Rambukwella H.M.W.K.G

E/19/309

Variation of time against input in sorting algorithms

In here we are trying to analyze the time varying with number inputs. We have tabulated the results we got with number inputs and case scenarios.

We have chosen 3 types of data arrays,

- Worst data (array with descending data)
- Best data (array with sorted data)
- Random data

Results

Metric	Bubble Worst	Bubble Best	Bubble Random	Selection Worst	Selection Best	Selection Random	Insertion Worst	Insertion Best	Insertion Random
10	11200	600	4200	8600	500	3200	5300	500	300
50	135700	2000	113000	29400	1100	57500	1900	1100	1000
250	1887300	2800	1914500	1496600	3100	1056900	500	2000	1700
1250	9545900	7300	2951000	5735400	37200	2661700	1600	800	900
6250	17362600	26000	54577300	14278200	6600	30319600	5500	4600	3500
1250	371176000	18300	354856100	69387300	17800	29458200	21200	19100	16500

In bubble sort its O(n^2), So with input count the time must be quadratically increasing.

So when n=10, t= 11200 and for n = 50 as theoretically t=280000
But as the results its not got that much, this is because the first time has interfere with another computer process.

In selection sort its also O(n^2), So with input count the time must be quadratically increasing.

When n=50, t=29400 and for n= 250 as theoretically t = 735000
But as the results its not got that much, this is because this has interfere with another computer process.

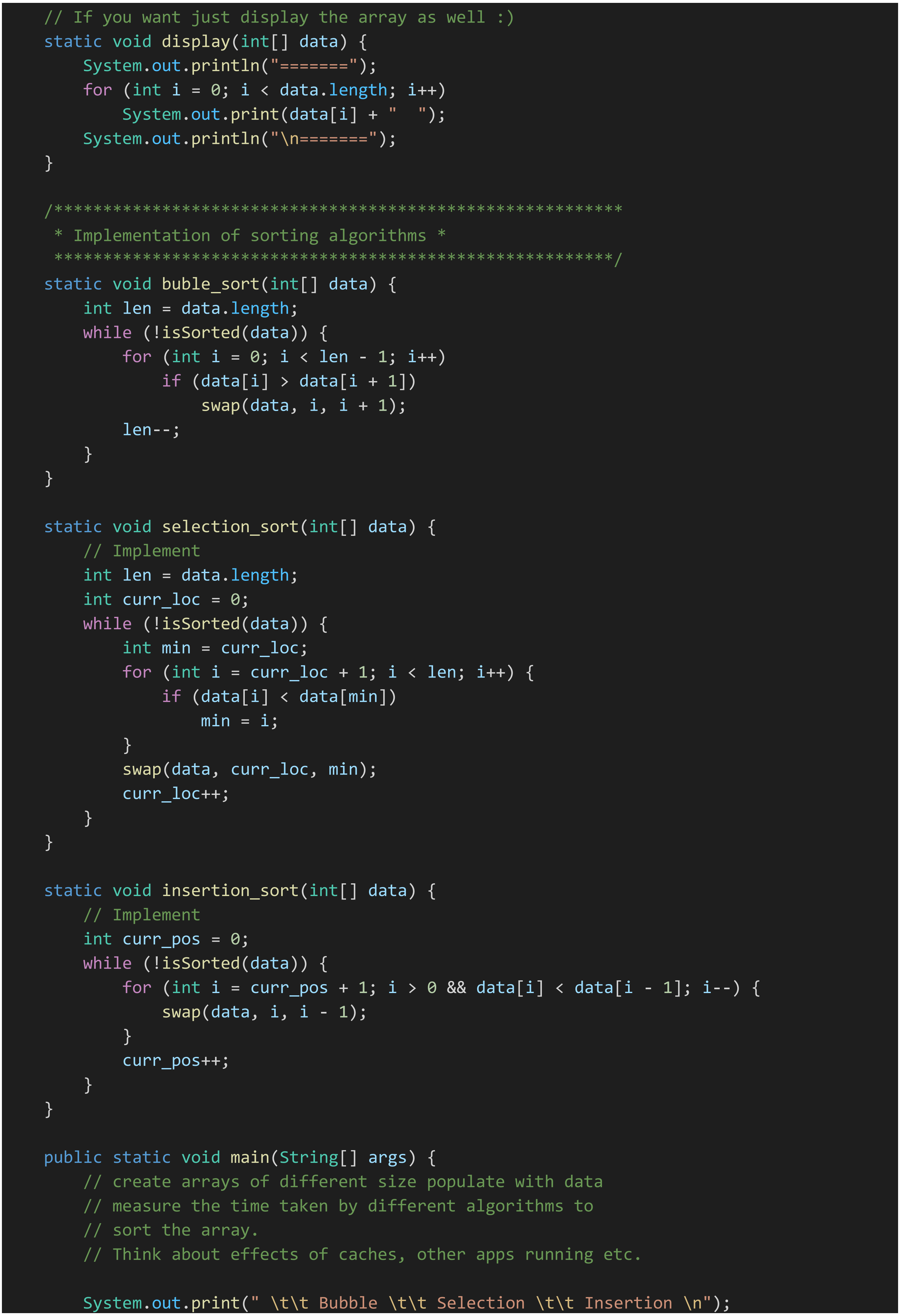
In Insertion sort its also O(n^2), So with input count the time must be quadratically increasing.

