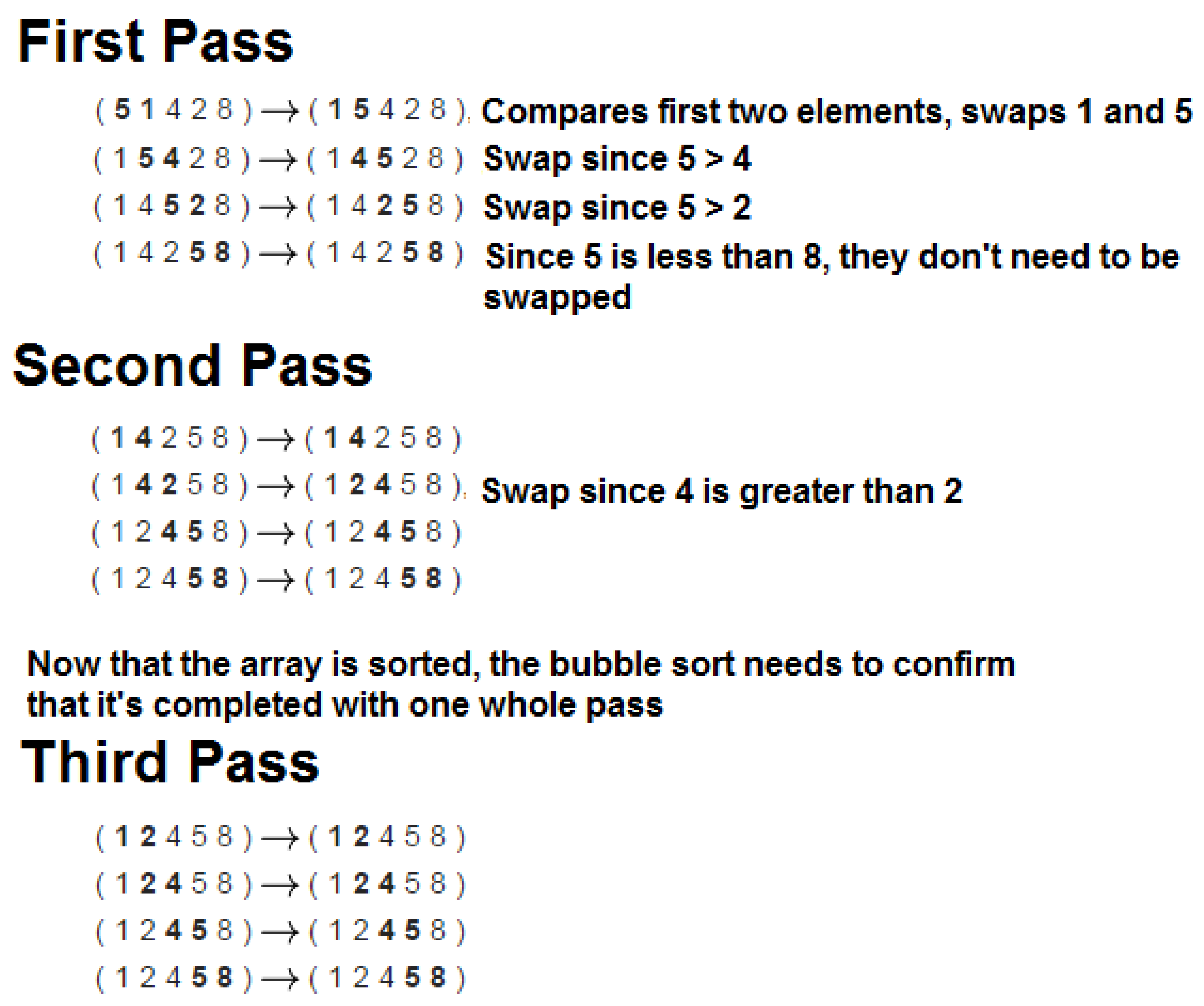
SORTING TYPES SPEED TEST

BUBBLE SORT VS. SELECTION SORT VS. INSERTION SORT

Rakib Hassan
Edwin Quintuna
Wanyi Chen

BUBBLE SORT ALGORITHM



First Pass

(**5 1** 4 2 8) → (**1 5** 4 2 8), Compares first two elements, swaps 1 and 5
(1 **5 4** 2 8) → (1 **4 5** 2 8) Swap since 5 > 4
(1 4 **5 2** 8) → (1 4 **2 5** 8) Swap since 5 > 2
(1 4 2 **5 8**) → (1 4 2 **5 8**) Since 5 is less than 8, they don't need to be swapped

