FUNDAMENTALS
+
BASIC DATA STRUCTURES

CS062

DATA STRUCTURES AND ADVANCED PROGRAMMING

Midterm I Study Guide

Alexandra Papoutsaki
she/her/hers

Information

- Midterm I: Monday February 26, 1:15-2:30pm in the same room we always meet.
- You can bring one hand-written (ok *hand-written* on tablets and then printed) sheet of papers (i.e. two pages).
- Review lecture slides and code, quizzes, labs, and assignments. Use the four practice problems in this presentation.
- Practice writing code on paper.

Java Basics

- Chapter 1.1 (Pages 8-35).
- Chapter 1.2 (Pages 64-77, 84–88, 96–99, 107).
- Quick overview of Java tutorials.
  - https://docs.oracle.com/javase/tutorial/java/
- In general, review the basics of OOP and of Java so that you are comfortable reading and writing code.

ArrayLists

- Chapter 1.3 (Pages 136-137).
- code
- Java Oracle API https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html
- Amortized and worst-case time analysis.

Analysis of Algorithms

- Chapter 1.4 (Pages 172-205).
- Experimental analysis including doubling hypothesis. Pick two pairs of the largest input sizes and check that the T(n)/T(n/2) is consistently expressed as some power of 2.
- Mathematical analysis including reviewing (not memorizing) useful approximations of sums.
- Order of growth classifications.
- Review of running time of operations on array lists, linked lists, stacks and queues.

Singly Linked Lists

- Chapter 1.3 (Pages 142-146).
- code
- Worst-case time analysis for standard operations.

Doubly Linked Lists

- Chapter 1.3 (Pages 126-157).
- code
- Java Oracle API.
  - https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html
- Worst-case time analysis for standard operations.

Stacks and Queues

- Chapter 1.3 (Pages 142-146).
- code for alternative implementations
- Worst-case time analysis for standard operations based on the underlying implementation

Practice Problem 1

You are given the following Java code that implements a simplified version of a stack of Strings.

```
1.      public class StringStack {
2.          private String[] a;
3.          private int n = 0;
4.          public StringStack(int size)
5.          {a = new String[size];}
6.          public void push(String item)
7.          {a[n++] = item;}
8.          public String pop() {
9.          {return a[--n];}
10.        public static void main(String args[]) {
11.            StringStack ss = new StringStack(10);
12.            ss.push("47");
13.            String s = ss.pop();
14.            System.out.println(s);
15.        }
16.    }
```

In the next page, mark with an X in each of the rows what line numbers correspond to the description.

Practice Problem 1 (cont'd)

	1	2	3	4-5	6-7	8-9	10-15	11	12	13	14
Defines a constructor											
Names a class											
Invokes a method (excluding constructor)											
Invokes a constructor											
Initializes a local variable											
Declares an instance variable											
Creates an object											
Implements an instance method (excl. constructor)											
Implements a static method											
Applies a unary operator											

Practice Problem 2

a. For each function $f(n)$ in the table below, please write down $O(f(n))$ in the simplest possible form. For example, if $f(n)$ was $2n$, then $O(f(n))$ would be written as n (or $O(n)$, if you like).

b. Order the answers from part a so that they are in increasing order of rate of growth, i.e., write the slowest growing function on the left (i.e. the fastest overall) and the fastest growing on the right (i.e. the slowest overall) with the others between in order of growth for large values of n.