When n=50, t=1900 and for n= 250 as theoretically t = 47500
But as the results its not got that much, this is because this has interfere with another computer process.

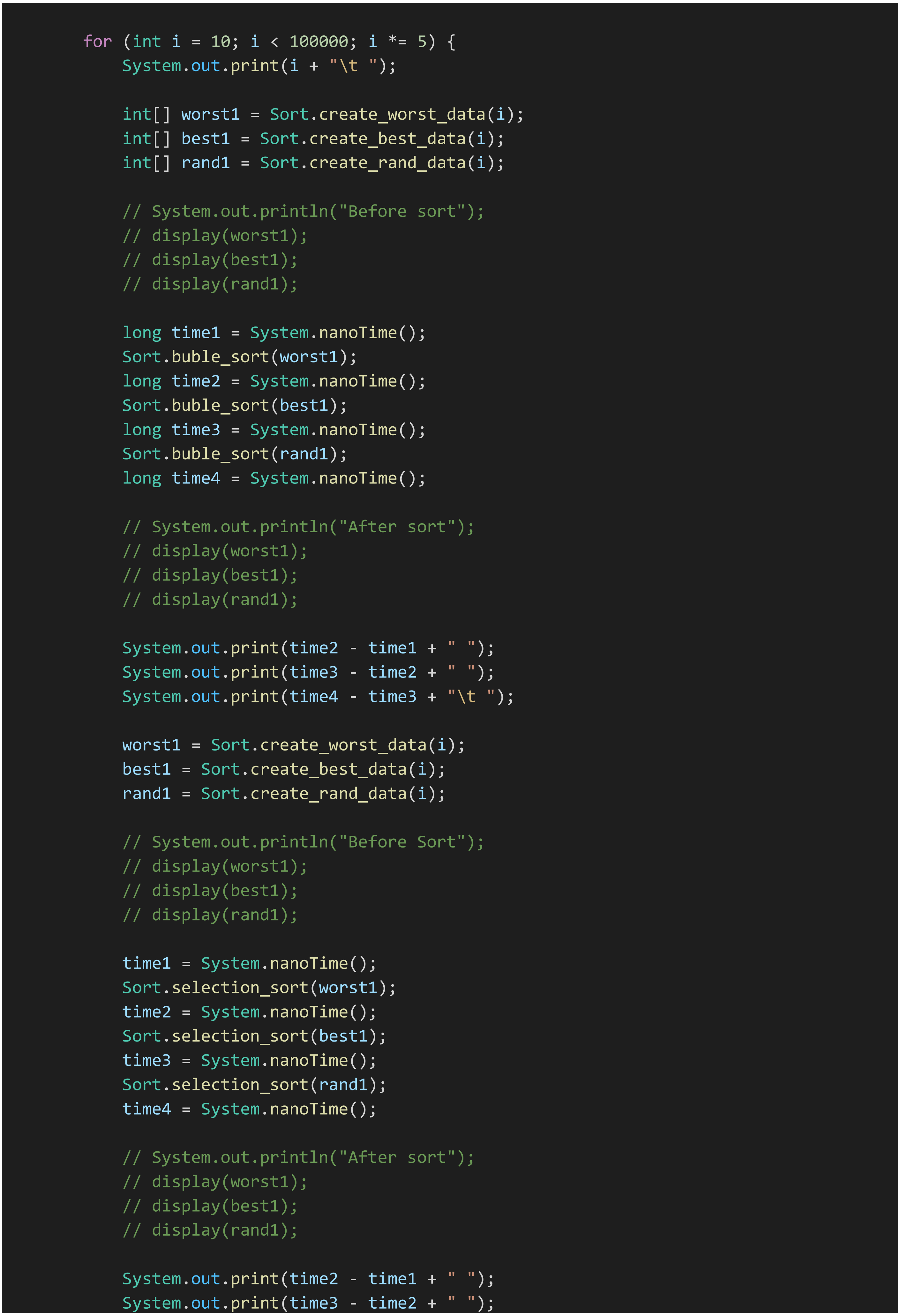
As you see here we have tested on normal windows computer that run different processes in background so it affects the result. If we want to check better timing we have to use virtual environment (isolated environment) so the timing reading will near perfect.

