# Lab assignment solution

## Problem 1:

A. No. it might work in certain cases, but there is no way of being sure it will the expected result, all we can do is calculate the probability it will work after a certain run.

B. The best-case would be the array is already sorted.

C. The best-case has an asymptotic runtime of $O(1)$.

D. The worst-case runtime is that the randomizing algorithm does not sort the array.

E. No, its not because the comparison operations are not related to the arrangement operations.

## Problem 2:

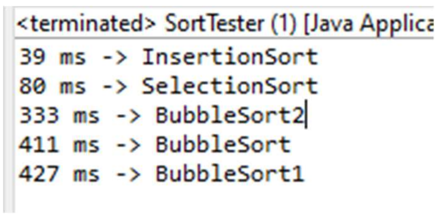
A.

```java
package problem02;

import java.util.Arrays;

public class BubbleSort1 {

    int[] arr;
    public int[] sort(int[] array){
        this.arr = array;
        bubbleSort();
        return arr;

    }
    private void bubbleSort(){

        int len = arr.length;
        for(int i = 0; i < len; ++i) {
            if(isSorted(arr))
                break;
            for(int j = 0; j < len-1; ++j) {
                if(arr[j]> arr[j+1]){
                    swap(j,j+1);
                }
            }
        }
    }

    int[] swap(int i, int j){
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
        return arr;

    }

    boolean isSorted(int[] arr) {
        for(int i=1; i<arr.length; i++)
            if(arr[i-1]>arr[i])
                return false;

        return true;
    }
    public static void main(String[] args){
        int[] test = {21,13,1,-22, 51, 5, 18};
        BubbleSort1 bs = new BubbleSort1();

        System.out.println(Arrays.toString(bs.sort(test)));

    }

}
```

B.

```java
package problem02;

import java.util.Arrays;

public class BubbleSort2 {

    int[] arr;
    public int[] sort(int[] array){
        this.arr = array;
        bubbleSort();
        return arr;

    }
    private void bubbleSort(){

        int len = arr.length;
        for(int i = 0; i < len; ++i) {
            if(isSorted(arr))
                break;
            for(int j = 0; j < len-1-i; ++j) {
                if(arr[j]> arr[j+1]){
                    swap(j,j+1);
                }
            }
        }
    }

    int[] swap(int i, int j){
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
        return arr;

    }

    boolean isSorted(int[] arr) {
        for(int i=1; i<arr.length; i++)
            if(arr[i-1]>arr[i])
                return false;

        return true;
    }
    public static void main(String[] args){
        int[] test = {21,13,1,-22, 51, 5, 18};
        BubbleSort1 bs = new BubbleSort1();

        System.out.println(Arrays.toString(bs.sort(test)));

    }

}
```

C.



```
<terminated> SortTester (1) [Java Applica
39 ms -> InsertionSort
80 ms -> SelectionSort
333 ms -> BubbleSort2
411 ms -> BubbleSort
427 ms -> BubbleSort1
```

Adding the isSorted method in BubbleSort1 did not make the algorithm faster than BubbleSort but made it slower because checking the array is sorted takes a time and it only increases the runtime for special cases (like the best-case). But the asymptotic runtime for both BubbleSort1 and BubbleSort2 is the same.

As for the BubbleSort2, the algorithm is faster because we didn’t include additional operations and only removed unnecessary operation.

## Problem 3:

One possible algorithm is to use the binary search algorithm to find a position in the array where the digit changes from zero to one. If the index of that position is found, it will be trivial to find out the number of zeros and the number of ones.

```
Algorithm countZerosAndOnesHelper(A, n, upper, lower )
    Input: array A of length n, and consists of zeros followed by ones
    Output: array of length 2, first count of 0s and second count of 1s
    mid <- (upper+lower)/2
    if(A[mid-1] = 0 && A[mid] = 1)
        return [mid, n-mid];
    if(A[mid] = 1)
        return countZerosAndOnesHelper(A, n, mid-1, lower)
    else
        return countZerosAndOnesHelper(A, n, upper, mid+1)
```

```
Algorithm countZerosAndOnes(A, n)
    Input: array A of length n, and consists of zeros followed by ones
    Output: array of length 2, first count of 0s, second count of 1s
    return countZerosAndOnes(A, n, n-1, 0);
```