Second Pass

(**1 4** 2 5 8) → (**1 4** 2 5 8)
(1 **4 2** 5 8) → (1 **2 4** 5 8), Swap since 4 is greater than 2
(1 2 **4 5** 8) → (1 2 **4 5** 8)
(1 2 4 **5 8**) → (1 2 4 **5 8**)

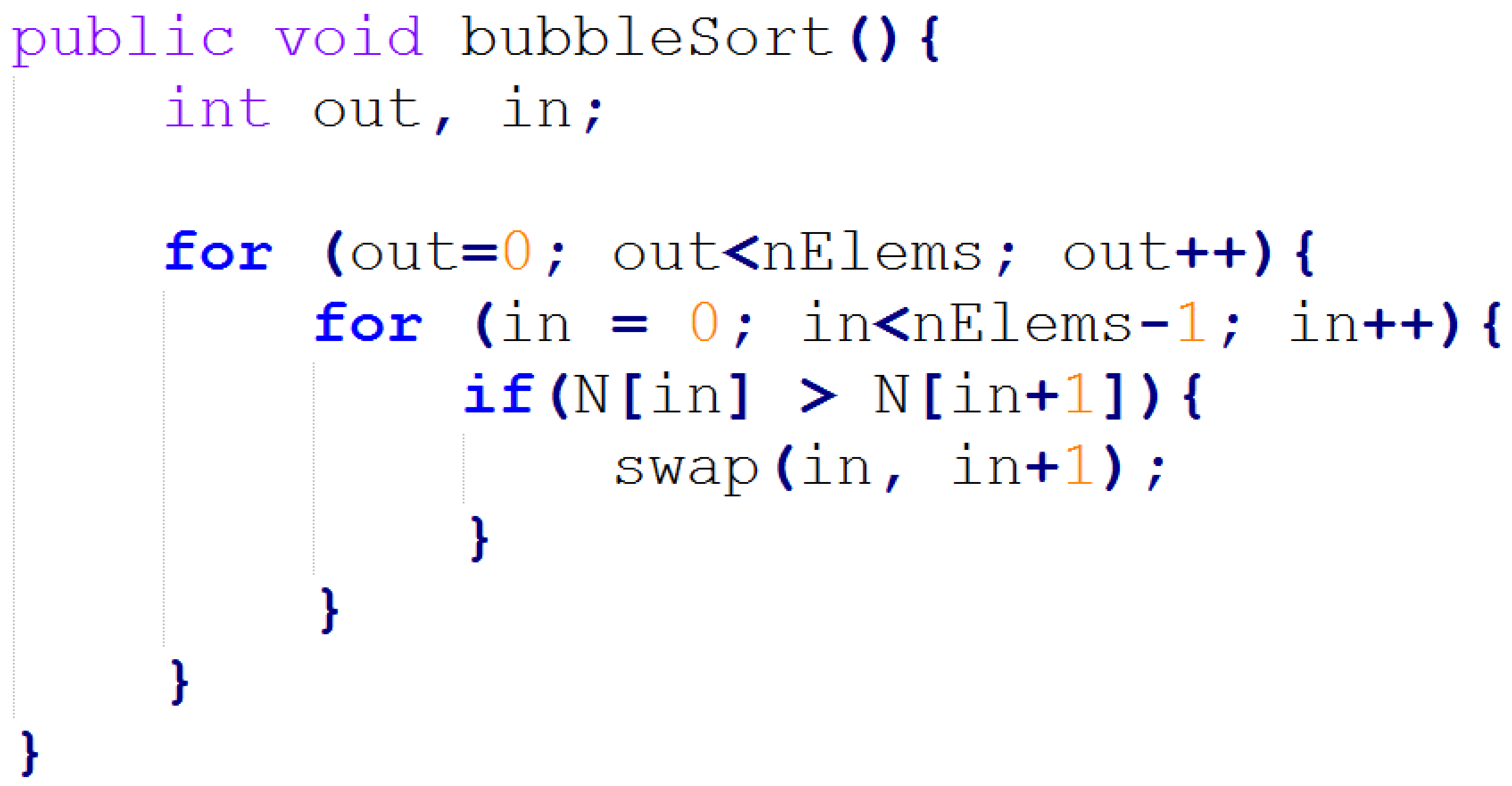
Now that the array is sorted, the bubble sort needs to confirm that it's completed with one whole pass

Third Pass

(**1 2** 4 5 8) → (**1 2** 4 5 8)
(1 **2 4** 5 8) → (1 **2 4** 5 8)
(1 2 **4 5** 8) → (1 2 **4 5** 8)
(1 2 4 **5 8**) → (1 2 4 **5 8**)