Code

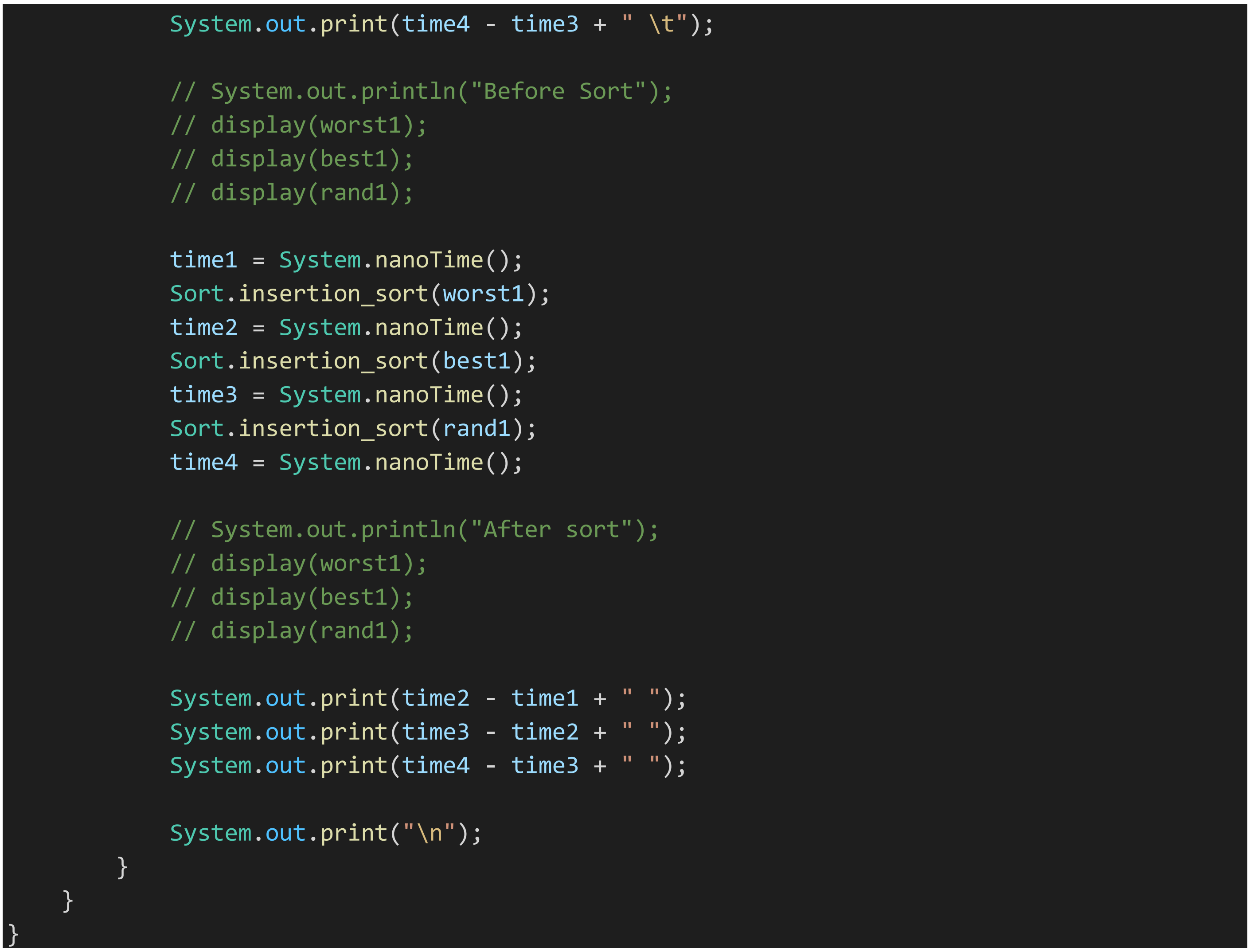
```java
/**
 * Simple sorting algorithms and their performance
 * Reg:
 *
 */

public class Sort {

    // create an array of given size and populate it with random data
    static int[] create_rand_data(int size_of_array) {
        int[] data = new int[size_of_array];
        int i;
        for (i = 0; i < data.length; i++)
            data[i] = (int) (Math.random() * 100);
        return data;
    }

    // create an array of given size and populate it with worst data arrangement
    static int[] create_worst_data(int size_of_array) {
        int[] data = new int[size_of_array];
        int i;
        for (i = 0; i < data.length; i++)
            data[i] = data.length - i;
        return data;
    }

    // create an array of given size and populate it with best data arrangement
    static int[] create_best_data(int size_of_array) {
        int[] data = new int[size_of_array];
        int i;
        for (i = 0; i < data.length; i++)
            data[i] = i;
        return data;
    }

    // function to swap. Would be useful since all need this
    static void swap(int[] d, int i, int j) {
        int tmp = d[i];
        d[i] = d[j];
        d[j] = tmp;
    }

    // check if the soring worked on the array
    static boolean isSorted(int[] data) {
        int i;
        for (i = 1; i < data.length; i++)
            if (data[i] < data[i - 1])
                break;
        return (i == data.length);
    }
```



```
    // If you want just display the array as well :)
    static void display(int[] data) {
        System.out.println("=======");
        for (int i = 0; i < data.length; i++)
            System.out.print(data[i] + "  ");
        System.out.println("\n=======");
    }

    /***********************************************************
     * Implementation of sorting algorithms *
     **********************************************************/
    static void buble_sort(int[] data) {
        int len = data.length;
        while (!isSorted(data)) {
            for (int i = 0; i < len - 1; i++)
                if (data[i] > data[i + 1])
                    swap(data, i, i + 1);
            len--;
        }
    }

    static void selection_sort(int[] data) {
        // Implement
        int len = data.length;
        int curr_loc = 0;
        while (!isSorted(data)) {
            int min = curr_loc;
            for (int i = curr_loc + 1; i < len; i++) {
                if (data[i] < data[min])
                    min = i;
            }
            swap(data, curr_loc, min);
            curr_loc++;
        }
    }

    static void insertion_sort(int[] data) {
        // Implement
        int curr_pos = 0;
        while (!isSorted(data)) {
            for (int i = curr_pos + 1; i > 0 && data[i] < data[i - 1]; i--) {
                swap(data, i, i - 1);
            }
            curr_pos++;
        }
    }

    public static void main(String[] args) {
