Test 1 Practice : Compsci 201

Owen Astrachan

February 6, 2019

Name: ____________________________________________

NetID/Login: ________________

Community standard acknowledgment (signature) ____________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>Value</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>10 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>20 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>18 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>21 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>81 pts.</td>
<td></td>
</tr>
</tbody>
</table>

This test has 14 pages, be sure your test has them all. Write your NetID clearly on each page of this test (worth 1 point).

In writing code you do not need to worry about specifying the proper import statements. Don’t worry about getting function or method names exactly right. Assume that all libraries and packages we’ve discussed are imported in any code you write. You can write any helper methods you would like in solving the problems. You should show your work on any analysis questions.

You may consult your six (6) note sheets and no other resources. You may not use any computers, calculators, cell phones, or other human beings. Any note sheets must be turned in with your test.
PROBLEM 1:  *(I-O, I-O, it's off to the pasture we go (12 points))*

**Part A: 8 points**
What is printed by each `System.out.printf` statement?

```java
String s = new String("hello");
String t = s;
String[] a = {"apple", "cherry", "banana", "guava", "orange"}
String[] b = a;
s = s + " world";
a[0] = "lemon";

System.out.printf("%s \t %d\n", s, s.length());
System.out.printf("%s \t %d\n", t, t.length());
System.out.printf("%s %s\n", a[0], b[0]);
System.out.printf("%s\n", a[4] + b[4]);
```

**Part B: 4 points**
True or False, Explain/Justify each answer

- If two string variables `s` and `t` have different values for `s.toString()` and `t.toString()` then `s.equals(t)` must be *false* (the strings cannot be equal).

- If two string variables `s` and `t` have different values for `s.toString()` and `t.toString()` then `s.hashCode() == t.hashCode()` must be *false* (the strings must have different values for `.hashCode()`).
The method `stringSubset` whose code is shown below correctly returns true if and only if `String` parameter `a` is a subset of `String` parameter `b` when each string parameter represents a space delimited set of strings. Consider the examples shown.

<table>
<thead>
<tr>
<th>Call</th>
<th>Return</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stringSubset(&quot;the CAT in the hat&quot;, &quot;cat hat&quot;)</code></td>
<td><code>false</code></td>
<td>&quot;the&quot; contained in <code>a</code> and not in <code>b</code></td>
</tr>
<tr>
<td><code>stringSubset(&quot;cat hat&quot;, &quot;the cat in the HAT&quot;)</code></td>
<td><code>true</code></td>
<td>all words in <code>a</code> are in <code>b</code></td>
</tr>
<tr>
<td><code>stringSubset(&quot;one fish two fish&quot;, &quot;one two FISH&quot;)</code></td>
<td><code>true</code></td>
<td>upper/lower case ok</td>
</tr>
</tbody>
</table>

You’ll be asked several questions about the method `stringSubset`.

```java
public boolean stringSubset(String a, String b) {
    String[] aa = a.toLowerCase().split(" ");
    String[] bb = b.toLowerCase().split(" ");

    Set<String> as = new HashSet<>(Arrays.asList(aa));
    Set<String> bs = new HashSet<>(Arrays.asList(bb));

    return bs.containsAll(as);
}
```

Part A (2 points)
What type of object does the method `a.toLowerCase()` return?

Part B (2 points)
Can the return statement be replaced by `return bs.containsAll(as) == true`? Answer Yes or No and explain your answer for credit.
Part C (3 points)
Would the code still work correctly if each occurrence of HashSet is replaced by TreeSet? Note that both HashSet and TreeSet implement the Set interface. Answer Yes or No and explain your answer for credit.

Part D (3 points)
Is true or false returned by the call stringSubset("CaT hat HAT HAT", "the cat in the HAT")? Explain why for credit.
As an example of how to think about some of the questions in this section, consider the method `stuff` below. The runtime complexity of this method is $O(n)$ and the value returned by the function is $O(n^2)$ for parameter $n$. As a concrete example, note that when $n = 100$ the loop executes 100 times doing an $O(1)$ operation each time. The value returned is $100 \times 100 = 100^2$. Note that even if the return statement was `return sum*2` that the value returned would still be $O(n^2)$.

```java
public int stuff(int n){
    int sum = 0;
    for(int k=0; k < n; k++){
        sum += n;
    }
    return sum;
}
```

In all these problems $n$ is a positive number. In each problem you should provide two big-Oh expressions: one for runtime and one for value returned. Briefly justify each answer you provide.