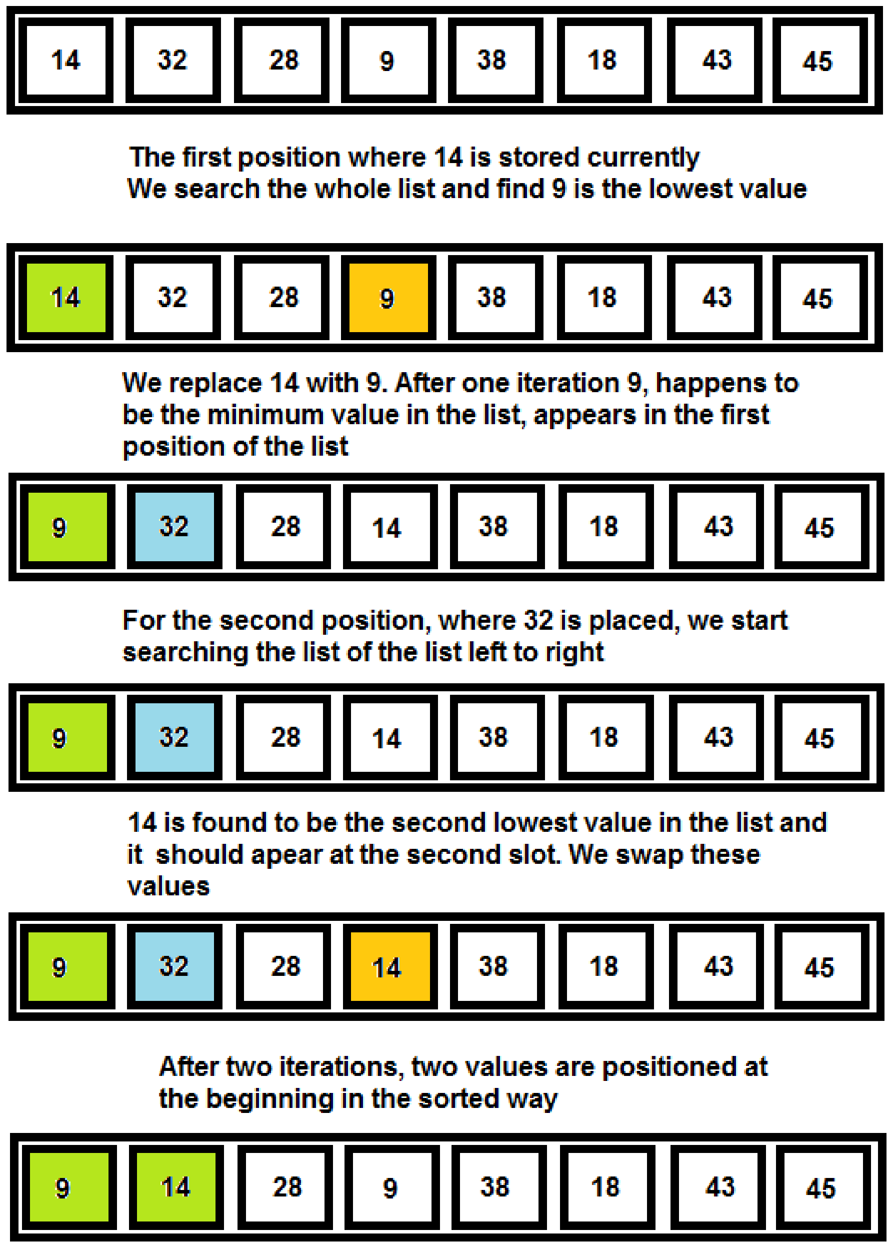
Source: http://cppbetterexplained.com/the-bubble-sort-algorithm/

BUBBLE SORT CODE



```
public void bubbleSort(){
    int out, in;

    for (out=0; out<nElems; out++){
        for (in = 0; in<nElems-1; in++){
            if(N[in] > N[in+1]){
                swap(in, in+1);
            }
        }
    }
}
```

SELECTION SORT ALGORITHM



The first position where 14 is stored currently
We search the whole list and find 9 is the lowest value

We replace 14 with 9. After one iteration 9, happens to be the minimum value in the list, appears in the first position of the list

For the second position, where 32 is placed, we start searching the list of the list left to right

14 is found to be the second lowest value in the list and it should apear at the second slot. We swap these values

After two iterations, two values are positioned at the beginning in the sorted way

