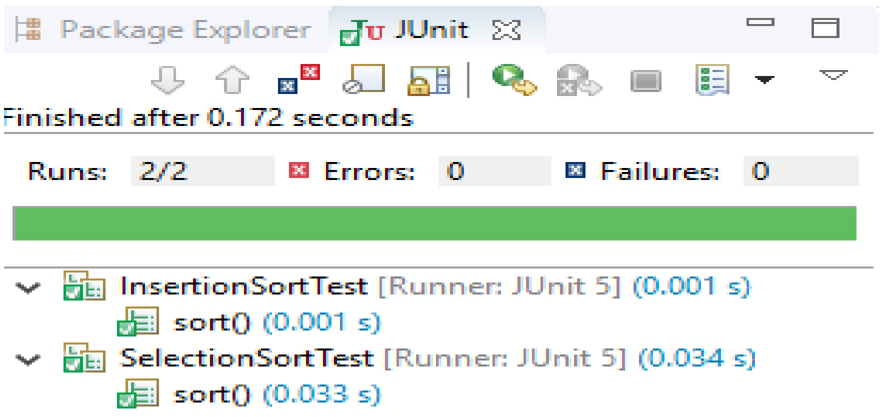
Benchmark Sort

Vikram Bharadwaj Ramesh

Problem Statement:

Implement a Benchmark Class and calculate the time taken to execute Insertion and Selection Sort by sorting the elements and running the instance for 100 times. Elements need to be considered randomly, ordered, reverse ordered and partially ordered for analysis.

Unit Test Cases:



```
Finished after 0.172 seconds
Runs: 2/2    Errors: 0    Failures: 0
InsertionSortTest [Runner: JUnit 5] (0.001 s)
    sort() (0.001 s)
SelectionSortTest [Runner: JUnit 5] (0.034 s)
    sort() (0.033 s)
```

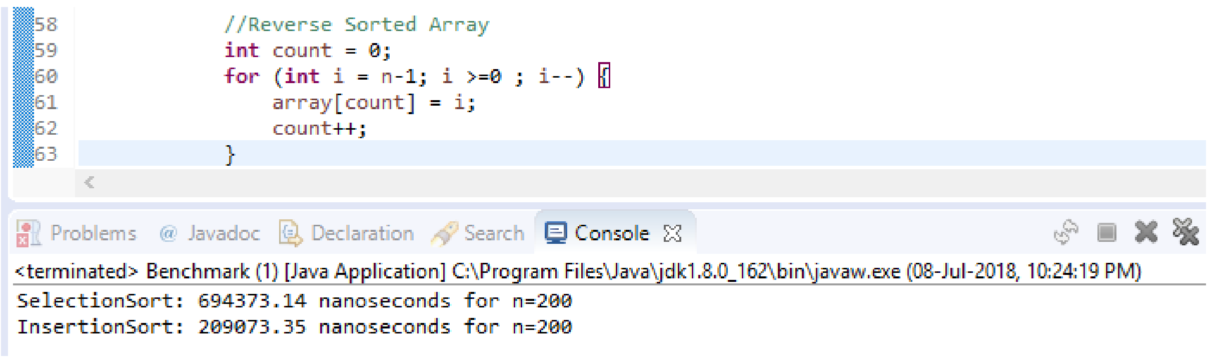
Analysis:

Benchmark class consists of a constructor, “run” and “benchmarkSort” methods. The constructor is used to initialize the function where the sort functionality is called. run() method is used to calculate the time taken for each sort methods in order for analysis.

Number of elements(N) taken for the analysis differs from 200 to 1000. Each value of N is being run for 100 times to get the mean value.

