Erarica Mehra
NUID: 002113683
Assignment 2: benchmark

Tasks

Part 1) Timer and Benchmark

- Implemented method getClock and toMillisecs to generate number of ticks in nanoseconds and milliseconds respectively
- Implemented repeat method to return average number of milliseconds per repetition

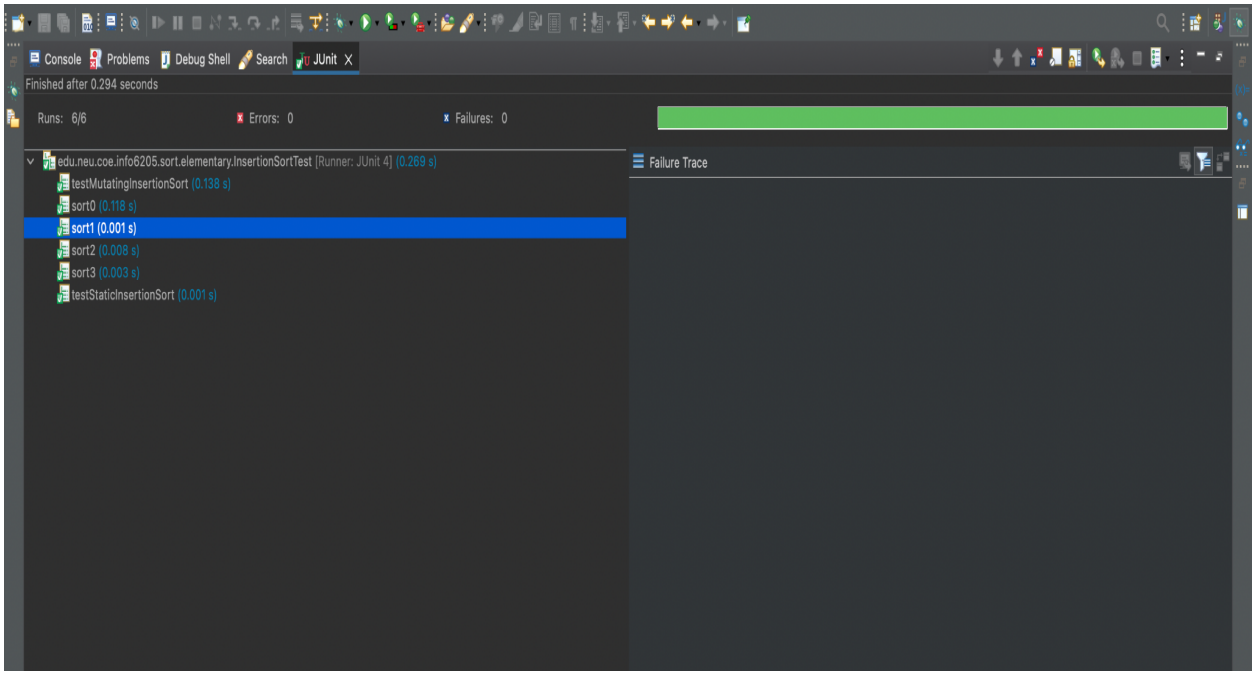
Part 2) Implemented Insertion Sort and executed Insertion Sort Tests

Part 3) Created a main method to measure the running time of three different types of arrays (randon, sorted, reverse sorted and partially sorted) and plotting the graph of running time versus size of array with the size increasing with every iteration.

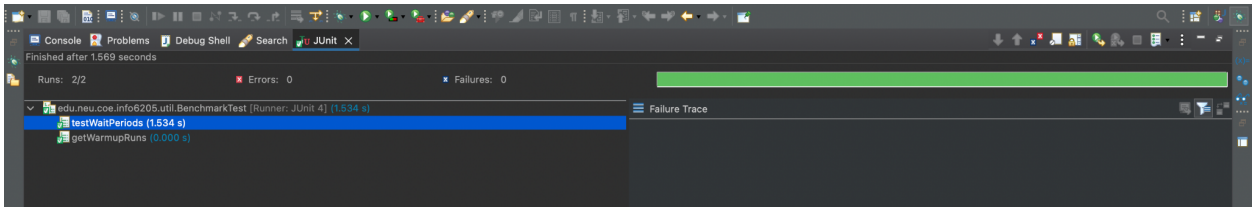
Conclusion:
The time complexity in the worst case is O(N2) and O(N) in the best case.