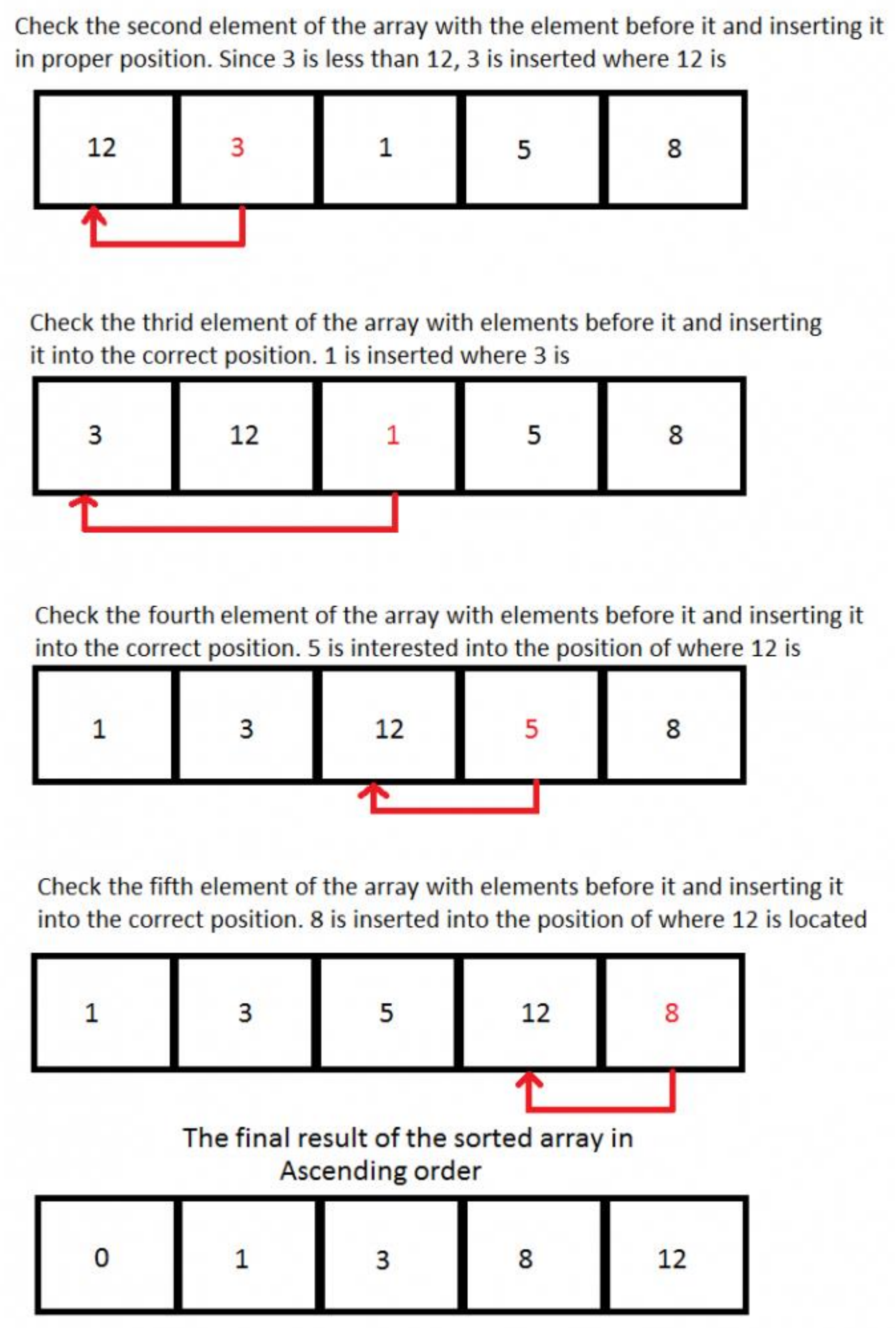
The same process is applied on the rest of the items in the list

Source: http://cppbetterexplained.com/the-selection-sort-algorithm/

SELECTION SORT CODE

```
public void selectionSort(){
    int out, in, min;

    for (out = 0; out<nElems-1; out++){
        min = out;
        for (in = out+1; in<nElems; in++){
            if(N[in] < N[min]){
                min = in;
            }
        swap(out, min);
        }
    }
}
```

INSERTION SORT ALGORITHM


Check the second element of the array with the element before it and inserting it in proper position. Since 3 is less than 12, 3 is inserted where 12 is

Check the thrid element of the array with elements before it and inserting it into the correct position. 1 is inserted where 3 is

Check the fourth element of the array with elements before it and inserting it into the correct position. 5 is interested into the position of where 12 is

Check the fifth element of the array with elements before it and inserting it into the correct position. 8 is inserted into the position of where 12 is located

