Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

Assignment 1 (3.75% - 50 points)

CSI2110/CSI2510 (Fall 2024)

Due: Wednesday October 9, 11:59PM

Late assignment policy: ***1min-24hs late are accepted with 30% off; no assignments accepted after 24hs late.***

Question 1. **[16 points =4+4+4+4]**

Decide if each of the following statements is true or false and give a proof. For a true statement you need to identify the values for the constants c *and* n₀ *as used in the definitions of* big-O, Ω *and* Θ *and show the corresponding inequalities. For a false statement, you need to justify/prove why finding those constants is impossible.*

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

$2n^2 + 10n^3 \le \underline{2}n^3 + 1\underline{0}n$

$2n^2 + 10n^3 \le \underline{12n^3}$

$c = 12$

Test: $n_0 = 1$

$2(1)^2 + 10(1)^3 \le 12(1)^3$ ✓

$\therefore$ True, since $2n^2 + 10n^3 \le 12n^2$ for $n \ge 1$

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

$\sqrt{n} + 10\log_2 n \le n + 10n$

$\sqrt{n} + 10\log_2 n \le 11n$

$C = 11$

Test $n_0 = 1$

$\sqrt{1} + 10\log_2(1) \le 11(1)$ ✓

$\therefore$ True since $\sqrt{n} + 10\log_2(n) \le 11n$ for $n \ge 1$

c) $3^n + n^3$ *is* $\Theta(2^n)$

False, since $3^n > 2^n$, it is always infinite increasing at a higher rate, so since 2^n is not Ω of the function, it is not Θ either

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

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

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

$c = 1$ $\therefore$ True, n^2 is Ω

Test $n_0 = 1$

$\log_{10}(1) + 10(1)^3 + 100 \ge (1)^2$ ✓

Question 2. [10 points]

```
boolean isPrime(int n) { // tests if n is a prime number
// Input: a positive integer n
// Output: return true if n is a prime number, false otherwise
    for (int x = 2; x*x <= n;  x++) {
        if (n % x == 0) {  // found that x divides n
                return false;
        }
    }
    return true; // found no divisor of n
}
```

Note: *when giving the big-Oh , give the tightest upper bound possible. For example, if you can prove that* $f(n)$ *is* $O(n)$, *and that* $f(n)$ *is* $O(n^2)$, *choose the tighter upper bound, i.e.* $f(n)$ *is* $O(n)$.

(a) *(5 pts) Give a big-Oh for* $T(n)$, *the **worst-case** running time of this algorithm for an integer input* n. *Explain how you obtained this worst case.*