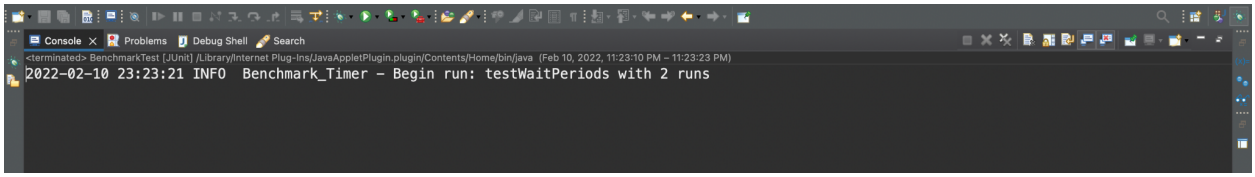
Evidence:

1. Unit Test Screenshot of InsertionSortTest: All test cases passed

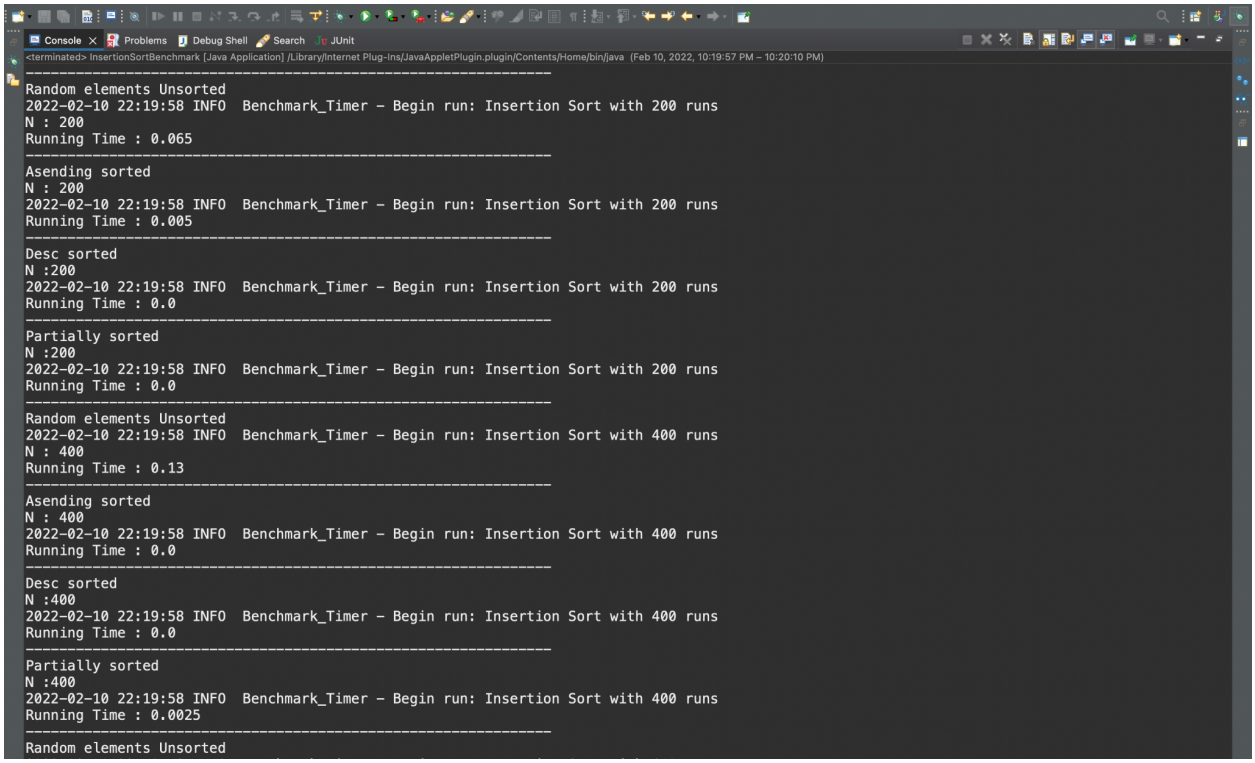


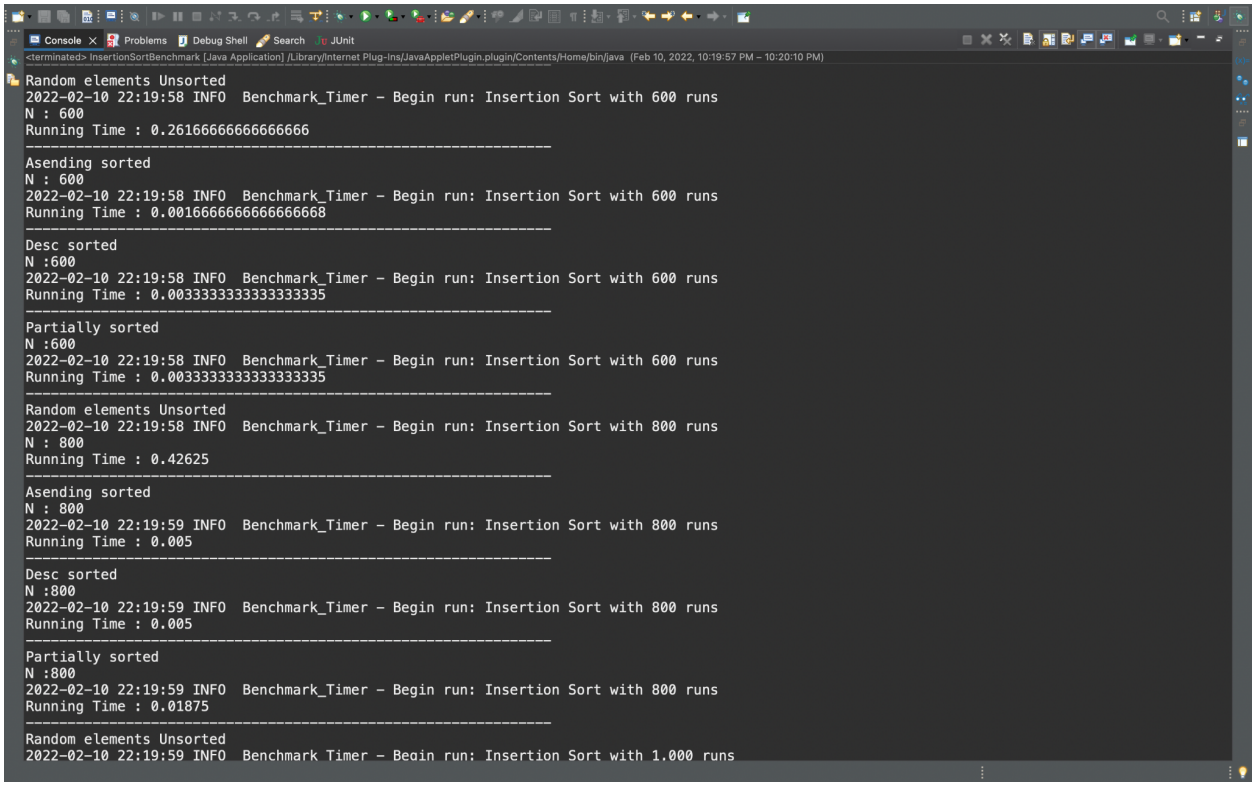
2. Unit Test Screenshot of BenchmarkTest: All test cases passed