The final result of the sorted array in Ascending order

Source: http://cppbetterexplained.com/the-insertion-sort-algorithm/

INSERTION SORT CODE

```
public void insertionSort(){
    int out, in;

    for (out=1; out<nElems; out++){
        int temp = N[out];
        in = out;
        while (in>0 && N[in-1] >= temp){
            N[in] = N[in-1];
            --in;
        }
        N[in] = temp;
    }
}
```

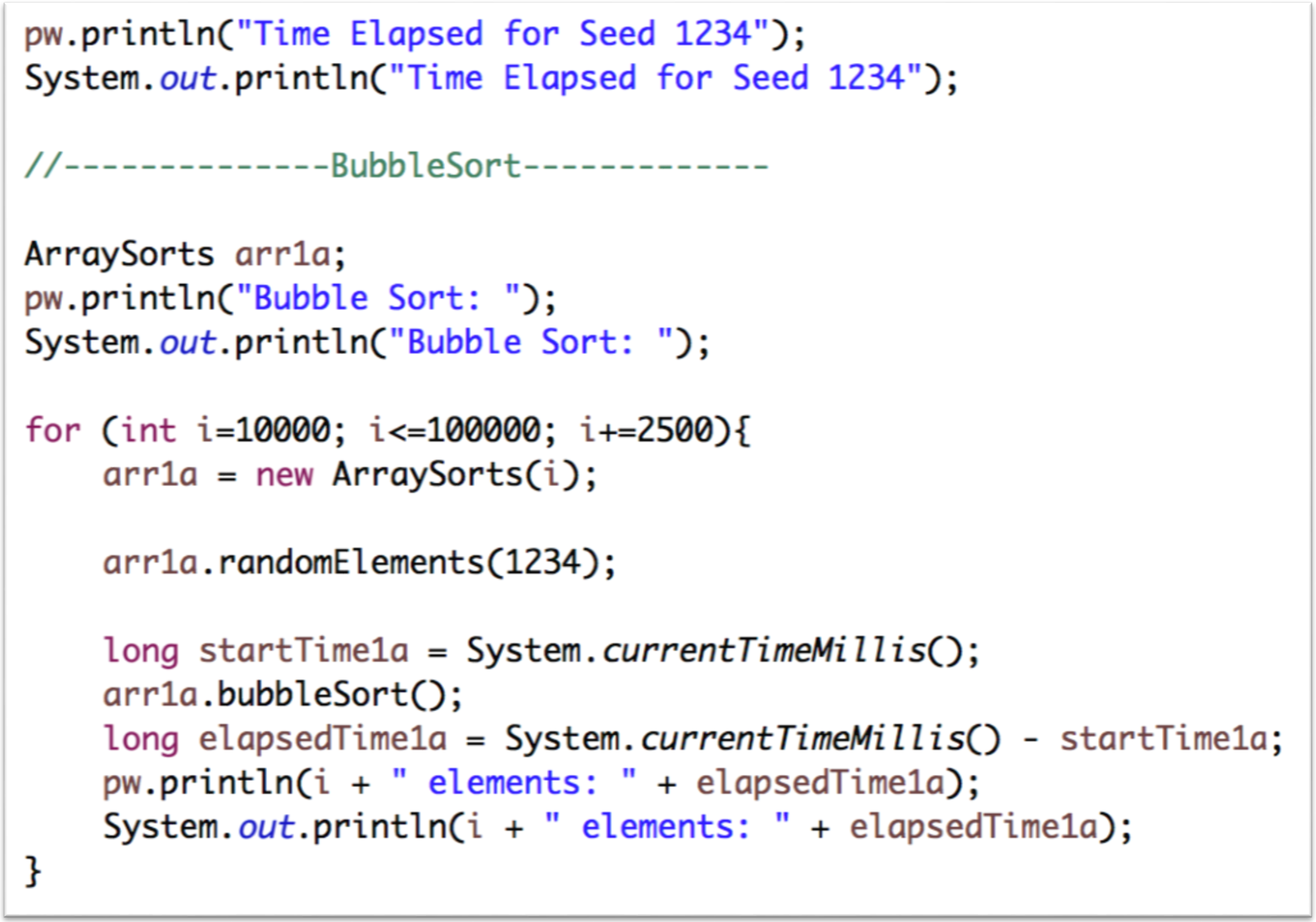
BIG O

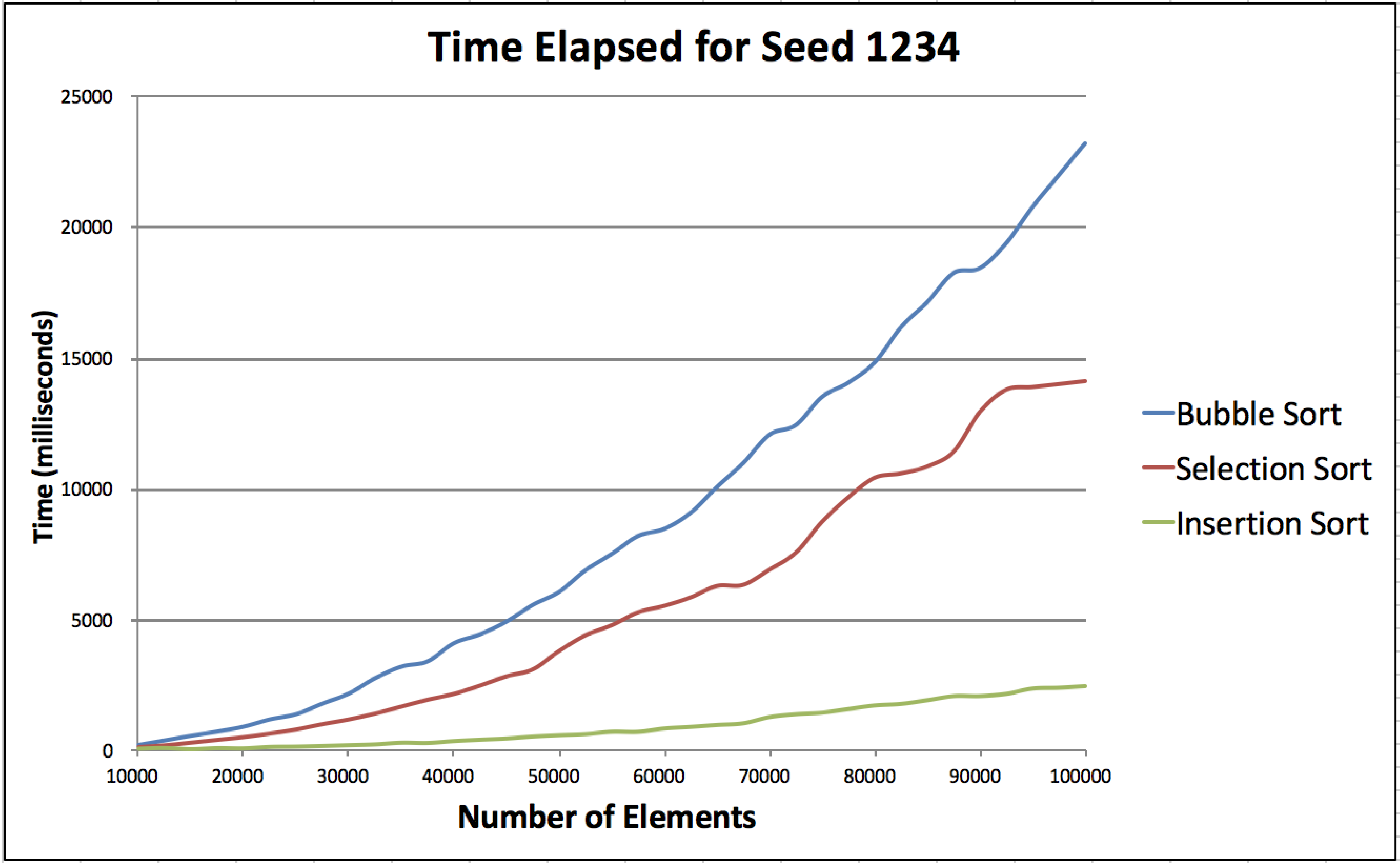
Used to describe performance or complexity of an algorithm – this project focuses on time complexity