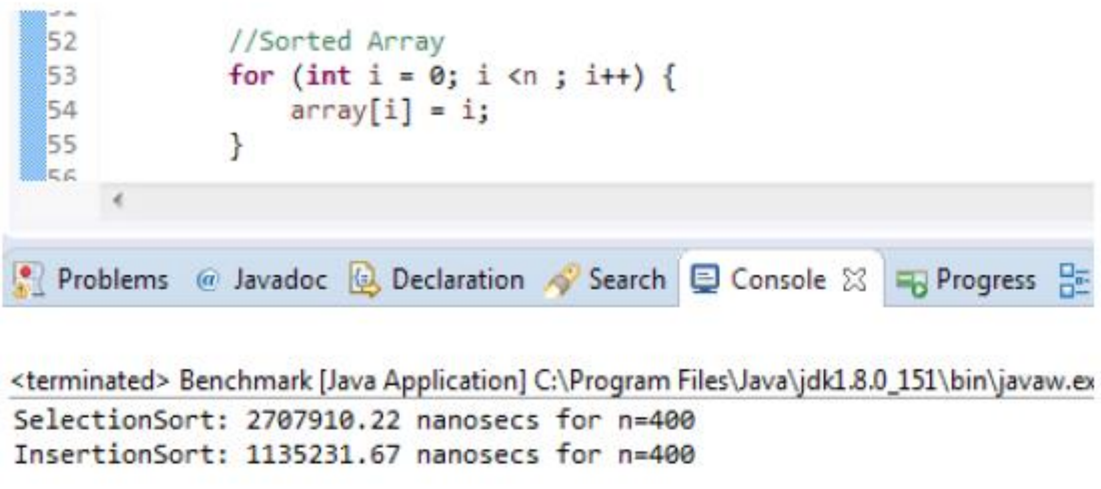
NOTE: The time for all analysis has been calculated in nanoseconds

Here are some of the screenshots for the different outputs:



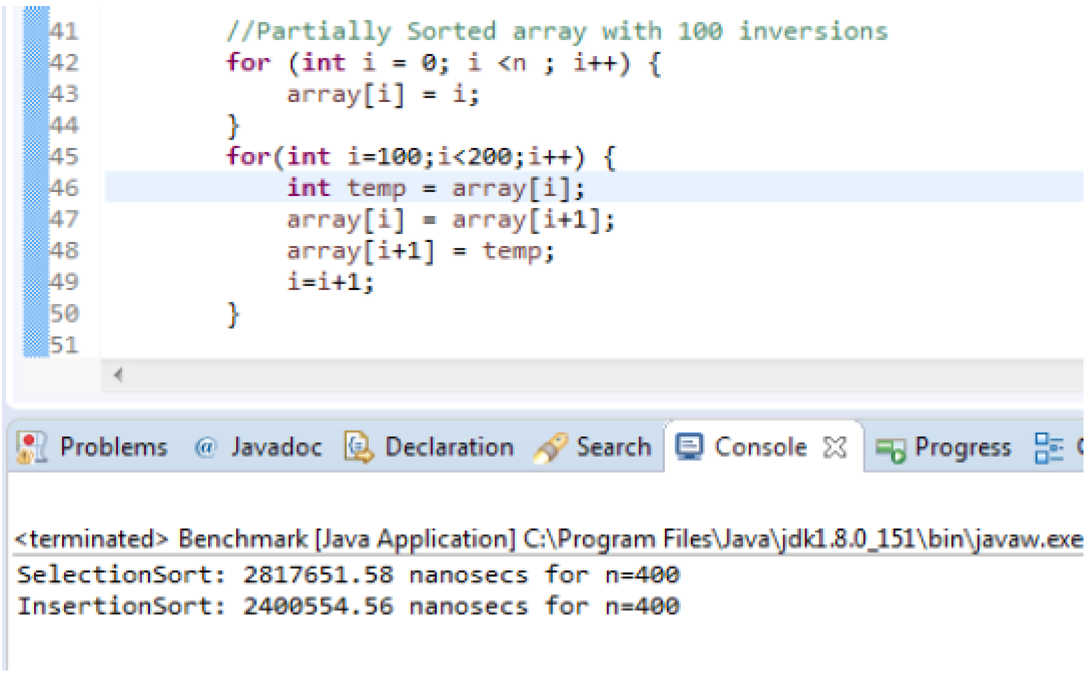
```
            //Reverse Sorted Array
            int count = 0;
            for (int i = n-1; i >=0 ; i--) {
                array[count] = i;
                count++;
            }
```

```
<terminated> Benchmark (1) [Java Application] C:\Program Files\Java\jdk1.8.0_162\bin\javaw.exe (08-Jul-2018, 10:24:19 PM)
SelectionSort: 694373.14 nanoseconds for n=200
InsertionSort: 209073.35 nanoseconds for n=200
```



```
        //Sorted Array
        for (int i = 0; i <n ; i++) {
            array[i] = i;
        }
```

```
<terminated> Benchmark [Java Application] C:\Program Files\Java\jdk1.8.0_151\bin\javaw.ex
SelectionSort: 2707910.22 nanosecs for n=400
InsertionSort: 1135231.67 nanosecs for n=400
```



```
        //Partially Sorted array with 100 inversions
        for (int i = 0; i <n ; i++) {
            array[i] = i;
        }
        for(int i=100;i<200;i++) {
            int temp = array[i];
            array[i] = array[i+1];
            array[i+1] = temp;
            i=i+1;
        }
```

