

Test 1 : Compsci 201

Owen Astrachan

October 11, 2019

Name: _____ Answer Key for Midterm

NetID/Login: _____

Community standard acknowledgment (signature) _____

(1 point for name, netid)

	value	grade
Problem 1	20 pts.	
Problem 2	8 pts.	
Problem 3	12 pts.	
Problem 4	34 pts.	
TOTAL:	75 pts.	

This test has 11 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 : *(IO, IO, it's off to work (20 points))***Part A: 12 points**

What is printed by each `System.out.printf` statement? There are four statements, each prints three values. Write the values below each `printf` statement.

```
String s = new String("DUKE");
String t = s;
String u = t.toLowerCase();
```

```
System.out.printf("%s \t %s\t %s\n",s, t, u);
```

DUKE DUKE duke

```
t = t + t;
s = "ROCKS";
```

```
System.out.printf("%s \t %s\t %s\n",s, t, u);
```

ROCKS DUKEDUKE duke

```
String[] a = {"one", "two", "three", "four"};
String[] b = Arrays.copyOf(a,a.length);
String[] c = a;
```

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

one one one

```
a[3] = "go";
```

```
System.out.printf("%s \t %s \t %s \n",a[3],b[3],c[3]);
```

go four go

Part B: 8 points

Write one of: True, False, or cannot be determined. Explain/Justify each answer

- If two string variables `s` and `t` have different hash code values, that is `s.hashCode() != t.hashCode()`, then what is the value of `s.equals(t)`?

False, two strings that are equal must have the same hashCode.

- If two string variables `s` and `t` have `s.equals(t)` is true; what is the value of `s == t`?

Cannot be determined. The variables `s` and `t` may point to/reference the same string, so `s == t`. But it's also possible that they point to different strings that have the same characters in which case `s != t`

- If two string variables `s` and `t` have `s.equals(t)` is false; what is the value of `s.hashCode() == t.hashCode()`?

Cannot be determined. Two different strings can have the same hashcodes, e.g., when there are collisions as we saw in class with "snitz" and "unprecludable" having the same hashcodes. But for many cases of unequal Strings the hashcodes will be different.

- If two string variables `s` and `t` have `s.equals(t)` is true; and `s.hashCode() == 7`; what is the value of `t.hashCode() == 7`?

true. equal strings have the same hashCode, so if `s` has 7 as hashCode then so does `t`

PROBLEM 2 : (*I Sing Electric (8 points)*)

The code below is the implementing code from a version of the `CelestialBody` class that passes all tests for the method `calcNetForceExertedByY`.

```

118 public double calcNetForceExertedByY(CelestialBody[] bodies) {
119     double sum = 0.0;
120     for(CelestialBody b : bodies) {
121         if (b != this) {
122             sum += calcForceExertedByY(b);
123         }
124     }
125     return sum;
126 }

```

Part A (4 points)

The `CelestialBody` class has no overloaded operator for `.equals()` — if the expression `(b != this)` in the code above is replaced by `(! b.equals(this))` will the code still pass all tests? Explain.

Yes, tests pass. Since there is no overloaded `.equals`, the default `.equals` will be used. The default `.equals` simply uses `==` to compare two objects, so `! b.equals(this)` is the same as `b != this`

Part B (4 points)

The code below is from the simulation code in `NBody.java` and illustrates *the call* of method `calcNetForceExertedByY` shown above. Which of the three expressions from this calling code (line 98) corresponds to `this` in the implementing code (line 121 above):

Explain your answer in few words.

1. `bodies[k]` bodies[k] — this is the object on which `calcNetForceExertedByY` is called as seen in the code since `bodies[k]` is to the left of the dot/.
2. `bodies`
3. `yforces[k]` So `bodies[k]` is this in the code.

```

94 double[] xforces = new double[bodies.length];
95 double[] yforces = new double[bodies.length];
96 for(int k=0; k < bodies.length; k++) {
97     xforces[k] = bodies[k].calcNetForceExertedByX(bodies);
98     yforces[k] = bodies[k].calcNetForceExertedByY(bodies);
99 }

```

PROBLEM 3 : (*Oh Oh Oh (12 points)*)

