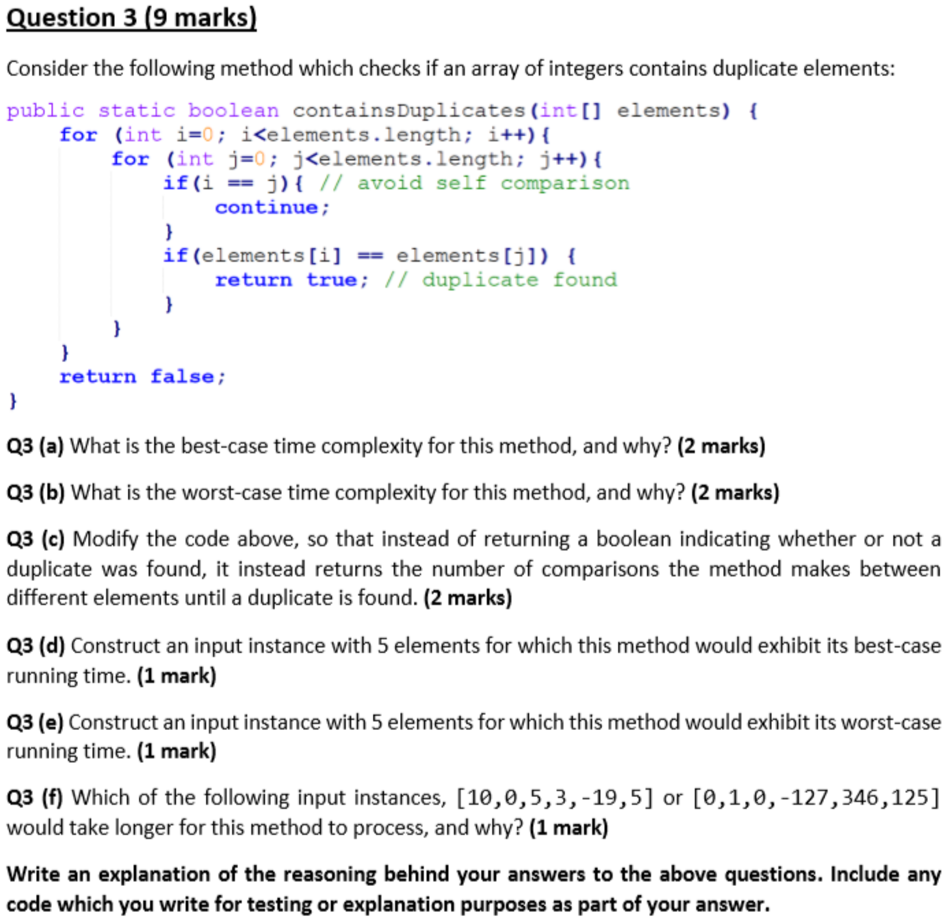
Computational Thinking with Algorithms Problem Sheet (Java)



Question 3 (9 marks)

Consider the following method which checks if an array of integers contains duplicate elements:

```
public static boolean containsDuplicates(int[] elements) {
    for (int i=0; i<elements.length; i++){
        for (int j=0; j<elements.length; j++){
            if(i == j){ // avoid self comparison
                continue;
            }
            if(elements[i] == elements[j]) {
                return true; // duplicate found
            }
        }
    }
    return false;
}
```

Q3 (a) What is the best-case time complexity for this method, and why? **(2 marks)**

Q3 (b) What is the worst-case time complexity for this method, and why? **(2 marks)**

Q3 (c) Modify the code above, so that instead of returning a boolean indicating whether or not a duplicate was found, it instead returns the number of comparisons the method makes between different elements until a duplicate is found. **(2 marks)**

Q3 (d) Construct an input instance with 5 elements for which this method would exhibit its best-case running time. **(1 mark)**

Q3 (e) Construct an input instance with 5 elements for which this method would exhibit its worst-case running time. **(1 mark)**

Q3 (f) Which of the following input instances, `[10,0,5,3,-19,5]` or `[0,1,0,-127,346,125]` would take longer for this method to process, and why? **(1 mark)**

Write an explanation of the reasoning behind your answers to the above questions. Include any code which you write for testing or explanation purposes as part of your answer.

3(a)

O(1) – The best-case time complexity for this method occurs if element 0 and 1 are identical. This involves 2 iterations

#1 → i=0, j=0

#2 → i=0, j=0

The second iteration returns true, there for finishing the loop.