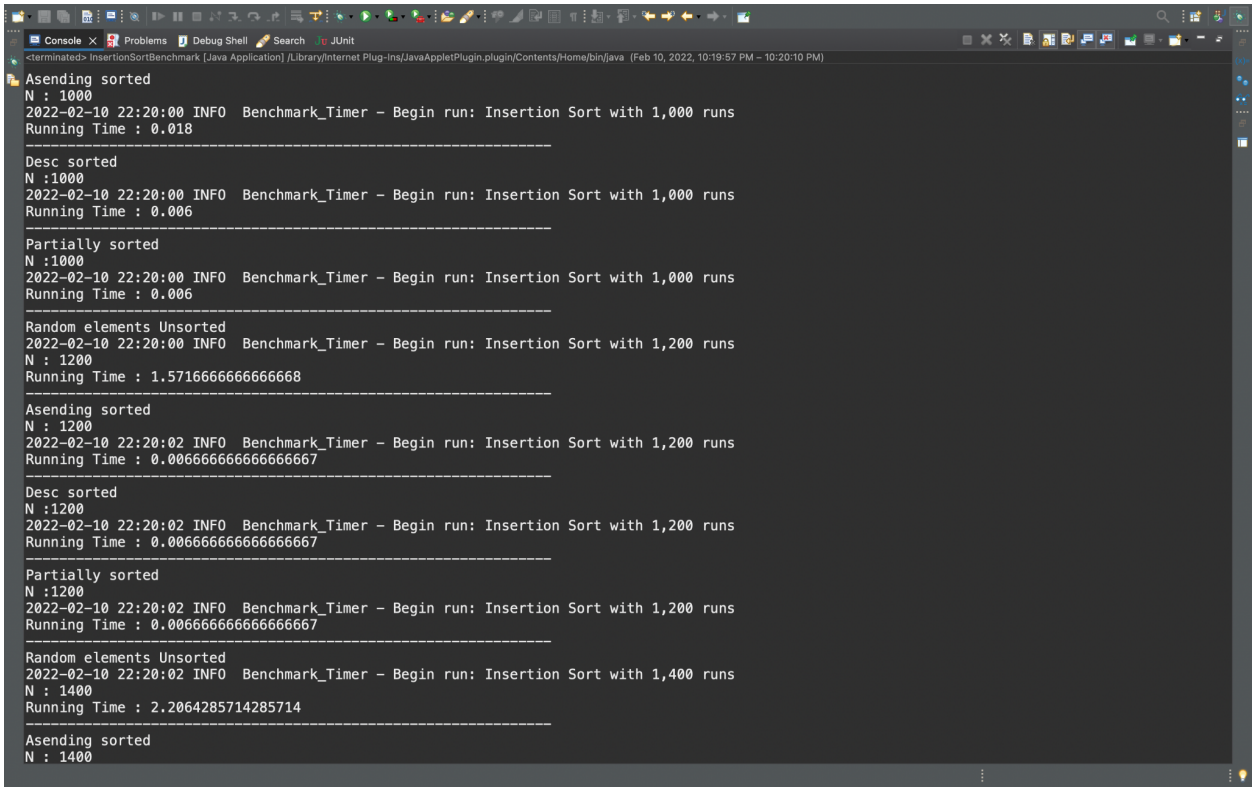


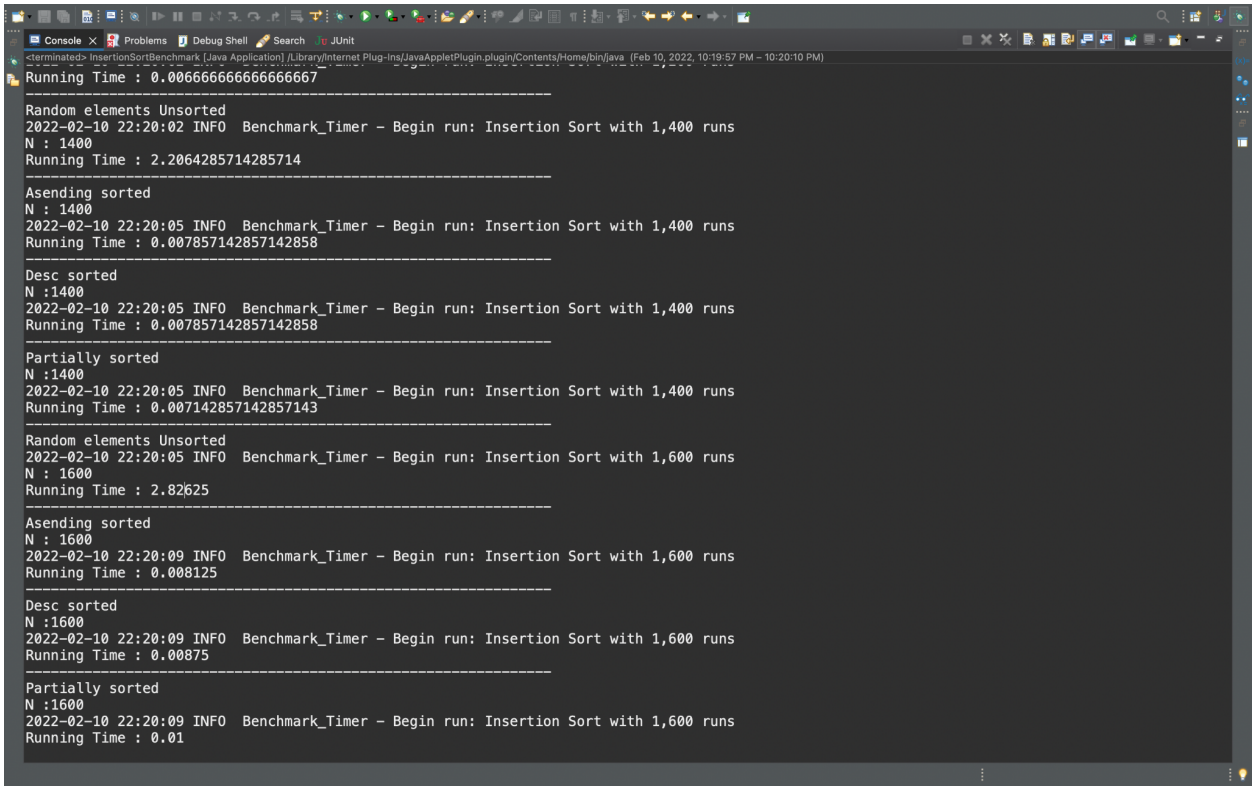


3. Snapshot of output:




Console
Problems
Debug Shell
Search
JUnit
<terminated> InsertionSortBenchmark [Java Application] /Library/Internet Plug-Ins/JavaAppletPlugin.plugin/Contents/Home/bin/java (Feb 10, 2022, 10:19:57 PM – 10:20:10 PM)
Random elements Unsorted
2022-02-10 22:19:58 INFO Benchmark_Timer – Begin run: Insertion Sort with 600 runs
N : 600
Running Time : 0.26166666666666666
Asending sorted
N : 600
2022-02-10 22:19:58 INFO Benchmark_Timer – Begin run: Insertion Sort with 600 runs
Running Time : 0.0016666666666666668
Desc sorted
N :600
2022-02-10 22:19:58 INFO Benchmark_Timer – Begin run: Insertion Sort with 600 runs
Running Time : 0.0033333333333333335
Partially sorted
N :600
2022-02-10 22:19:58 INFO Benchmark_Timer – Begin run: Insertion Sort with 600 runs
Running Time : 0.0033333333333333335
Random elements Unsorted
2022-02-10 22:19:58 INFO Benchmark_Timer – Begin run: Insertion Sort with 800 runs
N : 800
Running Time : 0.42625
Asending sorted
N : 800
2022-02-10 22:19:59 INFO Benchmark_Timer – Begin run: Insertion Sort with 800 runs
Running Time : 0.005
Desc sorted
N :800
2022-02-10 22:19:59 INFO Benchmark_Timer – Begin run: Insertion Sort with 800 runs
Running Time : 0.005
Partially sorted
N :800
2022-02-10 22:19:59 INFO Benchmark_Timer – Begin run: Insertion Sort with 800 runs
Running Time : 0.01875
Random elements Unsorted
2022-02-10 22:19:59 INFO Benchmark_Timer – Begin run: Insertion Sort with 1,000 runs