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 $1 + 2 + \dots + 99 = (98 \times 99)/2$. Note that even if the return statement was `return sum*2` that the value returned would still be $O(n^2)$.

```
public int stuff(int n){
    int sum = 0;
    for(int k=0; k < n; k++){
        sum += k;
    }
    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.** Your answers are for the entire method, justification can include discussing lines/loops in each method.

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 answer.*

```
public int evaluate(int n){
    int sum = 0;
    for(int k=0; k < n; k++){
        for(int j=0; j < k; j++) {
            sum += 100;
        }
    }
    for(int k=0; k < n; k++){
        sum += 100;
    }
    return sum;
}
```

Runtime is $O(n^2)$ — the first k loop iterates n times, the inner loop iterates k times, so a total of $1+2+3+\dots+n-1$. This is $(n-1)*n/2$ which is $O(n^2)$

The second loop is $O(n)$ and together this is $O(n^2)$

Value returned is $O(n^2)$ — the value returned is $100+100+100 \dots$ done n^2 times, this is essentially $100*n^2$ and this is $O(n^2)$

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. If it helps, you can assume n is a power of 2

```
public int calculate(int n){
    int sum = 0;
    for(int k=1; k < n; k *= 2){
        for(int j=0; j < n; j += 1){
            sum += 1;
        }
    }
    return sum;
}
```

Outer k-loop iterates $\log n$ times since $k = 2, 4, 8, \dots$ until reaching n . This is $\log_2(n)$ or $O(\log n)$

On each iteration the inner loop iterates n times so this is $n+n+n+\dots+n$ done $\log(n)$ times or $O(n \log n)$

The problem indicates the value returned is the same.

Part C (4 points)

Consider the method `stuff` from the beginning of this problem, reproduced below. Recall that the runtime complexity is $O(n)$ and the value returned is $O(n^2)$.

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

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

```
int x = stuff(stuff(n-10)); // big-Oh for runtime and value returned
int y = stuff(stuff(n*n)); // big-Oh for runtime and value returned
```

`stuff(n-10)` is $O(n^2)$ since value of `stuff(n)` is $O(n^2)$. Thus the question is: what is runtime and value returned by call `stuff(n^2)`. Since runtime of `stuff(X)` is $O(X)$, we have runtime is $O(n^2)$ and value is $O(n^4)$

`stuff(n*n)` is $O(n^4)$ since value of `stuff(n)` is $O(n^2)$. Thus the question is: what is runtime and value returned by call `stuff(n^4)`. Since runtime of `stuff(X)` is $O(X)$, we have runtime is $O(n^4)$ and value is $O(n^8)$

PROBLEM 4 : (*Tilt Muse (34 points)*)

A *multiset* of strings can contain more than one occurrence of a string. The *multiplicity* of a string in a multiset is the number of times the string occurs. The size of a multiset is the number of different strings it contains. Consider this code example and the output that follows.

```

112     Multiset s = new Multiset();
113     s.add("a");
114     s.add("a");
115     s.add("a");
116     s.add("b");
117     s.add("c");
118     s.add("c");
119     System.out.printf("%s\t%d\n",s,s.size());
120
121     for(String ss : new String[]{"a","b","c","d","e"}){
122         System.out.printf("%s\t%d\n",ss,s.get(ss));
123     }

```

The output shows that in the Multiset `s` the String "a" has multiplicity three since it was added three times to the set and the set's size is three since there are three unique strings: "a", "b", "c". Note that each element of the set is printed with its multiplicity so that "b" occurs once and "c" two times in the set. The code also shows how the `Multiset.get` method works to return the multiplicity of a string.

```

a:3,b:1,c:2 3
a   3
b   1
c   2
d   0
e   0

```

The last exam page contains Java code for a `Multiset` class with some methods missing that you'll complete as part of this question.

Part A (4 points)

Methods `toString` and `equals` have an `@Override` annotation in the code, but methods `size` and `add` do not. Explain both why the error message *does not override method in superclass* is generated if `@Override` is added before `size`, and not generated when added before `equals`.

The `Object` class has both `.equals` and `.toString`, so every class including `Multiset` inherits these, and if changing the default can use `@Override`

There is `.size` method and no `.add` method in `Object`, so `Multiset` doesn't inherit these since `Multiset` doesn't extend any class other than `Object`. Thus there is no `.size` to `@Override`, for example.

Part B (6 points)

A multiset **a** contains another multiset **b** if every element in **b** occurs in **a** with the same or greater multiplicity. In other words, if some element in **b** occurs more often in **b** than in **a**, then **a** does **not** contain **b**. The code below shows examples of how `contains` is called – the output from this code follows. When the code runs, Multiset **a** already has values as seen when it is printed.