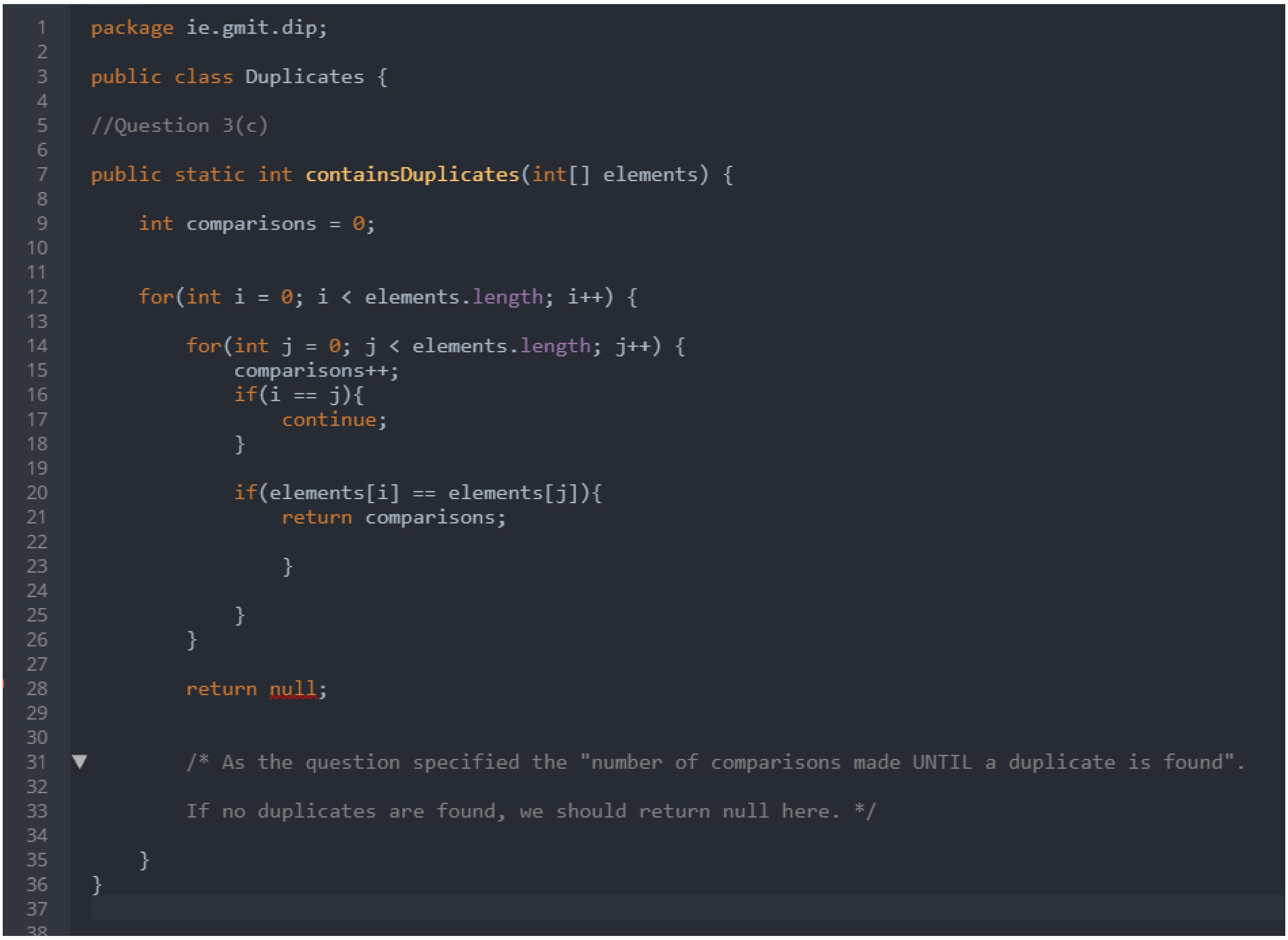
This is technically O(2). However, in Big O notation we simplify the result by removing any constants. Thus, we simplify to O(1).

3(b)

O(n^2) – The worst-case time complexity for this method occurs if there are no duplicate elements. Since there is a nested loop, of n loops each, we iterate n x n times, before returning. Thus O(n^2) is the worst-case complexity of this method.

3(c)

Code modification to return the number of comparisons the method makes between different elements until a duplicate is found.



```
package ie.gmit.dip;

public class Duplicates {

//Question 3(c)

public static int containsDuplicates(int[] elements) {

    int comparisons = 0;


    for(int i = 0; i < elements.length; i++) {

        for(int j = 0; j < elements.length; j++) {
            comparisons++;
            if(i == j){
                continue;
            }

            if(elements[i] == elements[j]){
                return comparisons;

                }

            }
        }

        return null;


        /* As the question specified the "number of comparisons made UNTIL a duplicate is found".

        If no duplicates are found, we should return null here. */

    }
}
```

3(d)

[1, 1, 2, 3, 4]

This would be an input instance where the method would exhibit its best-case running time as element 0 and 1 are the same. As mentioned in 3(a) if element 0 and 1 are identical 2 iterations occur with the 2nd iteration returning true, finishing the loop.

3(e)

[1, 2, 3, 4, 5]

This would be an input instance where the method would exhibit its worst-case running time as there are no duplicate elements. As mentioned in question 3(b) - Since there is a nested loop, of n loops each, we iterate n x n times, before returning. Thus, the worst-case is reflected by the square of the number of elements in the given array above.

3(f)

The first input i.e. [10, 0, 5, 3, -19, 5], would take the longest for this method to process.

The function works by comparing the 0th element with all other elements, then the 1st element with all other elements and so on.

In the second input, i.e. [0, 1, 0, -127, 346, 125], the 0th element has a duplicate.

Whereas In the first input i.e. [10, 0, 5, 3, -19, 5], the 2nd element has a duplicate.

Therefor, clearly the second input [0, 1, 0, -127, 346, 125], will be faster.