Console
Problems
Debug Shell
Search
JUnit
<terminated> InsertionSortBenchmark [Java Application] /Library/Internet Plug-Ins/JavaAppletPlugin.plugin/Contents/Home/bin/java (Feb 10, 2022, 10:19:57 PM – 10:20:10 PM)
Asending sorted
N : 1000
2022-02-10 22:20:00 INFO Benchmark_Timer – Begin run: Insertion Sort with 1,000 runs
Running Time : 0.018
Desc sorted
N :1000
2022-02-10 22:20:00 INFO Benchmark_Timer – Begin run: Insertion Sort with 1,000 runs
Running Time : 0.006
Partially sorted
N :1000
2022-02-10 22:20:00 INFO Benchmark_Timer – Begin run: Insertion Sort with 1,000 runs
Running Time : 0.006
Random elements Unsorted
2022-02-10 22:20:00 INFO Benchmark_Timer – Begin run: Insertion Sort with 1,200 runs
N : 1200
Running Time : 1.5716666666666668
Asending sorted
N : 1200
2022-02-10 22:20:02 INFO Benchmark_Timer – Begin run: Insertion Sort with 1,200 runs
Running Time : 0.006666666666666667
Desc sorted
N :1200
2022-02-10 22:20:02 INFO Benchmark_Timer – Begin run: Insertion Sort with 1,200 runs
Running Time : 0.006666666666666667
Partially sorted
N :1200
2022-02-10 22:20:02 INFO Benchmark_Timer – Begin run: Insertion Sort with 1,200 runs
Running Time : 0.006666666666666667
Random elements Unsorted
2022-02-10 22:20:02 INFO Benchmark_Timer – Begin run: Insertion Sort with 1,400 runs
N : 1400
Running Time : 2.2064285714285714
Asending sorted
N : 1400



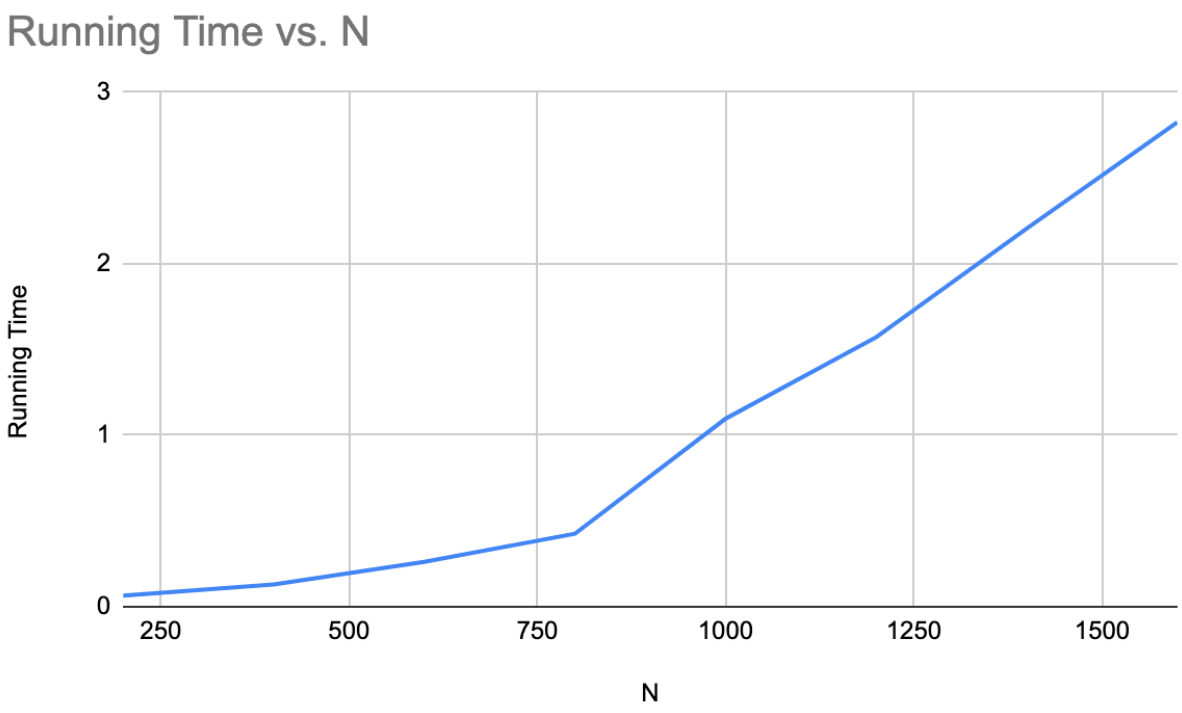
Observations:

As the size of the array increases, the running time to sort the array also increases.
Ordered array sorted in ascending order has the best time complexity of O(N).
Reverse ordered array has the worst time complexity of O(N2).
Hence we can conclude that the time complexity of insertion sort ranges from O(N2) to O(N).