Part A (4 points)

What is the runtime complexity and the value returned by method `evaluate` below in terms of $n$? Use big-Oh and justify your answer briefly. Label the run-time and the value returned. Justify each.

```java
public int evaluate(int n){
    int sum = 0;
    for(int k=0; k < n; k++){
        sum += 1;
    }
    for(int k=0; k < n; k++){
        sum += 2;
    }
    for(int k=0; k < n; k++){
        sum += 3;
    }
    return sum;
}
```
Part B (4 points)
What is the runtime complexity and the value returned by method `calculate` below in terms of `n`? Use big-Oh and justify your answer briefly. For this Part B the runtime and the value returned have the same big-Oh expression.

```java
public int calculate(int n) {
    int sum = 0;
    for (int k = 0; k < n; k += 5) {
        for (int j = 0; j < n; j += 10) {
            sum += 1;
        }
    }
    return sum;
}
```

Part C (4 points)
What is the runtime complexity and the value returned by method `value` below in terms of `n`? Use big-Oh and justify your answer briefly. You should have two answers, each justified.

```java
public int value(int n) {
    int result = 1;
    for (int k = 0; k < n; k++) {
        result *= 2;
    }
    return result;
}
```
Part D (4 points)
What is the runtime complexity and the value returned by method \texttt{mathize} below in terms of \( n \)? Use big-Oh and justify your answer briefly. You should have two answers, each justified. If it helps, assume \( n \) is a power of two.

```java
public int mathize(int n){
    int sum = 0;
    int amount = n;
    while (n > 0){
        sum += amount;
        n = n/2;
    }
    return sum;
}
```

Part E (4 points)
Consider the method \texttt{stuff} from the beginning of this problem, reproduced below. Recall that the runtime complexity is \( O(n) \) and the value returned is \( O(n^2) \).

```java
public int stuff(int n){
    int sum = 0;
    for(int k=0; k < n; k++){
        sum += n;
    }
    return sum;
}
```

Give big-Oh expressions for both the runtime complexity and the value returned for each of the expressions below. Justify your answer briefly.

```java
int x = stuff(stuff(n)); // big-Oh for runtime and value returned
int y = stuff(stuff(n/4)); // big-Oh for runtime and value returned
```
The class `Interval` is defined in `Interval.java` which is printed after this page. As the comments in the class say, `Interval a = new Interval(3,7)` would create an `Interval` object representing [3,7] on the x-axis and assign `a` to reference that object. The interval here is the set of points between, and including (3,0) and (7,0) on the x-axis. Only integer end-points are used in the class `Interval`.

The first few questions are based on the code in the class `Interval` and the code below that uses `Interval` objects and methods.

```java
Interval a = new Interval(3,7);
Interval b = new Interval(10,12);
Interval c = new Interval(11,15);
Interval d = new Interval(c);

System.out.println("length of "+a+" = "+a.length());
System.out.printf("%s overlaps %s = %s\n",a,b,a.overlaps(b));
System.out.printf("%s overlaps %s = %s\n",b,c,b.overlaps(c));
System.out.printf("distance between %s and %s = %d\n",a,b,a.distanceBetween(b));
System.out.printf("%s equals %s = %s\n",c,d,c.equals(d));
System.out.printf("%s equals %s = %s\n",d,d,d.equals(d));
```

The output of executing this code is shown below.

```
length of [3,7] = 4
distance between [3,7] and [10,12] = 3
```

**Part A (2 points)**

Explain why the object referenced by variable `a` is printed as [3,7]. Reference code in the `Interval` class.

**Part B (2 points)**

If the second printing statement is changed so that rather than `a.overlaps(b)` the expression `b.overlaps(a)` is used, will the output change? (Yes or No) Explain your answer.