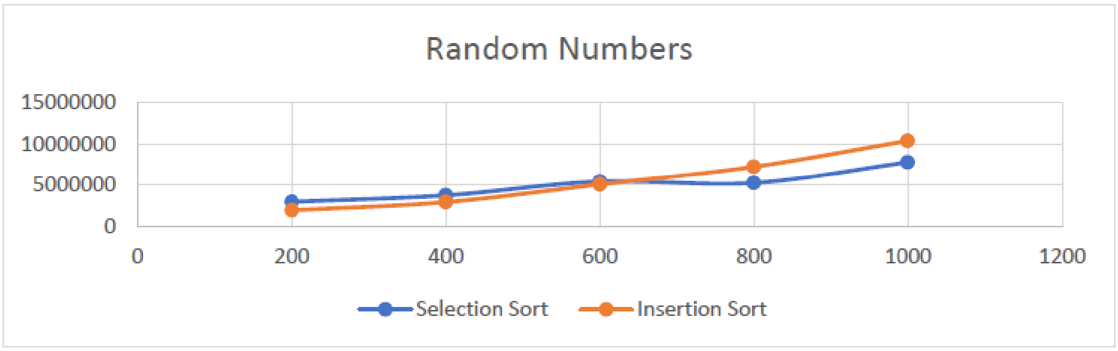
```
<terminated> Benchmark [Java Application] C:\Program Files\Java\jdk1.8.0_151\bin\javaw.exe
SelectionSort: 2817651.58 nanosecs for n=400
InsertionSort: 2400554.56 nanosecs for n=400
```

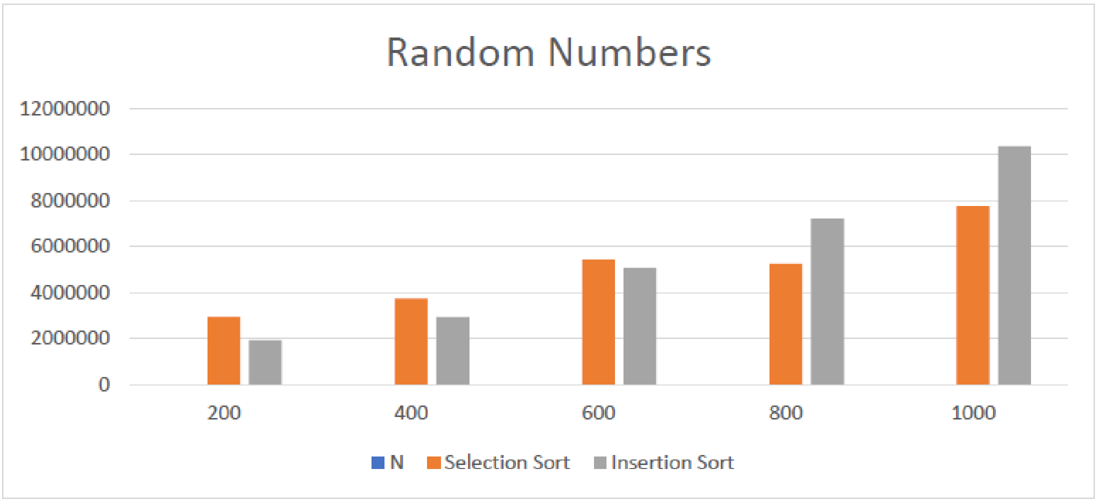
Evidence and Observations:

1. *Randomly Sorted Array:*

Type of Array	Number of runs	Number of elements	Time taken by Selection Sort (ns)	Time taken by Insertion Sort(ns)
Random Numbers	***100***	***200***	2958506.65	1938513.27
		400	3752346.1	2931235.48
		600	5420537.3	5071480.53
		800	5257509.77	723535.63
		1000	7755297.08	10346745.44

Graph based on above values:





From the above observations, we can see that selection sort is taking less time for less number of elements(N). As the number of elements increases, Insertion sort will take more time compared to Selection sort.