If n is a prime number, $x * x = n \longrightarrow x^2 = n$

$x = \sqrt{n}$, so the Big-O of $T(n)$ is $O(\sqrt{n})$

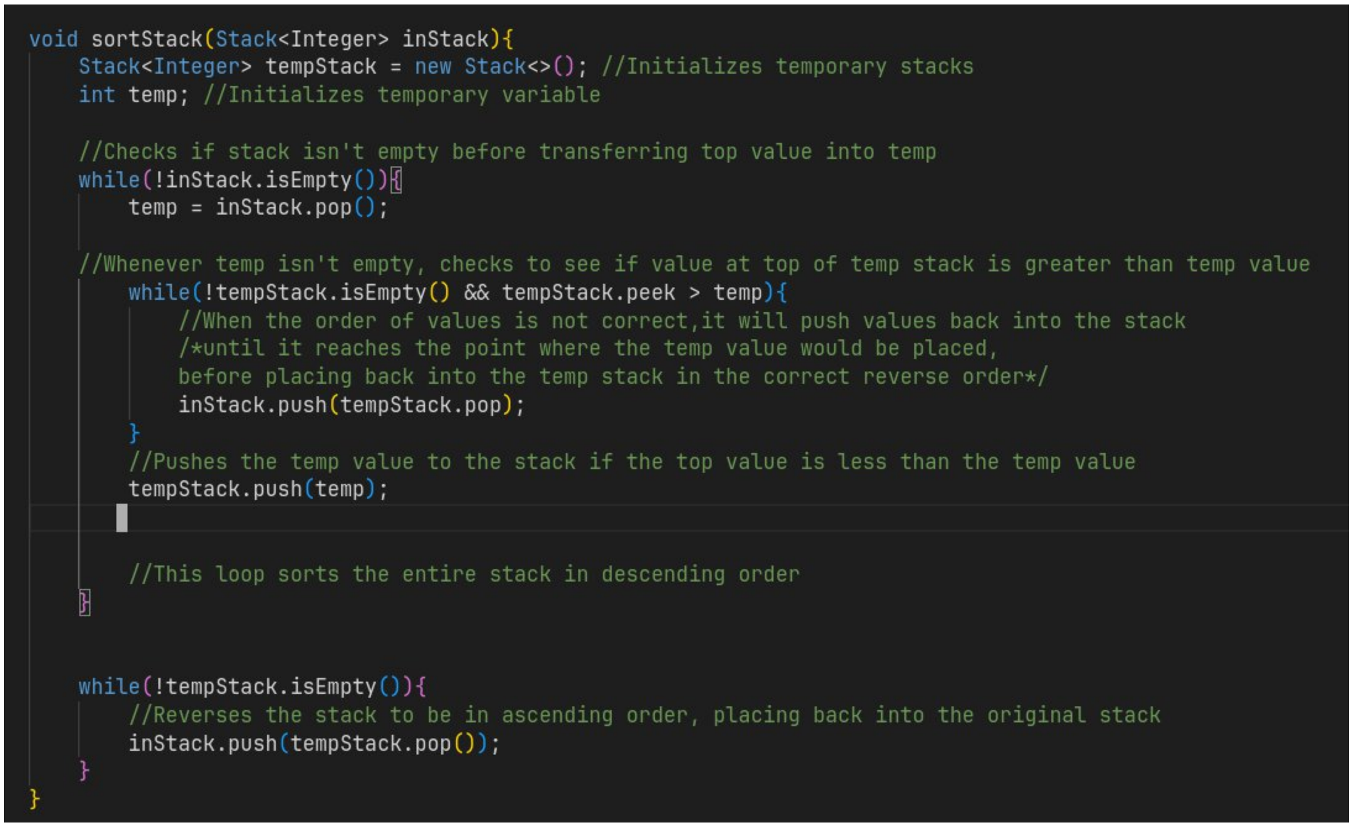
(b) *(5 pts) Give a big-Oh for* $B(n)$, *the **best-case** running time of this algorithm for an input integer input* n. *Explain the type of inputs (not just one example but an infinite sequence of inputs) that will give this best case.*

If n is divisible by 2, it will execute in 1 operation (1st Loop)

If n is 1, it will not enter loop, $\therefore$ Big-O of $B(n)$ is $O(1)$

Question 3. **[12 points=10+2]** (a) Write an algorithm to sort a stack of integers such that the smallest elements are on the top. You can only use one (1) additional temporary stack and a constant number of temporary integer variables. You may not copy the elements into any other data structure (such as an array). The stack supports the following operations: *push*, *pop*, *peek* and *isEmpty*. Give pseudocode or Java-like code. (b) Give the worst-case running time in big-Oh notation. You do not need to prove this, but please show your understanding by mentioning which type of instances would give the worst case.

Note: This question appears often in interviews. First, try to solve it without any hint; do not give up until you think about the question for several days. If after this you can't solve the problem, check the progressive hints on the last page of this assignment. You can also consult a TA or the prof during office hours, who will give you hints, without solving it for you. Asking genAI to solve the problem defeats the purpose of the assignment, and it is against course policies and prevents you from practicing for tests.



```
void sortStack(Stack<Integer> inStack){
    Stack<Integer> tempStack = new Stack<>(); //Initializes temporary stacks
    int temp; //Initializes temporary variable

    //Checks if stack isn't empty before transferring top value into temp
    while(!inStack.isEmpty()){
        temp = inStack.pop();

    //Whenever temp isn't empty, checks to see if value at top of temp stack is greater than temp value
        while(!tempStack.isEmpty() && tempStack.peek > temp){
            //When the order of values is not correct,it will push values back into the stack
            /*until it reaches the point where the temp value would be placed,
            before placing back into the temp stack in the correct reverse order*/
            inStack.push(tempStack.pop);
        }
        //Pushes the temp value to the stack if the top value is less than the temp value
        tempStack.push(temp);


        //This loop sorts the entire stack in descending order
    }


    while(!tempStack.isEmpty()){
        //Reverses the stack to be in ascending order, placing back into the original stack
        inStack.push(tempStack.pop());
    }
}
```