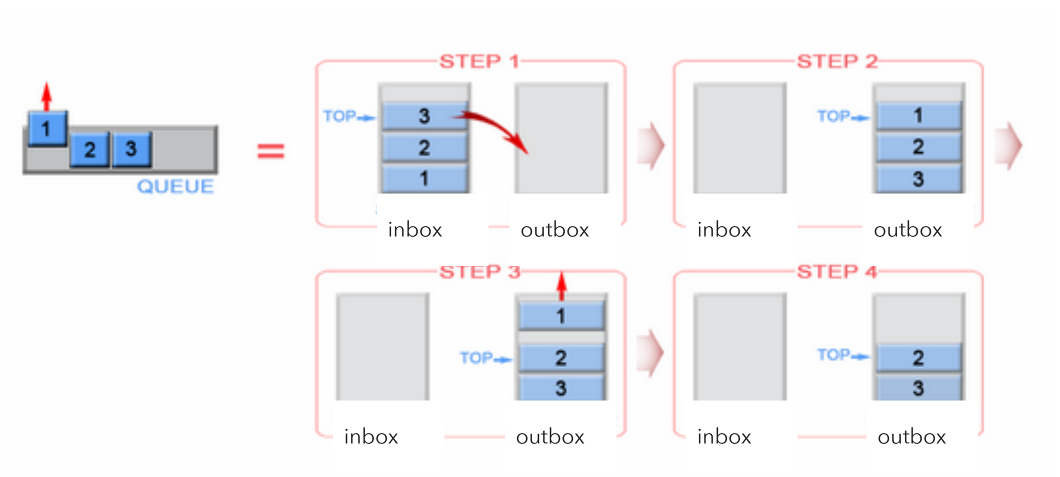
Function	Big-O
$100n \log n + 100n$	
$n^3 + 50n^2 + 10000$	
$10n^2 + 20n \log n$	
2^{12}	
2^n	
$30n$	
$50n \log n + n!$	
$20 \log n + 1000$	

Practice Problem 3

- We will be adding a new method to the class SinglyLinkedList we built together with the following signature: public void keep(int howMany)
- The method should modify the list so it only keeps the first howMany elements, dropping the rest of the elements from the list. E.g., if myList contains 10 elements, then executing myList.keep(6) should result in myList having only the first 6 elements of the list.
- a. Write the pre- and post-conditions (what assumptions need to be met for the method to execute correctly and what will be true after the execution of the method, respectively) in plain English.
- b. List at least one special case that either violates your preconditions or requires special handling.
- c. Write the code for keep. If the preconditions are violated, you should throw an IllegalArgumentException.

Practice Problem 4

- Fill in the following class to implement a queue using two stacks. When elements are enqueued, they are added to the inbox stack. During dequeue or peek operations, elements are transferred from the inbox stack to the outbox stack as needed.
- Here is an example of how it works:



Popping element "1" from the queue

https://stackoverflow.com/questions/69192/how-to-implement-a-queue-using-two-stacks

```java
public class TwoStackQueue<E> {

    ArrayListStack<E> inbox;
    ArrayListStack<E> outbox;

    public TwoStackQueue() implements Queue<E>{
        inbox = new ArrayListStack<E>();
        outbox = new ArrayListStack<E>();
    }

   public int size() {
        // FIX ME
   }

    public void enqueue(E element) {
        // FIX ME
    }

    private void transferElements() {
        // FIX ME
    }

    public E peek() {
        // FIX ME
    }

    public E dequeue() {
        // FIX ME
    }

    public boolean isEmpty(){
        // FIX ME
    }

    public static void main(String args[]) {
        TwoStackQueue<Integer> mq = new TwoStackQueue<Integer>();
        System.out.println(mq.isEmpty()); //true
        for (int i = 0; i < 8; i++){
            mq.enqueue(i);
        }
        System.out.println("Size: " + mq.size());
        System.out.println("Peek: " + mq.peek());
        for (int i = 0; i < 8; i++) {
            System.out.println(mq.dequeue()); // 0 1 2 3 4 5 6 7
        }


    }
}
```

Practice Problem 1 - ANSWER

	1	2	3	4-5	6-7	8-9	10-15	11	12	13	14
Defines a constructor				X							
Names a class	X										
Invokes a method (excluding constructor)									X	X	X
Invokes a constructor				X				X			
Initializes a local variable								X		X	
Declares an instance variable		X	X								
Creates an object				X				X	X		
Implements an instance method (excl. constructor)					X	X					
Implements a static method							X				
Applies a unary operator					X	X					

Practice Problem 2 - ANSWER

a. For each function $f(n)$ in the table below, please write down $O(f(n))$ in the simplest possible form. For example, if $f(n)$ was $2n$, then $O(f(n))$ would be written as n (or $O(n)$, if you like). ->

b. Order the answers from part a so that they are in increasing order of rate of growth, i.e., write the slowest growing function on the left (i.e. the fastest overall) and the fastest growing on the right (i.e. the slowest overall) with the others between in order of growth for large values of n.