        // create arrays of different size populate with data
        // measure the time taken by different algorithms to
        // sort the array.
        // Think about effects of caches, other apps running etc.

        System.out.print(" \t\t Bubble \t\t Selection \t\t Insertion \n");
```



```java
        for (int i = 10; i < 100000; i *= 5) {
            System.out.print(i + "\t ");

            int[] worst1 = Sort.create_worst_data(i);
            int[] best1 = Sort.create_best_data(i);
            int[] rand1 = Sort.create_rand_data(i);

            // System.out.println("Before sort");
            // display(worst1);
            // display(best1);
            // display(rand1);

            long time1 = System.nanoTime();
            Sort.buble_sort(worst1);
            long time2 = System.nanoTime();
            Sort.buble_sort(best1);
            long time3 = System.nanoTime();
            Sort.buble_sort(rand1);
            long time4 = System.nanoTime();

            // System.out.println("After sort");
            // display(worst1);
            // display(best1);
            // display(rand1);

            System.out.print(time2 - time1 + " ");
            System.out.print(time3 - time2 + " ");
            System.out.print(time4 - time3 + "\t ");

            worst1 = Sort.create_worst_data(i);
            best1 = Sort.create_best_data(i);
            rand1 = Sort.create_rand_data(i);

            // System.out.println("Before Sort");
            // display(worst1);
            // display(best1);
            // display(rand1);

            time1 = System.nanoTime();
            Sort.selection_sort(worst1);
            time2 = System.nanoTime();
            Sort.selection_sort(best1);
            time3 = System.nanoTime();
            Sort.selection_sort(rand1);
            time4 = System.nanoTime();

            // System.out.println("After sort");
            // display(worst1);
            // display(best1);
            // display(rand1);

            System.out.print(time2 - time1 + " ");
            System.out.print(time3 - time2 + " ");
```



```java
            System.out.print(time4 - time3 + " \t");

            // System.out.println("Before Sort");
            // display(worst1);
            // display(best1);
            // display(rand1);

            time1 = System.nanoTime();
            Sort.insertion_sort(worst1);
            time2 = System.nanoTime();
            Sort.insertion_sort(best1);
            time3 = System.nanoTime();
            Sort.insertion_sort(rand1);
            time4 = System.nanoTime();

            // System.out.println("After sort");
            // display(worst1);
            // display(best1);
            // display(rand1);

            System.out.print(time2 - time1 + " ");
            System.out.print(time3 - time2 + " ");
            System.out.print(time4 - time3 + " ");

            System.out.print("\n");
        }
    }
}
```