1. Randomly Ordered Array

N	Running Time
200	0.065
400	0.13
600	0.261
800	0.426
1000	1.097
1200	1.571
1400	2.206
1600	2.826

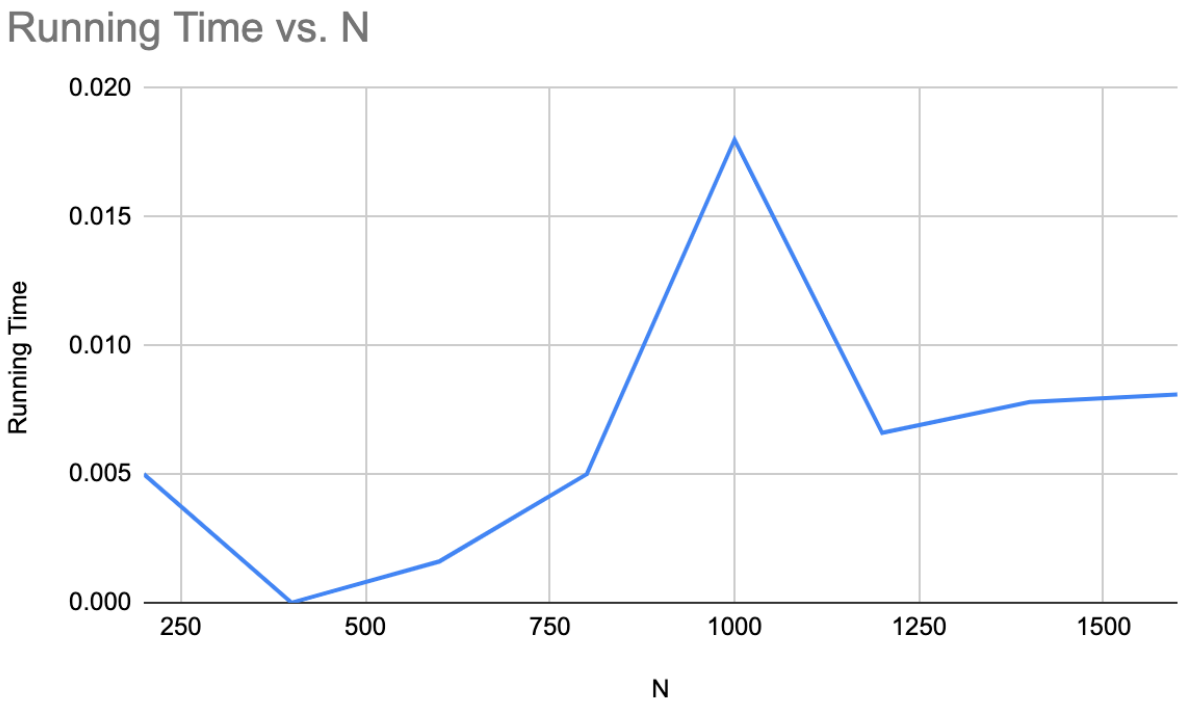
The following graph shows the order of growth of insertion sort for randomly sorted array



2. Ascending Sorted Array

N	Running Time
200	0.005
400	0
600	0.0016
800	0.005
1000	0.018
1200	0.0066
1400	0.0078
1600	0.0081

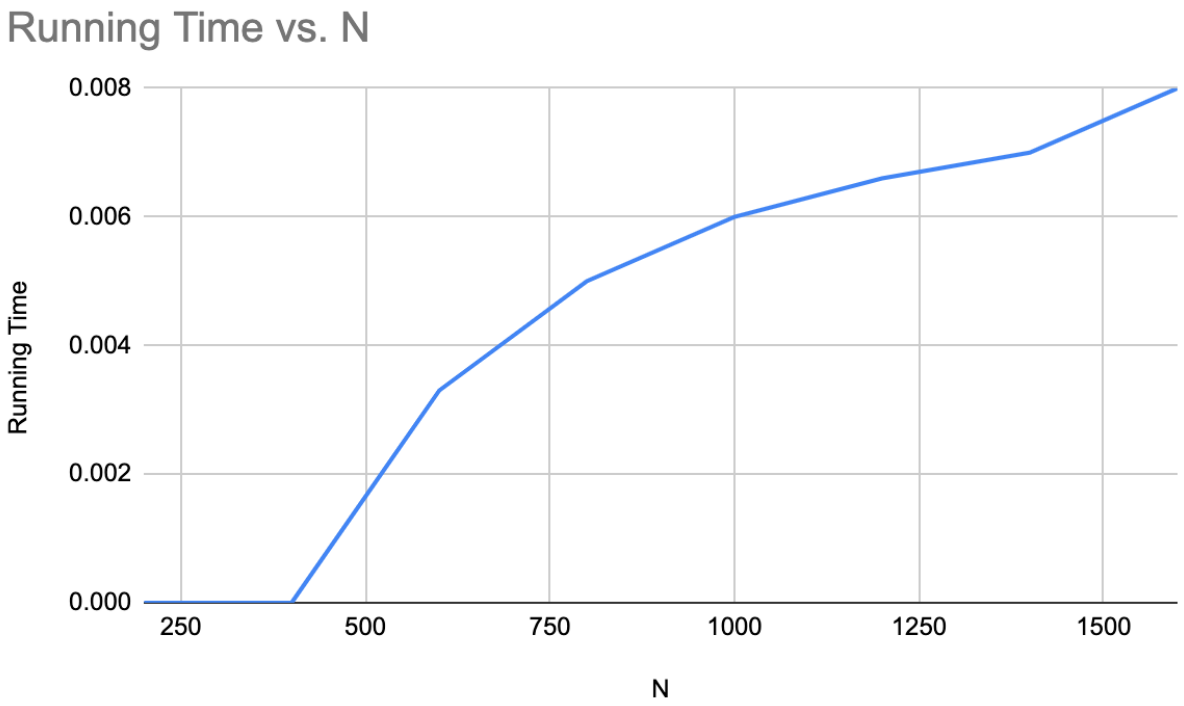
The following graph shows the order of growth of insertion sort for sorted array



