Lab 5: Checkpoint 1 study guide

Basic Data Structures

Checkpoint 1

Information

- Checkpoint 1 is tomorrow in class.

- You can bring a note sheet
  - hand-written (ok hand-written on tablets and then printed)
  - back and front sheet of paper (i.e., two pages)
  - **NO slides shrunk and copy pasted.**

- Studying
  - Review lecture slides (including practice problems) and links to code.
  - Go over quizzes, labs, and assignments.
  - Use the five practice problems in this presentation.
  - Practice writing code on paper.

Java Basics

LECTURES 1-4

- 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).
- 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 Big O, Big Omega, and Big Theta difference
- Order of growth classifications.
- Review of running time of operations on array lists, linked lists, stacks, and queues.

Logarithms - refresher

- $a^b = c \rightarrow b = \log_a c$
- $\log_a a = 1, \log_a 1 = 0$
- $\log_a \frac{x}{y} = \log_a x - \log_a y$
- $\log_a x \times y = \log_a x + \log_a y$
- $\log_a x^y = y \times \log_a x$
- $\log_a x = \frac{\log_b x}{\log_b a}$
- $x^{\log_a y} = y^{\log_a x}$
- $a^{\log_a x} = x$
- $\lg n! \approx nlgn$

Summations - refresher

- $\sum_{i=1}^{n} i = 1 + 2 + \ldots + n$
- $\sum_{i=1}^{n} c = c + c + \ldots + c = n \times c$, assuming c does not depend on i
- $\sum_{i=1}^{n} c \times f_i = c \times \sum_{i=1}^{n} f_i$
- $\sum_{i=1}^{n} (f_i + g_i) = \sum_{i=1}^{n} f_i + \sum_{i=1}^{n} g_i$
- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \sim n^2$
- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \sim n^3$
- $\sum_{i=0}^{n} 2^i = 2^{(n+1)} - 1$
- $\sum_{i=0}^{n} (\frac{1}{2})^i = 1 + \frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^n} \sim 2$
- $\sum_{i=1}^{n} \frac{1}{i} \sim \ln n$

Linked Lists

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

Stacks and Queues

- Chapter 1.3 (Pages 142-146)
- Worst-case time analysis for standard operations based on the underlying implementation (ArrayList vs Linked List)

Practice problems

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

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	11	12	13	14	15
Defines a constructor															
Names a class															
Invokes a method (excluding 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 $O(n)$.

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.

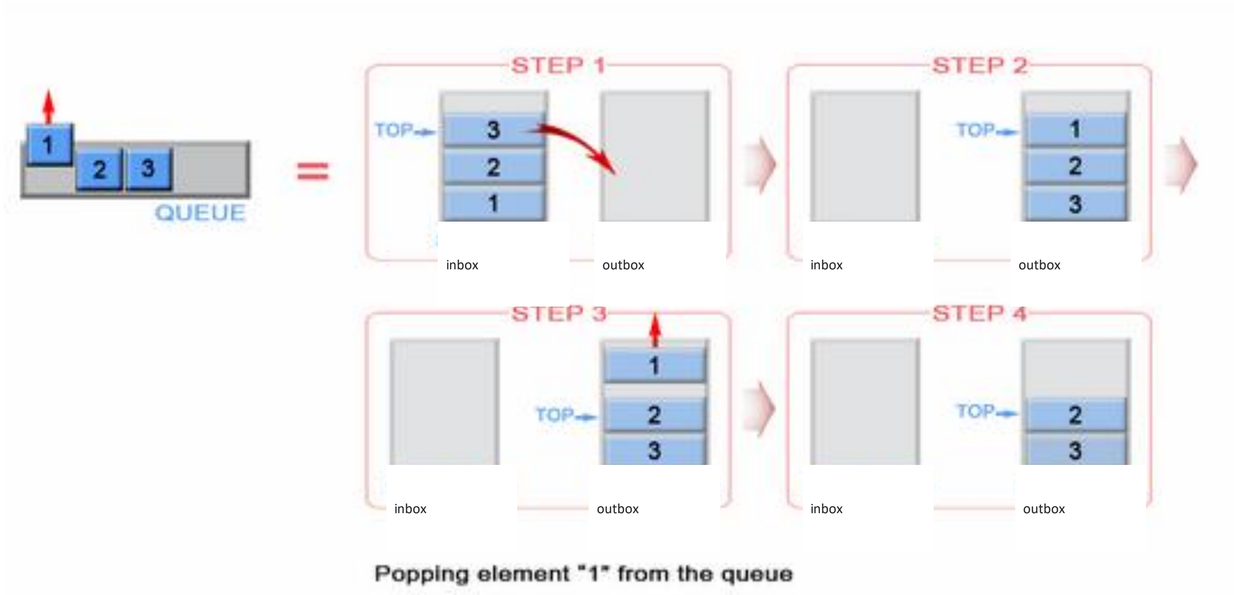
Function	Big-O
$100n\log n + 100n$	
$n^3 + 50n^2 + 10000$	
$10n^2 + 20n\log n$	
2^{12}	
2^n	
$30n$	
$50n\log n + n!$	
$20\log n + 1000$	

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 a SinglyLinkedList 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:



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

```
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 5

- For each of the following pieces of code, find the number of times operation() is called as a function of the input size n. Express your answer in terms of the order of growth of the running time.

a.
```
for (int i = 10; i < n + 5; i += 2){
   operation();
  }
```

b.
```
for (int i = 1; i < n; i *= 2){
   operation();
  }
```

c.
```
for (int i = 10; i < n; i++){
      for (int j = 0; j < n; j += 2){
          operation();
      }
  }
```

d.
```
for (int i = 1; i <= n; i++){
      for (int j = 1; j <= i; j ++)
          operation();
      }
  }
```

e.
```
for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= n; j += i){
          operation();
      }
     for (int j = 1; j <= i; j++){
          operation();
     }
  }
```

Solutions

Practice Problem 1 (cont'd)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Defines a constructor					x										
Names a class	x														
Invokes a method (excluding constructor)											x	x			
Initializes a local variable										x					
Declares an instance variable		x	x												
Creates an object					x					x					
Implements an instance method (excl. constructor)						x	x								
Implements a static method									x	x	x	x	x	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 $O(n)$. (See table)

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
$50n\log n + n!$	$n!$
$20\log n + 1000$	$\log n$

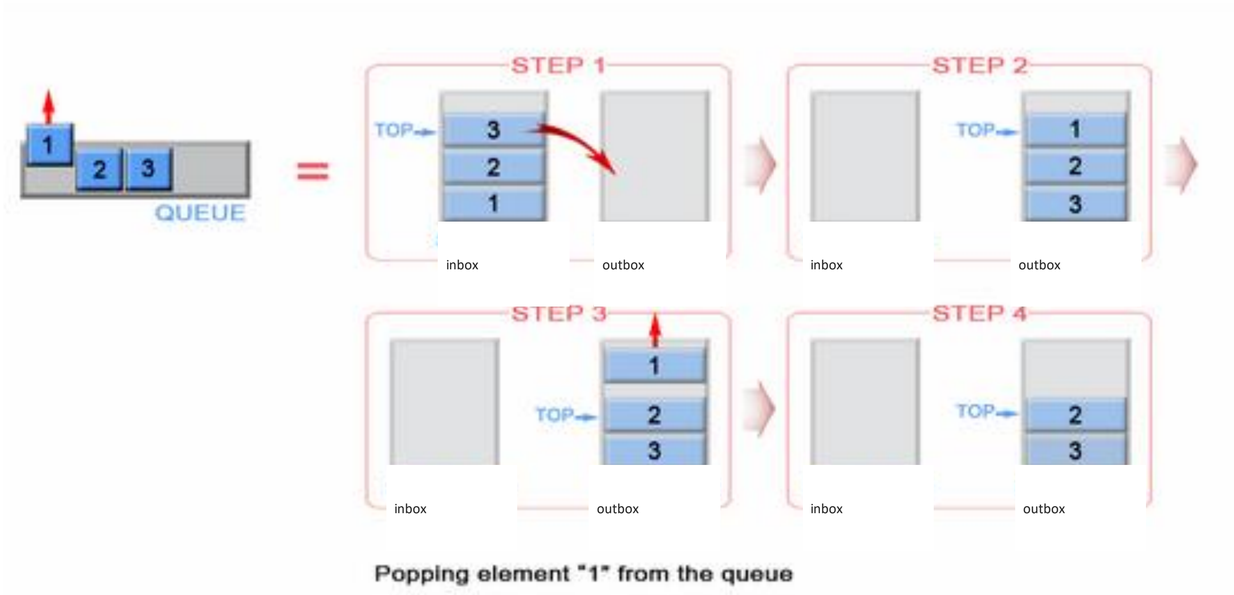
Practice Problem 3 - ANSWER

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

```
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:



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

```
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();
  }
}
```

Practice Problem 5 (a) - ANSWER

a. for (int i = 10; i < n + 5; i += 2){
operation();
}

- operation is called $(n + 5 - 10)/2$ times, which is in the order of $O(n)$.

Practice Problem 5 (b) - ANSWER

```
b. for (int i = 1; i < n; i *= 2){
    operation();
   }
```

- The number of steps needed to get from 1 to n by doubling is $\log_2 n$. The order of growth is $O(\log n)$---the base is not important.

Practice Problem 5 (c) - ANSWER

```
c. for (int i = 10; i < n; i++){
       for (j = 0; j < n; j += 2){
           operation();
       }
   }
```

- operation is called $(n-10) \times \frac{n}{2} = \frac{1}{2}n^2 - 5n$ times, therefore the order of growth is $O(n^2)$.

Practice Problem 5 (d) - ANSWER

```
d. for (int i = 1; i <= n; i++){
       for (j = 1; j <= i; j ++)
           operation();
       }
   }
```

- For i = 1, the inner loop is called 1 times
- For i = 2, the inner loop is called 2 times
- ...
- For i = n, the inner loop is called n times
- Overall, 1+2+...+n = $1 + 2 + \ldots + n = \frac{n(n+1)}{2} \sim O(n^2)$

Practice Problem 5 (e) - ANSWER

```
e. for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= n; j += i){
         operation();
      }
      for (int j = 1; j <= i; j++){
         operation();
      }
   }
```

- Note that the two inner for loops are independent .
- The first inner loop combined with outer loop run in the order of $O(n\log n)$.
  - When i=1, the inner loop performs n=n/1 operations
  - When i=2, the inner loop performs n/2 operations
  - ...
  - When i=n, the inner loop performs 1 = n/n operations
  - Overall, $\frac{n}{1}+\frac{n}{2}+\ldots\frac{n}{n}=\sum_{i=1}^{n}\frac{n}{i}=n\times\sum_{i=1}^{n}\frac{1}{i}\sim n\ln n\sim O(n\log n)$
- The second inner loop combined with outer loop run in the order of $O(n^2)$ (look at problem d).
- Overall, the order of growth for the entire code fragment is $O(n\log n+n^2)=O(n^2)$