(continued)
import java.util.*;

/** * Class representing an interval on the x-axis, where [x,y] is * an interval between (x,0) and (y,0). So [3,5] represents * the interval between/including (3,0) and (5,0) whereas [-1,4] represents * the interval between/including (-1,0) and (4,0) */

public class Interval implements Comparable<Interval> {

    private int myStart; // beginning of interval
    private int myEnd; // end of interval

    /** 
     * Construct the interval [start,end]. It's an error 
     * to create an interval with start > end 
     * @param start is beginning of interval 
     * @param end is end of interval 
     */
    public Interval(int start, int end) {
        myStart = start; myEnd = end;
        if (myStart > myEnd) {
            throw new IllegalArgumentException(myStart + " > " + myEnd);
        }
    }

    /** 
     * Construct an Interval that's a duplicate of i 
     * @param i is the interval being copied 
     */
    public Interval(Interval i) {
        myStart = i.myStart; myEnd = i.myEnd;
    }

    /** * Compares this interval to other. 
     * Compare first using starting values. If equal use ending values 
     * @return a value indicating how this Interval compares to other 
     */
    @Override
    public int compareTo(Interval other) {
        int startDiff = myStart - other.myStart;
        if (startDiff == 0) {
            return myEnd - other.myEnd;
        }
        return startDiff;
    }

    /** 
     * Returns true if and only if this interval overlaps 
     * the interval passed to the method. 
     * @param other is the interval considered for overlap with this one 
     */
    public boolean overlaps(Interval other) {
        if (myStart < other.myEnd && other.myStart < myEnd) {
            return true;
        }
        return false;
    }

    /** * Returns the "size" of this interval 
     * @return distance between end and start */
    public int length() {
        return myEnd - myStart;
    }

    /** *
     * Return the starting value of this interval 
     * @return starting value of interval 
     */
    public int getStart() {
        return myStart;
    }

    /** *
     * Return the end value of this interval 
     * @return ending value of interval 
     */
    public int getEnd() {
        return myEnd;
    }

    @Override
    public String toString() {
        return "[" + myStart + "," + myEnd + "]
    }

    @Override
    public int hashCode() {
        // default Object.hashCode returns address in memory
        return super.hashCode();
    }

    @Override
    public boolean equals(Object o) {
        // default Object.equals -- same location in memory
        return this == o;
    }

    /** *
     * Return distance from the end of this interval to the beginning 
     * of another. If this interval ends after other one starts, a 
     * negative number will be returned, otherwise a non-negative number 
     * is returned indicating the distance between non-overlapping intervals 
     * @param i is the other interval 
     * @return the distance until interval i starts 
     */
    public int distanceBetween(Interval i) {
        if (overlaps(i)) {
            throw new IllegalArgumentException(this + " overlaps with " + i);
        }
        return i.myStart - myEnd;
    }

    /** * Return the "size" of this interval 
     * @return distance between end and start */
    public String toString() {
        return "[" + myStart + "," + myEnd + "]";
    }

    /** * @return the starting value of this interval 
     * @return starting value of interval */
    public int getStart() {
        return myStart;
    }

    /** * Return the end value of this interval 
     * @return ending value of interval */
    public int getEnd() {
        return myEnd;
    }

    @Override
    public int hashCode() {
        // default Object.hashCode returns address in memory
        return super.hashCode();
    }

    @Override
    public boolean equals(Object o) {
        // default Object.equals -- same location in memory
        return this == o;
    }

    /** *
     * Return distance from the end of this interval to the beginning 
     * of another. If this interval ends after other one starts, a 
     * negative number will be returned, otherwise a non-negative number 
     * is returned indicating the distance between non-overlapping intervals 
     * @param i is the other interval 
     * @return the distance until interval i starts 
     */
    public int distanceBetween(Interval i) {
        if (overlaps(i)) {
            throw new IllegalArgumentException(this + " overlaps with " + i);
        }
        return i.myStart - myEnd;
    }

    /** * Return the starting value of this interval 
     * @return starting value of interval */
    public int getStart() {
        return myStart;
    }

    /** * Return the end value of this interval 
     * @return ending value of interval */
    public int getEnd() {
        return myEnd;
    }

    @Override
    public int hashCode() {
        // default Object.hashCode returns address in memory
        return super.hashCode();
    }

    @Override
    public boolean equals(Object o) {
        // default Object.equals -- same location in memory
        return this == o;
    }

}
Part C (4 points)
Explain the output of the last two print statements in the code above, in particular explain why the value of c.equals(d) is false and d.equals(d) is true.