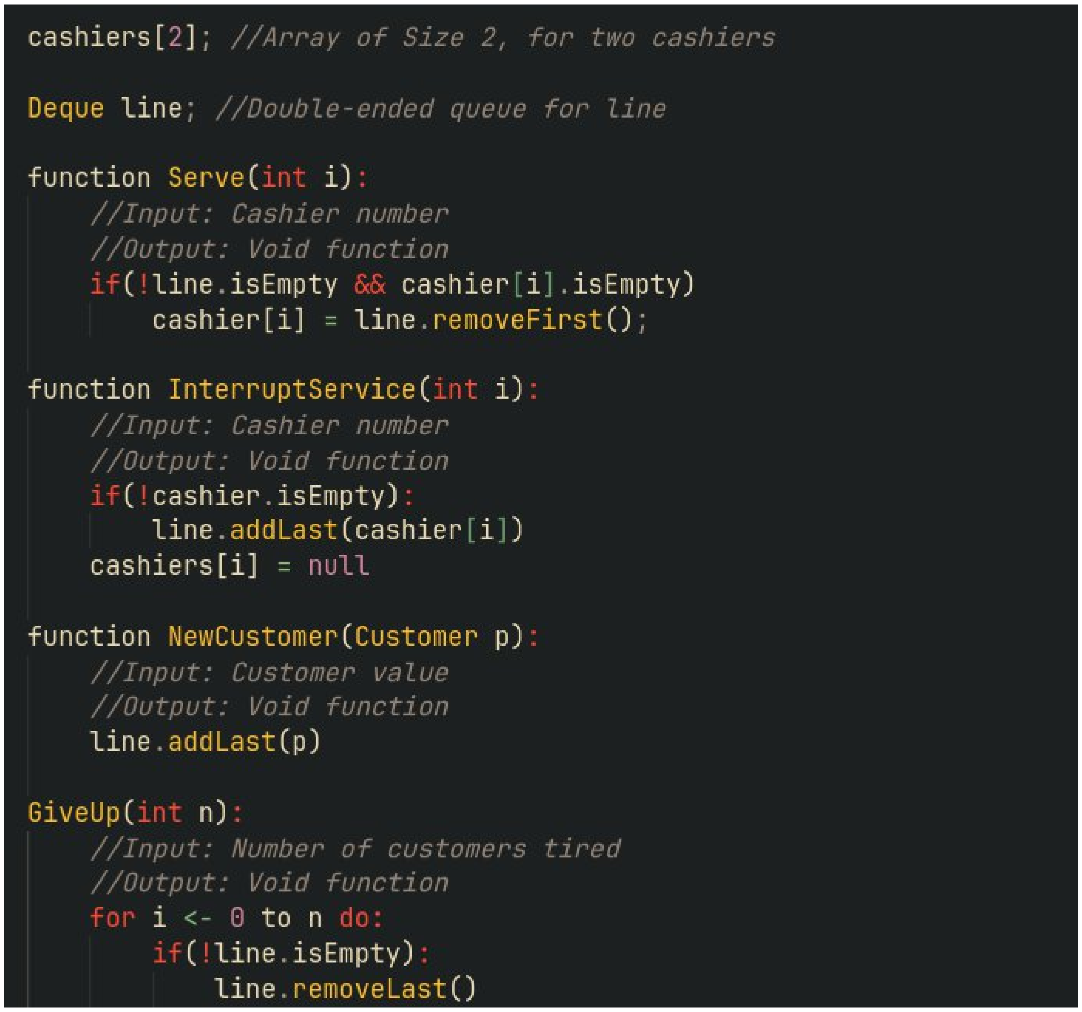
The worst case running time in Big-O will be $O(n^2)$. Assuming the stack isn't empty or less than two values, the worst case running time of the first `while` loop in $O(n)$. Other that same assumption, the second `while` loop inside the first is also $O(n)$. The Big-O of the nested loops is $O(n^2)$. The third `while` is also $O(n)$, but it's Big-O is smaller than that of the nested loop.

