Mohamed Bouzabbous: 300376202

Assignment 1.

Question 1.

a) $2n^2 + 10n^3$ is $O(n^3)$

- true.

In fact, in this statement, $10n^3$ dominates $2n^2$ because n^3 grows faster than n^2:

$n^2 < n^3$ for all $n \geq 2$ thus:

$2n^2 + 10n^3 \leq 2n^3 + 10n^3 = 12n^3$

so, for: $c = 12$ and $n_0 = 1$:

$2n^2 + 10n^3 \leq 12n^3$, for all $n \geq 1$.

In conclusion: $2n^2 + 10n^3$ is $O(n^3)$.

b) $\sqrt{n} + 10\log_2 n$ is $O(n)$

- true:

$\sqrt{n} \leq n$ for $\forall n \geq 1$

&

$10\log_2 n \leq 10n$ for all $n \geq 1$; so:

so: $\sqrt{n} + 10\log_2 n \leq n + 10n$

$\Leftrightarrow \sqrt{n} + 10 \leq 11n$ for all $n \geq 1$.

(We have chosen: $c = 11$ and $n_0 = 1$)

In conclusion: $\sqrt{n} + 10\log_2 n$ is $O(n)$

n.h/B $3^n + n^3$ is $\Theta(2^n)$.

False because $3^n + n^3$ is not $O(n^2)$, which we will prove below:

Assume by contradiction that $3^n + n^3$ is $O(2^n)$

Then we conclude that there is $c > 0$ and $n_0 \geq 0$ such that:

$$3^n + n^3 \leq c \cdot 2^n, \quad \forall n \geq n_0$$

$$\text{so: } \frac{3^n}{2^n} + \frac{n^3}{2^n} \leq c, \quad \forall n \geq n_0$$

We have $\lim_{n \to +\infty} \frac{3^n}{2^n} = +\infty$ and $\lim_{n \to +\infty} \frac{n^3}{2^n} = 0$

and $\lim_{n \to +\infty} c = c$:

So: $\lim_{n \to +\infty} \frac{3^n}{2^n} + \frac{n^3}{2^n} = +\infty$: Which leads us to a contradiction for the inequality: $3^n + n^3 \leq c \cdot 2^n$: it cannot count for large n.
$3^n + n^3$ is not $O(2^n)$ so it is not $\Theta(2^n)$

d) $\log_{10} n + 10n^3 + 100$ is $\Omega(n^2)$

False True: For all $n \geq 1$: We have:

$$\log_{10} n + 10n^3 + 100 \geq \log_{10} n + 10n^3 \geq 10n^3 \geq n^2.$$

We note that: $10n^3 \geq 1 \cdot n^2$ ✓

Therefore:

$$\log_{10} n + 10n^3 + 100 \geq 1 \cdot n^2.$$

We used $c = 1$ and $n_0 = 1$.

Question 2:

a) The loop in line 4 runs, in the worst case possible, for all values of x where $x^2 \leq n$. It happens when n is a prime number because there is no divisor found, so the algorithm must check all possible divisors up to $\sqrt{n}$.

(line 4) This loop runs for all x from 2 to $\sqrt{n}$ and checks for each iteration if $n \% x == 0$. Each iteration of this loop take $O(1)$ (it only involves basic arithmetic operations). Finally, in the worst case possible, it runs $\sqrt{n}$ times because it checks for divisors up to $\sqrt{n}$.
In conclusion, $T(n)$ is $O(\sqrt{n})$

b) The best case occurs when n is an even number greater than 2 (ex: n = 4, 6, 8, 10, ...). For this case, the function checks only one divisor (which is 2): (n % 2 == 0) and then returns false, on the first iteration, because n is an even number.

So, it performs, in this case, only one iteration (for i = 2) and returns False. In conclusion, the time complexity there is O(1):

$$B(n) = O(1)$$

question 3: p. 5 /8

```
1.  public static void sortStack (Stack<Integer> original) {
2.      Stack<Integer> stacktemp = new Stack<>();
3.      While (!original.isEmpty()) {
4.          int temp = original.pop();
5.          while (! stacktemp.isEmpty() && stacktemp.peek() > temp){
6.              original.push(stacktemp.pop());
            } // we want to have the elements stored in the temporary
            // stack with the biggest value at the top (peek).
7.          stacktemp.push(temp);
        }
8.      while (! stacktemp.isEmpty()){
9.          original.push(stacktemp.pop());
        }
    }
```

- the while loop at line 3 will remove all the elements from the original stack.
- the while loop at line 5 maintains the croissant order at Stack temp: it compares the value of the peek element of Stacktemp to the value of temp: if the peek value is greater than temp (and stacktemp is not Empty, otherwise we can't compare the peek value to temp because the peek value doesn't exist!)

it pushes the past value to the original stack.
to preserve the order in stacktemp.
croissat

- then, in the line 9, we push the temp value in the the temp stack temp (temporarystack).

- the while boucle in line 8 remove the Elements from the top of stack temp (reminder: this Elements are organized in a croissat order) and pushe them to the original stack. With this operation. the elements are organized in a decroissat order (the smallest value is at the top...).

question 4 p.7/9

a)

1) serve (int i):
    if not Lineup.isEmpty then :
        customer = Lineup.removeFirst()
        Cashier[i] = customer.

2) InterruptService (int i):

    if Cashier[i] != null then
        customer = Cashier[i]
        Lineup.insertFirst(customer)
        Cashier[i] = null

3) NewCustomer (Customer p):
        Lineup.insertLast(p).

4) Give4p (int n):
    for k from 1 to n do
        if not Lineup.isEmpty() then
            Lineup.removeLast()
        else
            break.

b) 3) NewCustomer (A);
- In this operation, A is added to the end of the queue LineUp:
We have LineUp = [A]
2) NewCustomer (B)
Similar to the last case, B is added to the end of the Queue:
We have LineUp = [A, B].
3) NewCustomer (C);
C is added to the end of the Queue; LineUp = [A, B, C].
4) PrintLineUp ();
Output: A, B, C
5) serve (2);
A is sent to Cashier [2] (the second Cashier)
LineUp = [B, C] and Cashier [2] = A.
6) Serve (1);
B is sent to Cashier [1] (the first Cashier).
LineUp = [C] and Cashier [2] = B.
7) serve (1);
C is sent to Cashier [1] and in consequence replaces B. We have LineUp = [] and Cashier [1] = C
8) NewCustomer (D);
D is added to the end of the Queue LineUp;
We have: LineUp = [D].
9) NewCustomer (E);
E is added to the end of the queue LineUp. We have:
LineUp = [D, E].

10) PrintLineup();
the content of the Queue Lineup is displayed on the screen. We have the output is:

D, E.

11) InterruptService(2).
the customer A who is currently at the cashier [2] is sent back to the Front of Lineup (beginning
Lineup = [A, D, E] and cashier[2] = null

12) InterruptServic(1);
C is currently at cashier[1] is sent back to the Front of Lineup.
Lineup = [C, A, D, E] and cashier[1] = null
13) PrintLineup();
the output is: C, A, D, E.
14) NewCustomer(F);
F is added to the end of the Lineup. Now:
Lineup = [C, A, D, E, F].
15) GiveUp(2);
the last 2 customer (E and F) get tired of waiting and abandon the Lineup:
Lineup = [C, A, D].
16) PrintLineup();
output is: C, A, D

In conclusion, the first PrintLineup(); prints on the screen: A, B, C
the second prints: D, E
the third one prints: C, A, D, E
the Forth prints: C, A, D.