$1, \log n, n, n \log n, n^2, n^3, 2^n, n!$

Function	Big-O
$100n \log n + 100n$	$n \log n$
$n^3 + 50n^2 + 10000$	n^3
$10n^2 + 20n \log n$	n^2
2^{12}	1
2^n	2^n
$30n$	n
$50n \log n + n!$	$n!$
$20 \log n + 1000$	$\log n$

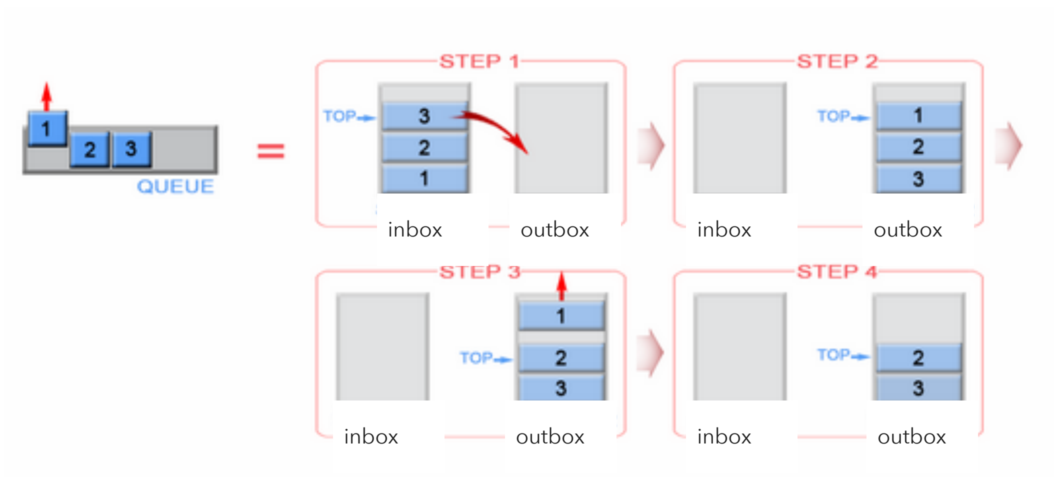
Practice Problem 3 - ANSWER

- a.
  - pre-condition: howMany>=0 && howMany<=size
  - post-condition: list has howMany elements
- b. howMany ==0, howMany==size, howMany<0 or howMany>=size
- c. ->

```java
public void keep(int howMany) {
        if (howMany > size || howMany < 0) {
            throw new IllegalArgumentException();
        }
        if(howMany==0){
            head = null;
        }
        else if(howMany == size){
            return;
        }
        else{
            Node finger = head;
            // Traverse the list until the (howMany - 1)th element
            for (int i = 0; i < howMany - 1; i++) {
                finger = finger.next;
            }
            // Set the next of the (howMany - 1)th element to null,
            // effectively cutting off the rest of the list.
            finger.next = null;
        }
        size = howMany;
}
```

Practice Problem 4 - ANSWER

- Fill in the following class to implement a queue using two stacks. When elements are enqueued, they are added to the inbox stack. During dequeue or peek operations, elements are transferred from the inbox stack to the outbox stack as needed.
- Here is an example of how it works:



Popping element "1" from the queue

https://stackoverflow.com/questions/69192/how-to-implement-a-queue-using-two-stacks

```java
public class TwoStackQueue<E> implements Queue<E>{

    ArrayListStack<E> inbox;
    ArrayListStack<E> outbox;

    public TwoStackQueue() {
        inbox = new ArrayListStack<E>();
        outbox = new ArrayListStack<E>();
    }

    public int size() {
        return inbox.size() + outbox.size();
    }

    public void enqueue(E element) {
        inbox.push(element);
    }

    private void transferElements() {
        while (!inbox.isEmpty()) {
            outbox.push(inbox.pop());
        }
    }

    public E peek() {
        if(outbox.isEmpty()){
            transferElements();
        }
        return outbox.peek();
    }

    public E dequeue() {
        if(outbox.isEmpty()){
            transferElements();
        }
        return outbox.pop();
    }

    public boolean isEmpty(){
        return inbox.isEmpty() && outbox.isEmpty();
    }
}
```