3. Reversed Array

N	Running Time
200	0
400	0
600	0.0033
800	0.005
1000	0.006
1200	0.0066
1400	0.007
1600	0.008

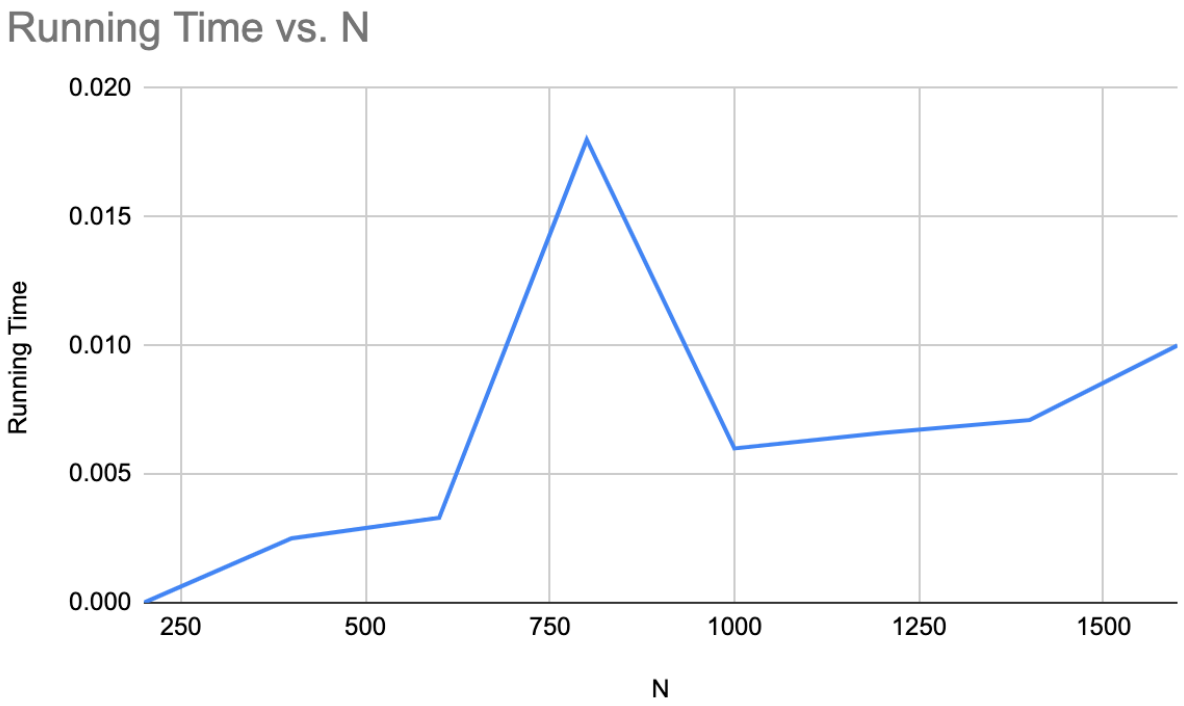
The following graph shows the order of growth of insertion sort for reversed sorted array