Question 4. **[12 points = 10+2]** You need to provide operations for simulating a store with two cashiers (each able to serve one customer at a time) and a single (first-come first-serve) lineup of customers.

a) Provide pseudocode for the following operations:

- `Serve(int i):` Send first customer in line to cashier `i (i∈{1,2}),` if line is not empty.
- `InterruptService(int i):` Send customer currently at cashier `i` back to the beginning of the line.
- `Newcustomer(Customer p):` A new customer arrives at the end of the cashier lineup.
- `GiveUp(int n):` the last n customers in line get tired of waiting and abandon the lineup.

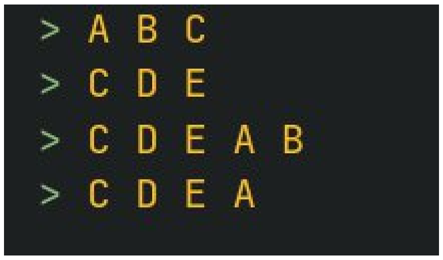
Use the most suitable abstract data type (ADT) seen in class as the basis to design the operations above. Your operations must simply call operations on the chosen ADT, without specifying their implementation. Simplicity counts.



```
cashiers[2]; //Array of Size 2, for two cashiers

Deque line; //Double-ended queue for line

function Serve(int i):
    //Input: Cashier number
    //Output: Void function
    if(!line.isEmpty && cashier[i].isEmpty)
        cashier[i] = line.removeFirst();

function InterruptService(int i):
    //Input: Cashier number
    //Output: Void function
    if(!cashier.isEmpty):
        line.addLast(cashier[i])
    cashiers[i] = null

function NewCustomer(Customer p):
    //Input: Customer value
    //Output: Void function
    line.addLast(p)

GiveUp(int n):
    //Input: Number of customers tired
    //Output: Void function
    for i <- 0 to n do:
        if(!line.isEmpty):
            line.removeLast()
```

b) Perform the following operations and show the output, assuming `PrintLineup()` prints the customers in the lineup from beginning to the end.

```
Newcustomer(A);
Newcustomer(B);
Newcustomer(C);
PrintLineup();
Serve(2);
Serve(1);
Serve(1);
Newcustomer(D);
Newcustomer(E);
PrintLineup();
InterruptService(2);
InterruptService(1);
PrintLineup();
Newcustomer(F);
GiveUp(2);
PrintLineup();
============
```



```
> A B C
> C D E
> C D E A B
> C D E A
```

HINTS for question 3:

Hint 1:
One way to sort the array is to iterate through the array and insert each element into a new array in sorted order. Can you do this using an auxiliary stack?

Hint 2:
Imagine your secondary stack is sorted. Can you insert elements into the secondary stack in sorted order using just the primary stack plus a temporary variable that holds 1 element?

Hint 3: Next page (don't go there so soon!!!)

Final hint for Question 3

Your original stack is S1 and your final stack is S1. At every step, keep S2 sorted with the largest element on top and insert the top element of stack S1 into stack S2, using S1 as auxiliary memory. When S2 contains all elements sorted with the largest element on top, move the elements of stack S2 back to stack S1.

Example:

input:	intermediate iteration:		final iteration:		output:
S1	S1	S2	S1	S2	S1
(top)					
8				12	1
1				10	3
12				8	5
3		12		7	7
5	5	8		5	8
10	10	3		3	10
7	7	1		1	12
(bottom)	(bottom)	(bottom)	(bottom)	(bottom)	(bottom)

Note: this question appeared in a previous final exam, with the above hint given.