Random numbers are generated by considering the in-built Random class and the array elements are considered up to “N”(random.nextInt(N)) which is varied

Hence, we can conclude that for randomly generated numbers,

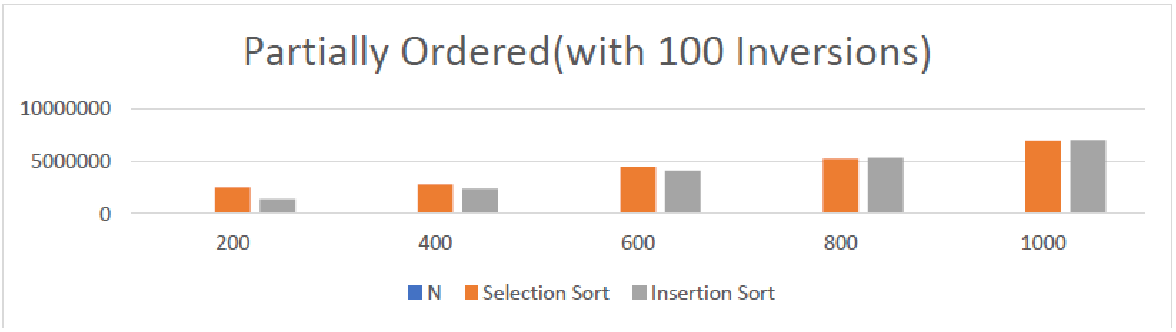
- Selection Sort takes more time for less values of “N”
- Insertion sort takes more time for more value of “N”.

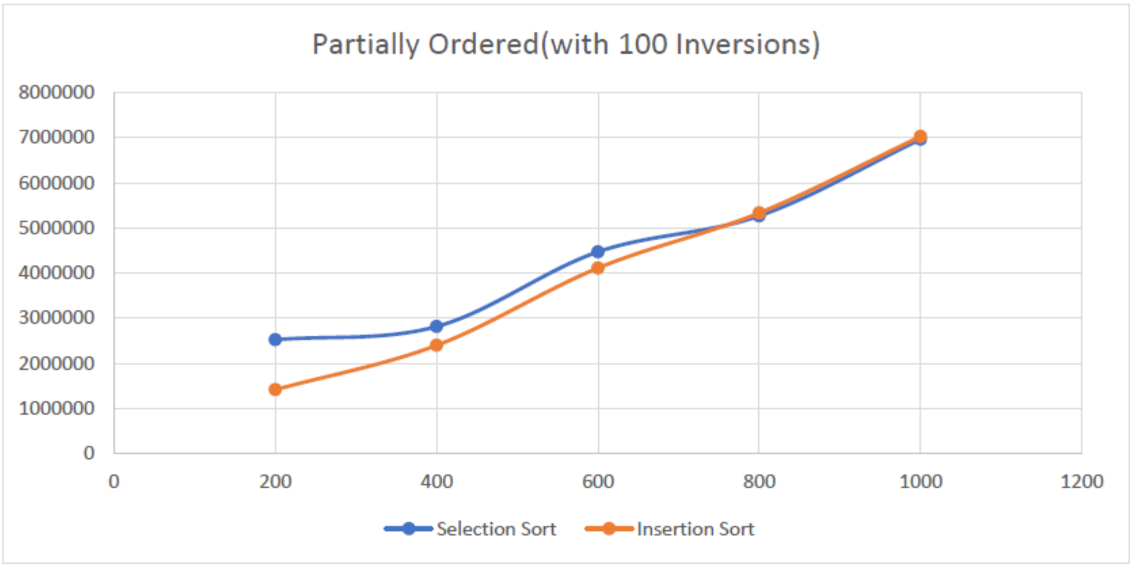
2. Partially Sorted Array:

Time taken for each sort

Type of Array	Number of runs	Number of elements	Time taken by Selection Sort (ns)	Time taken by Insertion Sort(ns)
Random Numbers	***100***	***200***	2524697.21	1417820.38
		400	2817651.58	2400554.56
		600	4471457.86	4112963.16
		800	5268832.03	5334799.25
		1000	6964249.01	7027987.03

Graph based on above values:





Array is called partially sorted based on number of inversion in the elements. For this experiment, I have considered 100 inversions in the array for all the values of “N”. Inversions are made for each element from 100th to 200th element in the analysis.