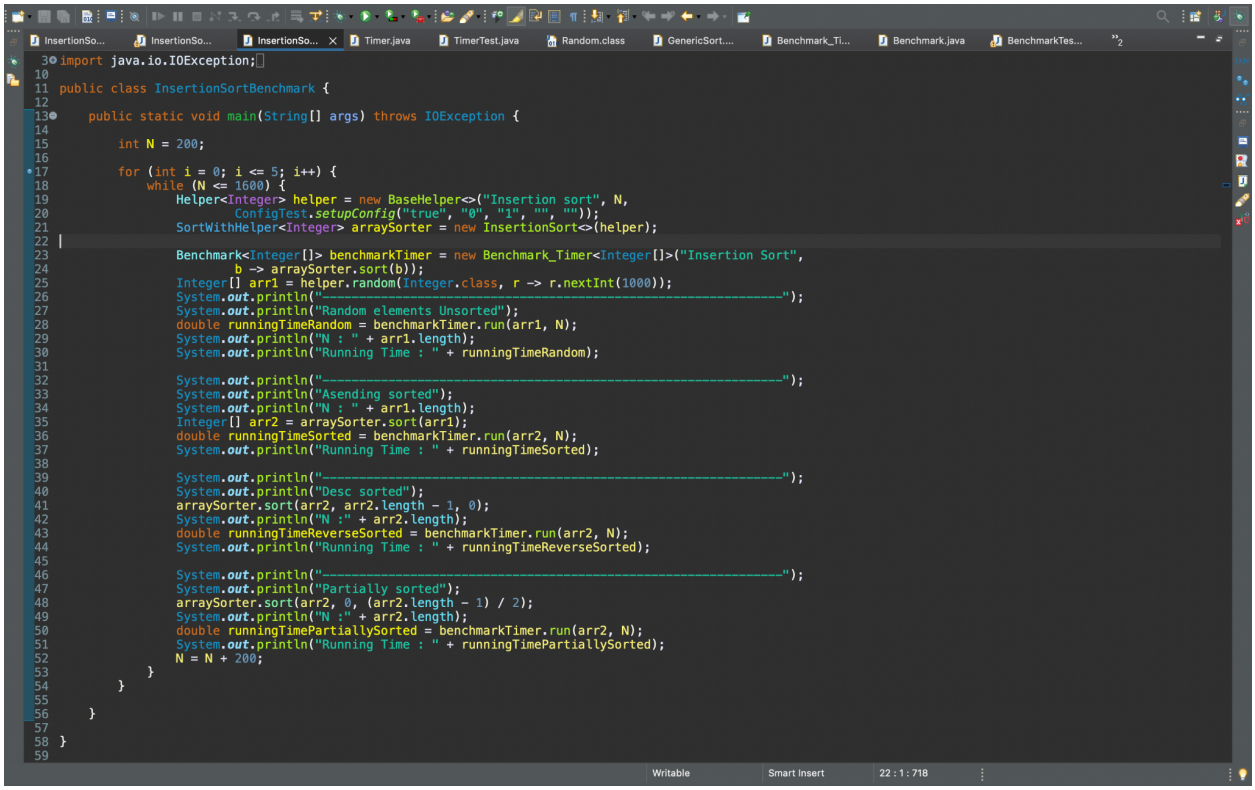
4. Partially Sorted Array

N	Running Time
200	0
400	0.0025
600	0.0033
800	0.018
1000	0.006
1200	0.0066
1400	0.0071
1600	0.01

The following graph shows the order of growth of insertion sort for partially sorted array


Running Time vs. N
0.020
0.015
0.010
0.005
0.000
Running Time
250
500
750
1000
1250
1500
N

Code :




```java
import java.io.IOException;

public class InsertionSortBenchmark {

    public static void main(String[] args) throws IOException {

        int N = 200;

        for (int i = 0; i <= 5; i++) {
            while (N <= 1600) {
                Helper<Integer> helper = new BaseHelper<>("Insertion sort", N,
                        ConfigTest.setupConfig("true", "0", "1", "", ""));
                SortWithHelper<Integer> arraySorter = new InsertionSort<>(helper);

                Benchmark<Integer[]> benchmarkTimer = new Benchmark_Timer<Integer[]>("Insertion Sort",
                        b -> arraySorter.sort(b));
                Integer[] arr1 = helper.random(Integer.class, r -> r.nextInt(1000));
                System.out.println("-----------------------------------------------------------------");
                System.out.println("Random elements Unsorted");
                double runningTimeRandom = benchmarkTimer.run(arr1, N);
                System.out.println("N : " + arr1.length);
                System.out.println("Running Time : " + runningTimeRandom);

                System.out.println("-----------------------------------------------------------------");
                System.out.println("Asending sorted");
                System.out.println("N : " + arr1.length);
                Integer[] arr2 = arraySorter.sort(arr1);
                double runningTimeSorted = benchmarkTimer.run(arr2, N);
                System.out.println("Running Time : " + runningTimeSorted);

                System.out.println("-----------------------------------------------------------------");
                System.out.println("Desc sorted");
                arraySorter.sort(arr2, arr2.length - 1, 0);
                System.out.println("N :" + arr2.length);
                double runningTimeReverseSorted = benchmarkTimer.run(arr2, N);
                System.out.println("Running Time : " + runningTimeReverseSorted);

                System.out.println("-----------------------------------------------------------------");
                System.out.println("Partially sorted");
                arraySorter.sort(arr2, 0, (arr2.length - 1) / 2);
                System.out.println("N :" + arr2.length);
                double runningTimePartiallySorted = benchmarkTimer.run(arr2, N);
                System.out.println("Running Time : " + runningTimePartiallySorted);
                N = N + 200;
            }
        }

    }
}

```