Part D (4 points)
The first statement below prints 4, the second generates an exception. Explain both.

```java
System.out.println(new Interval(3,7).
    distanceBetween(new Interval(11,12)));
System.out.println(new Interval(3,7).
    distanceBetween(new Interval(6,12)));
```

(continued)
Part E (6 points)

Write the static `Interval` method `merge` that creates and returns a new `Interval` object that is the union or `merge` of two overlapping intervals. If the `Interval` parameters do not overlap the method should return `null`. See the example code and output.

```java
Interval b = new Interval(10, 12);
Interval c = new Interval(11, 15);
Interval d = new Interval(14, 28);

System.out.printf("%s merge with %s = %s\n", b, c, Interval.merge(b, c));
System.out.printf("%s merge with %s = %s\n", b, c, Interval.merge(c, b));
System.out.printf("%s merge with %s = %s\n", b, d, Interval.merge(b, d));
```

The output of executing the code above is

```
[10,12] merge with [11,15] = [10,15]
[10,12] merge with [11,15] = [10,15]
[10,12] merge with [14,28] = null
```

Complete `Interval.merge` below:

```java
public static Interval merge(Interval a,
    Interval b) {

```

(continued)
PROBLEM 5 :  (Biped, Bimodal (21 points))

The method \texttt{modal} below correctly returns the string that is the \textit{mode} of its parameter \texttt{list}, that is the string that occurs maximally often in \texttt{list}. (If there are ties for maximally often one such maximally occurring string is returned.) Questions follow the code.

```java
public String modal(String[] list) {
    String smax = "";
    for(String s : list) {
        int sf = Collections.frequency(Arrays.asList(list), s);
        int max = Collections.frequency(Arrays.asList(list), smax);
        if (sf > max) smax = s;
    }
    return smax;
}
```

\textbf{Part A (9 points)}

Three students are analyzing the runtime of the code above in terms of \(N\), the number of elements in parameter \texttt{list}.

- A student named Chris says that the code is \(O(N)\) because the for-loop executes \(N\) times, so the runtime is \(O(N)\).

- A student named Pat says that the code is \(O(N^2)\) because while it’s true that the for-loop executes \(N\) times, the code \textit{inside} the loop is \(O(N)\) so that the total runtime is \(N \times N\) which is \(O(N^2)\).

- A student named Leslie says that the code is \(O(N^3)\) because while it’s true that the for-loop executes \(N\) times, there are two statements \textit{inside} the loop that are each \(O(N)\), so that the total runtime is \(N \times N \times N\) which is \(O(N^3)\).

Only one student can be right. Explain three things: which answer is right, and why. Also explain why the other two answers are each wrong.
A student named Gerry claims that regardless of what the runtime analysis shows for the code above it would be much faster if written using a `HashMap<String, Integer>` that keeps track of the number of times each String in parameter `list` occurs.

The complete code for a method using a `HashMap` is on the next page. Gerry uses the code below to return the modal element after storing the correct count in the map for how many times each string in `list` occurs, e.g., so that `map(key)` is the value of how many times `key` occurs in `list`. This code works correctly.

```java
int max = Collections.max(map.values()); \ 1
for(String s : map.keySet()) { \ 2
    if (map.get(s) == max) return s; \ 3
}
```

Explain in words each of the three lines in the code above that are labeled in terms of what the line does and how it helps in returning the correct modal value (the string that occurs most in `list`).
String modal(String[] list) {
    HashMap<String,Integer> map = new HashMap<String,Integer>();
    // add code here to fill map

    int max = Collections.max(map.values());
    for(String s : map.keySet()) {
        if (map.get(s) == max) return s;
    }
    return "";
}

Part D (2 points)
The statement return "" as the last line in the code above is necessary. If it is removed, for example, Eclipse will put a red-X on the method signature for modal with the explanation “this method must return a result of type String”.

Explain why the return statement above is never actually executed but why Eclipse and the Java compiler generate the error message so that the return statement is needed.