DSA Assignment 1

Artem Bahanov (BS18-03)

February 17, 2019

0 Introduction

0.1 Information about codeforces submission

The code that is in java class file was tested on codeforces.com.
The submission number is 50036757.

0.2 Solution for bonus question

Answer: selection sort.
Every iteration selection sorting algorithm selects maximum (or minimum, as we want). So first k elements is first k maximums (or minimums), it means that it is possible to stop sorting array of teams at certain iteration (l) and then get sublist of first l elements. This sublist consists of first maximums. Insertion sort does not provide this feature.

0.3 Important corrections

In the task 3 of Assignment 1 it is very important to say that lines 4 and 5 have to be divided into two parts: comparison and the statement after it. It happens because 1st and 2nd part of this lines run different number of times (**condition part** runs always when program reaches this line but **statement part** not always). Finally, the code should look like this:

```
  isContainedArray(A, B)
1. for i ← 0 to n−1 do
2.   contained ← FALSE
3.   for j ← 0 to n−1 do
4.       if A[i] = B[j] then
5.           contained ← TRUE
6.    if not contained then
7.       return FALSE
8. return TRUE
```

In this solution I will refer to this code, that is semantically the same as code presented in the task sheet.

1 Part A

For part A it is needed to analyze program and show that worst-case runtime of the algorithm is $O(n^2)$.

1.1 Code analysis

For each line I will give a cost - abstract time needed for executing this line and then count the number of executions of each line. The table is shown below.

	isContainedArray(A, B)	cost	times
1.	**for** i $\leftarrow$ 0 **to** n$-$1 **do**	c_1	$n+1$
2.	contained $\leftarrow$ FALSE	c_2	n
3.	**for** j $\leftarrow$ 0 **to** n$-$1 **do**	c_3	$n \cdot (n+1)$
4.	**if** A[i] = B[j] **then**	c_4	n^2
5.	contained $\leftarrow$ TRUE	c_5	n^2
6.	**if not** contained **then**	c_6	n
7.	**return** FALSE	c_7	0
8.	**return** TRUE	c_8	1

Explanation:
1. For-loop executes exactly n + 1 times (but body of loop does n times). It happens because the statement checks n true statements and 1 false.
2. Each simple statement inside a loop executes n times.
3. Similar situation as with (1) line. Loop inside loop.
4. n^2 - simple body statement inside double loop.
5. In worst case this statement can execute EVERY time (e.g. $A = [1, ..., 1], B = [1, ..., 1]$).
6. ...
7 & 8. Return statement executes only once.

The running time of this algorithm is

$$T(n) = c_1 \cdot (n+1) + c_2 \cdot n + c_3 \cdot (n+1) \cdot n + c_4 \cdot n^2 + c_5 \cdot n^2 + c_6 \cdot n + \\ + c_7 \cdot 0 + c_8 \cdot 1 = (c_3 + c_4 + c_5) \cdot n^2 + (c_1 + c_2 + c_3 + c_6) \cdot n + (c_1 + c_8) \quad (1)$$

$(c_3 + c_4 + c_5)$, $(c_1 + c_2 + c_3 + c_6)$ and $(c_1 + c_8)$ can be replaced with α, β and γ respectively, so (**??**) becomes

$$T(n) = \alpha \cdot n^2 + \beta \cdot n + \gamma \tag{2}$$

1.2 Proving that time complexity is $O(n^2)$

To prove that $T(n) = O(n^2)$ we only need to prove that there exist numbers n_0 and c such that

$$0 \leq T(n) \leq c \cdot n^2 \text{ for all } n \geq n_0. \tag{3}$$

by the definition of $O(n)$ It is obvious that $T(n) \geq 0$ for all $n \geq 0$. We can omit this part.
Let us rewrite (**??**):

$$\alpha \cdot n^2 + \beta \cdot n + \gamma \leq c \cdot n^2 \tag{4}$$

division both parts of the inequality by n^2 yields:

$$\alpha + \frac{\beta}{n} + \frac{\gamma}{n^2} \leq c \tag{5}$$

Let us choose $n \geq n_0 = 1$, it is obvious that maximum value of left part of (**??**) is $\alpha + \beta + \gamma$, so it it possible to choose $c = \alpha + \beta + \gamma$.
The inequality holds, then $T(n) = O(n^2)$. **QED**.

1.3 Results

We got that worst-case runtime of the algorithm is $O(n^2)$.

2 Part B

Let us consider $A = B = [1, ..., 1]$, then it is the worst case because it will go through all lines in double loop since return false line will not be executed.
For showing that worst-case runtime of the algorithm is $\Omega(n^2)$ we will use the definition of $\Omega(f(n))$. From 1.1:

$$0 \leq c \cdot n^2 \leq \alpha \cdot n^2 + \beta \cdot n + \gamma \text{ for all } n \geq n_0 \tag{6}$$

$$0 \leq c \leq \alpha + \frac{\beta}{n} + \frac{\gamma}{n^2} \text{ for all } n \geq n_0 \tag{7}$$

If $n \geq n_0 = 1$ and $c = \alpha$, we get:

$$\alpha \cdot n^2 \leq \alpha \cdot n^2 + \beta \cdot n + \gamma \tag{8}$$

$$0 \leq \beta \cdot n + \gamma \tag{9}$$

Inequality (**??**) holds. Then $T(n) = \Omega(n^2)$. **QED**.

3 Part C

From **Part A** and **Part B** we got

$$T(n) = O(n^2); T(n) = \Omega(n^2) \tag{10}$$

then by **Theorem 3.1** (For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$) we get

$$T(n) = \Theta(n^2) \tag{11}$$

It means that worst-case runtime is $\Theta(n^2)$.