```
37 System.out.println(a);|
38 Multiset d = new Multiset(a);
39 System.out.println(a.equals(d));
40 System.out.printf("%s contains %s: %s\n",a,d,a.contains(d));
41
42 a.add("a");
43 System.out.printf("%s contains %s: %s\n",a,d,a.contains(d));
44 d.add("a"); d.add("a");
45 System.out.printf("%s contains %s: %s\n",a,d,a.contains(d));
```

```
a:2,b:3,c:1
true
a:2,b:3,c:1 contains a:2,b:3,c:1: true
a:3,b:3,c:1 contains a:2,b:3,c:1: true
a:3,b:3,c:1 contains a:4,b:3,c:1: false
```

Complete the implementation of the method `Multiset.contains`.

```
public boolean contains(Multiset a) {
    for (String key : a.keySet()) {
        if (get(key) < a.get(key)) return false;
    }
    return true;
}
```

It's also possible to do this in one line, though that was not expected:

```
return this.equals(union(this,a))
```

```
}
```


Part C (4 points)

The `Multiset` class has two constructors. Code shown earlier/above in this problem illustrates both constructors being called. Briefly, explain why the code below will always print `true`, regardless of how many times `a.add(...)` is called. You must refer to the constructor used in creating `Multiset b` and to the `.equals` method in your response.

```
Multiset a = new Multiset();

// call a.add(..) multiple times

Multiset b = new Multiset(a);
System.out.println(a.equals(b));
```

The copy constructor, called to create the `Multiset` pointed to by `b`, simply creates a new `HashMap` that is equal to the `HashMap` being passed to the constructor (see line 11 of code)

The `.equals` method simply uses `myMap.equals(other.myMap)` — and these are the same after the constructor executes.

Part D (4 points)

The *cardinality* of a `Multiset` is the total of all multiplicities of strings in the multiset (the total number of strings including duplicates since a multiset can contain multiple occurrences of a string). The code below and its output show two calls of the method `cardinality` for two different multisets. Note that the cardinality is the sum of the multiplicities for each set.

```
35 System.out.printf("%s --- %d\n",b, b.cardinality());
36 System.out.printf("%s --- %d\n",a, a.cardinality());
```

```
a:1,b:2,d:4 --- 7
a:2,b:3,c:1 --- 6
```

If `a.contains(b)` is true, is `b.cardinality() <= a.cardinality()` always true? Explain.

Yes, it's always true. Every element in `b` must occur at least that many times in `a` since `a.contains(b)`.

This means `a.cardinality()` is greater than or equal to `b.cardinality()`

If `a.contains(b)` is false, is `b.cardinality() <= a.cardinality()` always false? Explain.

Not always false if `a` is "x:3,y:4" and `b` is "x:8" then `a.contains(b)` is false, but `b.cardinality() > a.cardinality()`. But if `b` is "x:5" `a.contains(b)` is still false, but `b.cardinality() < a.cardinality()`

Part E (6 points)

Implement the method `cardinality`

```
public long cardinality() {
    long sum = 0;

    for (int i : myMap.values()) {
        s += i;
    }

    for (String key: keySet()) {
        s += get(key);
    }

    return sum;
}
```

Part F (2 points)

The instance variable `myMap` stores integer values as the multiplicity of each string in a `Multiset`, but the return type of `cardinality` is `long`. Briefly explain why `long` is useful as the return type even though integer values are stored.

A `Multiset` could contain many keys all of which have a cardinality of close to `Integer.MAX_VALUE`.

Adding these together would result in integer overflow, but adding them to a `long` would NOT result in overflow.

Part G (4 points)

Currently, client code must call the method `.add("duke")` ten times to add ten occurrences of "duke" to a multiset.