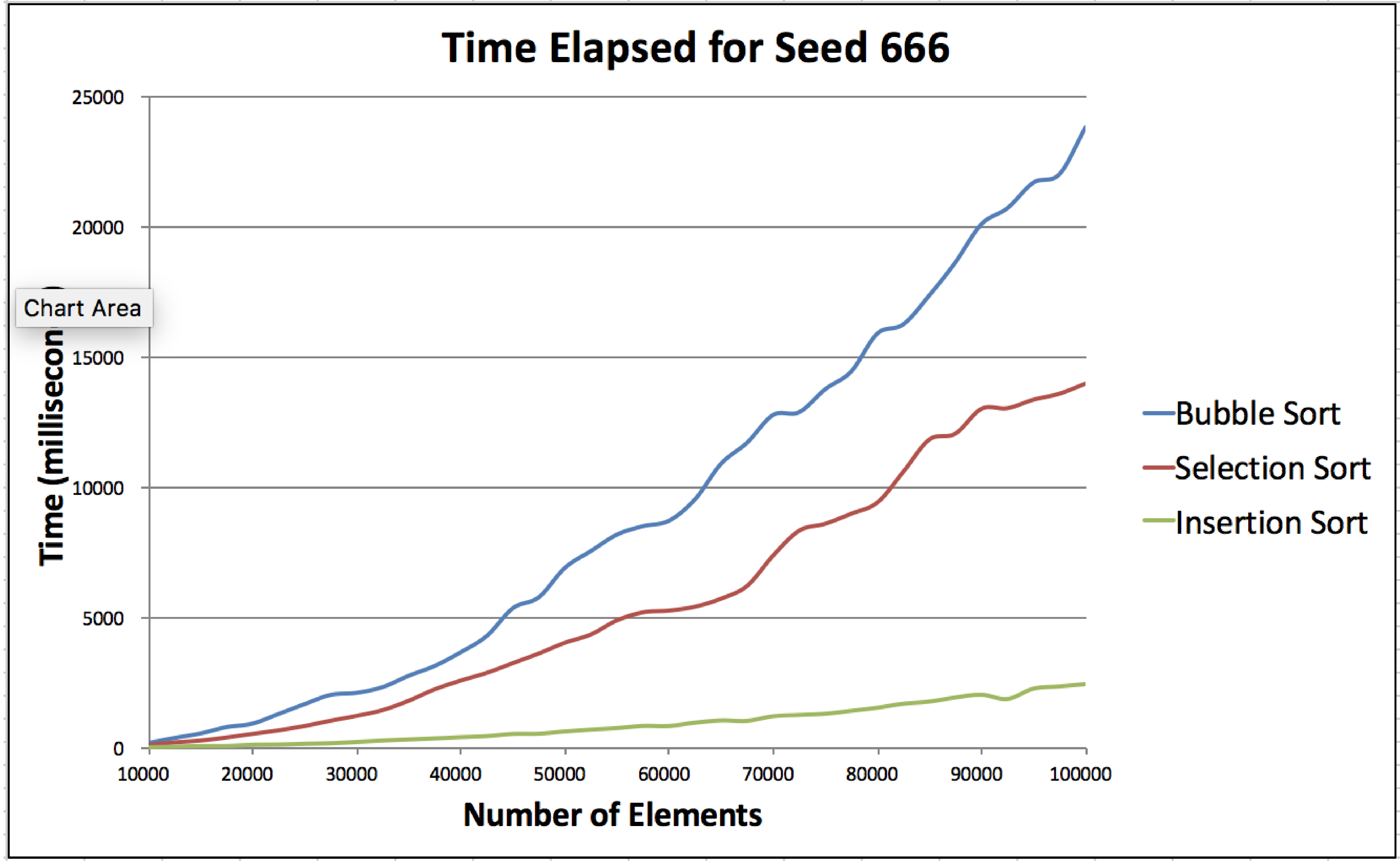
Algorithm	Best	Average	Worst
Bubble Sort	O(n)	O(n^2)	O(n^2)
Selection Sort	O(n^2)	O(n^2)	O(n^2)
Insertion Sort	O(n)	O(n^2)	O(n^2)