From the above observations, we can see that for partially sorted arrays:

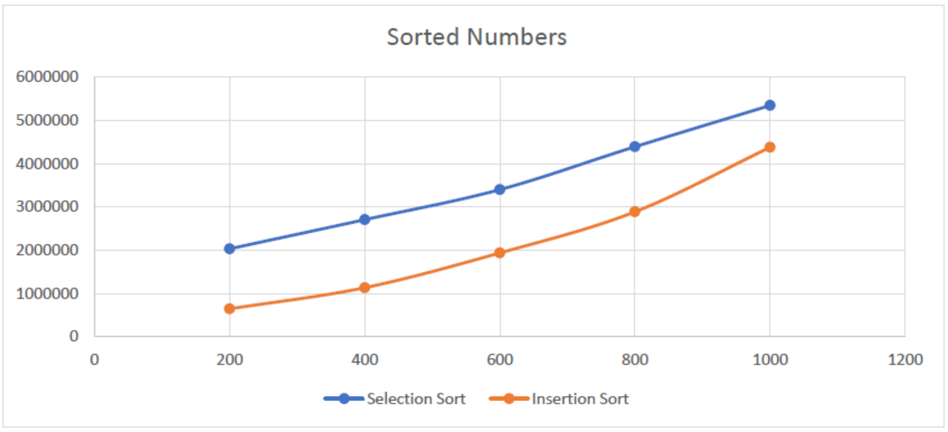
- Insertion sort for the partially sorted array is linear since, the number of inversions will be equal to the number of exchanges that occur during the sort.
- We cannot define inversion for selection sort because exchanges happen irrespective of checking entire loop on its left. The sort functionality works based on the minimum and maximum values by checking the elements from starting till the end.

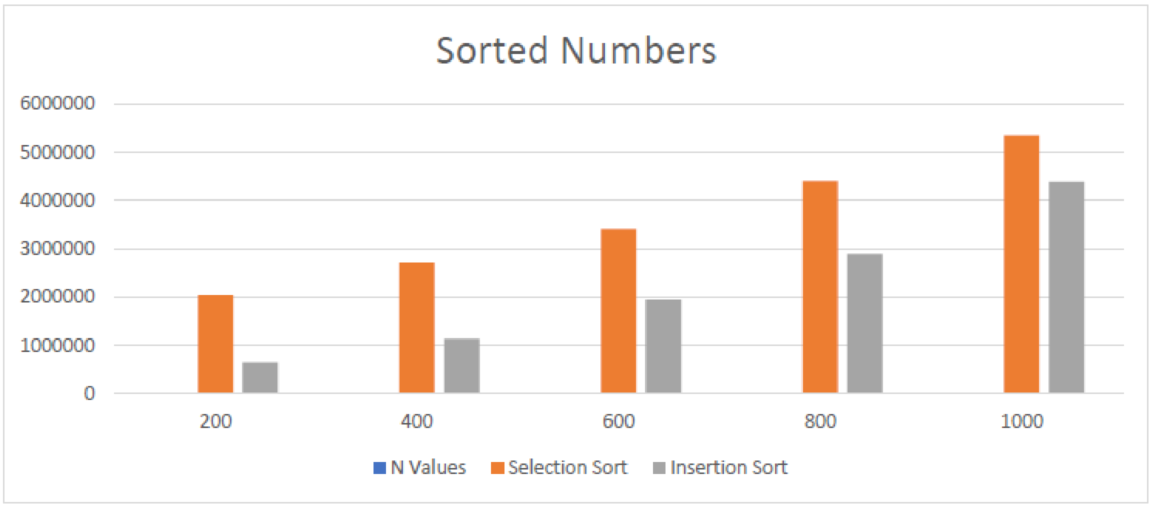
3. <u>Sorted Array:</u>

Time taken for each sort

Type of Array	Number of runs	Number of elements	Time taken by Selection Sort (ns)	Time taken by Insertion Sort(ns)
Random Numbers	***100***	***200***	2034276.38	646128.69
		400	2707910.22	1135231.67
		600	3402612.08	1939010.01
		800	4393544.42	2889563.12
		1000	5347155.93	4380407.59

Graph based on above values:





Sorted arrays are generated by looping an integer until "N" values and assigning array values from 0 to N, which turns out to be in sorted order.