```
Multiset a = new Multiset();
for(int k=0; k < 10; k++) a.add("duke");
```

Suppose a new, overloaded method `add` method is included in the `Multiset` class so that the call `a.add("duke",10)` works to add "duke" ten times. Complete the implementation below

```
public void add(String key, int occurrences) {

    for(int k=0; k < occurrences; k++) {
        add(key);
    }

    myMap.putIfAbsent(key,0);
    myMap.put(key, myMap.get(key) + occurrences);

}
```

Part H (4 points)

If both constructors use `new TreeMap` instead of `new HashMap` then two statements are true:

- all `Multiset` methods will work correctly
- it's possible that the value returned by `toString` might change as far as the order of the keys in the multiset that are part of the string returned.

Briefly explain both of these statements.

Since both `TreeMap` and `HashMap` implement the `Map` interface, the methods they have all work correctly (though performance may be different).

The code in `toString` loops over the keys in the map and for `TreeSet` this results in keys being visited in alphabetical order. The order in which keys are visited in a `HashMap` is unknown/based on the internal collections of buckets/lockers.

```
1: import java.util.*;
2:
3: public class Multiset {
4:     private Map<String, Integer> myMap;
5:
6:     public Multiset() {
7:         myMap = new HashMap<>();
8:     }
9:
10:    public Multiset(Multiset a) {
11:        myMap = new HashMap<>(a.myMap);
12:    }
13:
14:    @Override
15:    public boolean equals(Object o) {
16:        if (o == null || !(o instanceof Multiset)) {
17:            return false;
18:        }
19:        Multiset other = (Multiset) o;
20:        return myMap.equals(other.myMap);
21:    }
22:
23:    public static Multiset union(Multiset a, Multiset b) {
24:        Multiset ret = new Multiset();
25:        Set<String> keys = new HashSet(a.keySet());
26:        keys.addAll(b.keySet());
27:        for (String key : keys) {
28:            int occurrences = Math.max(a.get(key), b.get(key));
29:            ret.myMap.put(key, occurrences);
30:        }
31:        return ret;
32:    }
33:
34:    public void addAll(Multiset b) {
35:        for (String key : b.keySet()) {
36:            myMap.put(key, get(key) + b.get(key));
37:        }
38:    }
39:
40:    /**
41:     * returns total number of all elements represented by multiset
42:     * @return cardinality of this multiset
43:     */
44:    public long cardinality() {
45:        long s = 0;
46:        // code not shown
47:        return s;
48:    }
49:
50:    /**
51:     * Add a value to the multiset so that it occurs once more
52:     * than previous number of occurrences
53:     *
54:     * @param key is string added to multiset
55:     */
56:    public void add(String key) {
57:        myMap.putIfAbsent(key, 0);
58:        myMap.put(key, myMap.get(key) + 1);
59:    }
60:
61:    public void add(String key, int occurrences) {
62:        // code not shown
63:    }
```

```
64:
65:    @Override
66:    public String toString() {
67:        String ret = "";
68:        for (String key : myMap.keySet()) {
69:            ret += String.format("%s:%d,", key, myMap.get(key));
70:        }
71:        return ret.substring(0, ret.length() - 1); // remove last comma
72:    }
73:
74:    /**
75:     * Returns number of distinct strings in this multiset.
76:     * @return the size of this multiset
77:     */
78:    public int size() {
79:        return myMap.size();
80:    }
81:
82:    /**
83:     * Returns number of occurrences of key in this multiset.
84:     * @param key is string
85:     * @return number of occurrences of key in this multiset
86:     */
87:    public int get(String key) {
88:        if (!myMap.containsKey(key)) {
89:            return 0;
90:        }
91:        return myMap.get(key);
92:    }
93:
94:    /**
95:     * Returns a set of distinct strings in this multiset
96:     * @return set of strings in this multiset
97:     */
98:    public Set<String> keySet() {
99:        return myMap.keySet();
100:    }
101:
102:    /**
103:     * Returns true if a contained in this multiset and false otherwise
104:     * @param a is a multiset
105:     * @return true if and only if a contained in this multiset
106:     */
107:    public boolean contains(Multiset a) {
108:        // code not shown
109:        return true;
110:    }
111:
112: }
```