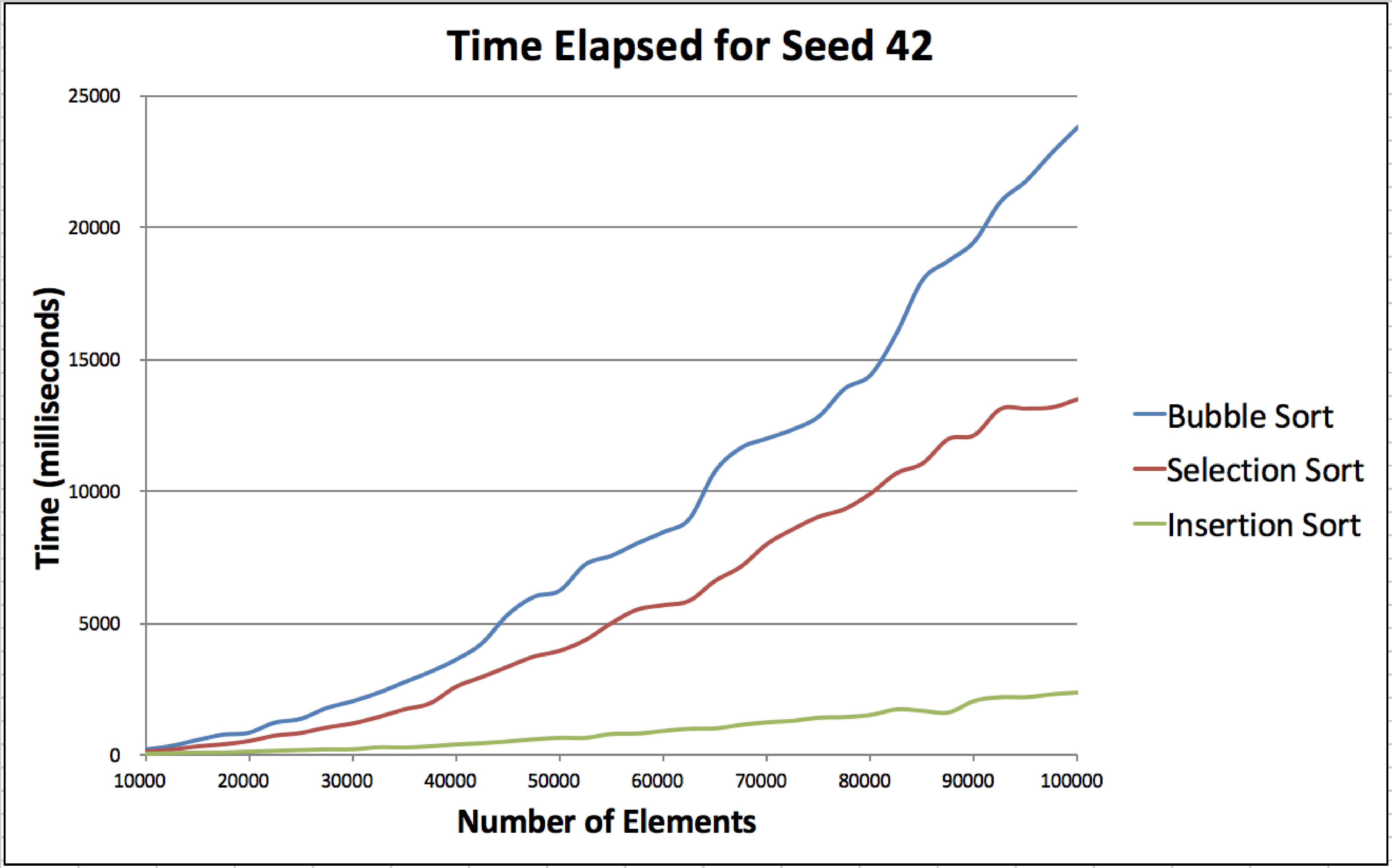
HOW ARE WE RUNNING THE TEST?

1. Sort array of size 10,000 using bubble sort and record elapsed time
2. Increment element number by 2,500 until 100,000 elements is reached and record elapsed time value each time
3. Repeat steps 1-2, but with selection sort and insertion sort
4. Repeat steps 1-3 with different seed number to ensure validity of data


```
pw.println("Time Elapsed for Seed 1234");
System.out.println("Time Elapsed for Seed 1234");

//-------------BubbleSort-------------

ArraySorts arr1a;
pw.println("Bubble Sort: ");
System.out.println("Bubble Sort: ");

for (int i=10000; i<=100000; i+=2500){
    arr1a = new ArraySorts(i);

    arr1a.randomElements(1234);

    long startTime1a = System.currentTimeMillis();
    arr1a.bubbleSort();
    long elapsedTime1a = System.currentTimeMillis() - startTime1a;
    pw.println(i + " elements: " + elapsedTime1a);
    System.out.println(i + " elements: " + elapsedTime1a);
}
```


Time Elapsed for Seed 1234
Time (milliseconds)
25000
20000
15000
10000
5000
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
Number of Elements
Bubble Sort
Selection Sort
Insertion Sort


Time Elapsed for Seed 666
Time (millisecon
Chart Area
25000
20000
15000
10000
5000
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
Number of Elements
Bubble Sort
Selection Sort
Insertion Sort


Time Elapsed for Seed 42
Time (milliseconds)
25000
20000
15000
10000
5000
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
Number of Elements
Bubble Sort
Selection Sort
Insertion Sort

CONCLUSIONS

- 1st Place: **INSERTION SORT –**

  significantly faster than selection and insertion sort
- 2nd Place: **SELECTION SORT –**

  slightly faster than bubble sort, but much
  slower than insertion sort
- 3rd Place: **BUBBLE SORT –**

  slowest and least efficient sorting type

IMPORTANT THINGS TO NOTE

- Slight variance in value of elapsed time each time program is run
- Slight computer hiccups may cause some strange results such as slightly higher element number accompanied by slightly lower time required to sort