From the above observations, we can see that,

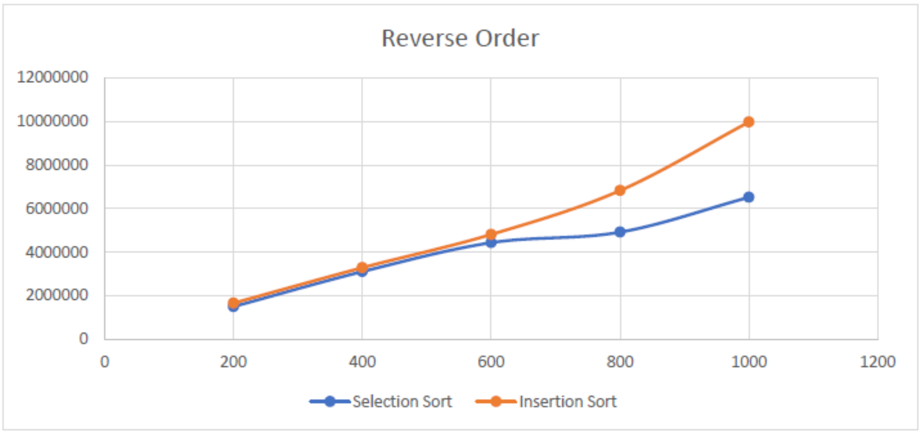
- Insertion sort will always take less time than Selection sort.
- Insertion sort is the best sorting algorithm, if the number of elements in the array is already or nearly sorted.
- Since the swap happens irrespective of checking previous and next element in selection sort, it takes more time than Insertion sort.

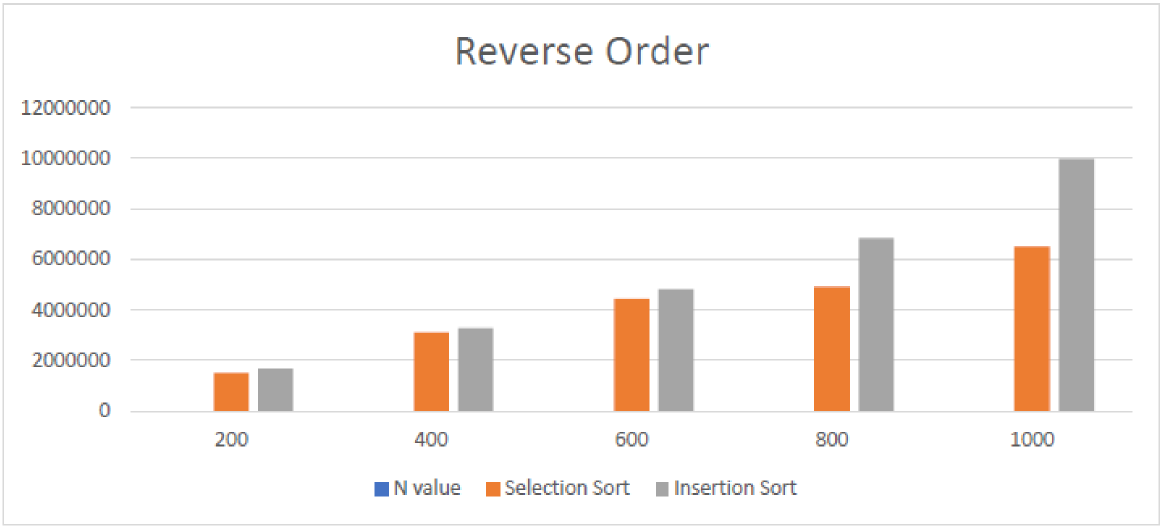
4. Reverse Sorted Array

Time taken for each sort

Type of Array	Number of runs	Number of elements	Time taken by Selection Sort (ns)	Time taken by Insertion Sort(ns)
Random Numbers	***100***	***200***	1498394.06	1658543.7
		400	3104226.44	3285440.18
		600	4431940.8	4808982.86
		800	4914791.57	6823599.24
		1000	6514704.18	9968258.21

Graph based on above values:





From the above observation, we can see that:

- Insertion sort and Selection sort takes almost same time to sort the values.
- As the number of elements increases, timing of insertion sort will increase proportionately the number of swaps increases.

Conclusion:

From this experiment, we can see that both Insertion sort and Selection sort has its own advantages and disadvantages. For partially and completely sorted array, Insertion sort is better than Selection sort as the number of swaps will be less. For a randomly sorted array, we cannot conclude on best possible sort as the time differs based on number of elements. For a reverse sorted array, Insertion sort and Selection sort takes almost same amount of time to execute the sort.