Midterm Exam 1, Compsci 201
Fall 2022, Duke University
Version A

September 28, 2022

General Directions

Before you begin, **make sure to write your name and netid on the exam, read the Duke community statement, and sign indicating your understanding and agreement to these directions**. You are encouraged to write your netid on every page of the exam where indicated in the event that pages become separated during scanning.

Every question on the exam will have a box indicating where you should write your answer. Answers outside of the corresponding box will not be graded.

For all problems you should assume that any necessary libraries (for example, from `java.util`) are imported. Where relevant, give the most tight analysis you can using big O notation. For example, if the running time is $O(N)$ then answering $O(N^2)$, while technically true, will not earn full credit.

You may not communicate with anyone while completing this exam. You may not discuss this exam with anyone else on the day of the exam. You may not access any electronic devices (including but not limited to phones, smartwatches, laptops, etc.) during the exam period. If you need to leave the exam room during the exam period, you should not communicate with anyone and should not access any electronic devices. You are allowed one 8.5x11 in. reference sheet.

Duke Community Standard

Duke University is a community dedicated to scholarship, leadership, and service and to the principles of honesty, fairness, respect, and accountability. Citizens of this community commit to reflect upon and uphold these principles in all academic and nonacademic endeavors, and to protect and promote a culture of integrity.

To uphold the Duke Community Standard:

- I will not lie, cheat, or steal in my academic endeavors;
- I will conduct myself honorably in all my endeavors; and
- I will act if the Standard is compromised.

Print Name. Solution Key **NetID.** _____

Signature. _____ **Date.** _____

```
1  public class Circle {
2      static final double PI = 3.14;
3      private int x;
4      private int y;
5      public int radius;
6
7      public Circle(int x, int y, int r) {
8          this.x = x; this.y = y;
9          this.radius = r;
10     }
11
12     public static double distance(int x1, int y1, int x2, int y2) {
13         int dx = x1 - x2; int dy = y1 - y2;
14         return Math.sqrt(Math.pow(dx, 2) + Math.pow(dy, 2));
15     }
16
17     public boolean isInside(Circle c) {
18         double dist = distance(x, y, c.x, c.y);
19         if (dist + radius <= c.radius) {
20             return true;
21         }
22         return false;
23     }
24 }
```

Figure 1: Circle class

1. (4 points). Consider the `Circle` class shown in Figure 1. What are the **instance variables** of the class, the variables such that every object of the class can have different values for those variables? Can any of these instance variables can be directly accessed (without using a method) by code outside of `Circle.java`? If so, which one(s)?

Answer: The instance variables are x, y, and radius. Only radius can be directly accessed by code outside the class.

2. (4 points). Consider the `isInside()` method of the `Circle` class. For each of the following proposed changes to the code, state and briefly explain whether the proposed change would affect the behavior of the code.

A. Change line 18 to `double dist = distance(c.x, c.y, x, y);`

B. Change line 19 to `if (dist + radius <= this.radius) {`

Answer: A. No change, the distance method calculation does not depend on order of x1 vs. x2 or y1 vs. y2 because $(x1-x2)^2 = (x2-x1)^2$ and similarly for y.

B. Changes. c refers to the parameter whereas this refers to the Point object on which the method was called.

```
1         Set<String> myStrings = new HashSet<>();
2         Set<Circle> myCircles = new HashSet<>();
3         String s = new String("Duke");
4         Circle c = new Circle(0, 0, 1);
5         for (int i=0; i<10; i++) {
6             // consider
7             // consider
8             myStrings.add(s);
9             myCircles.add(c);
10        }
11        System.out.println(myStrings.size());
12        System.out.println(myCircles.size());
```

Figure 2: Sets of Strings, Sets of Circles

3. (6 points). Consider the code in Figure 2. Answer the following questions. For each, briefly explain your answer.

A. If you run the code as shown, what will be printed by lines 11 and 12 respectively?

B. Suppose the code on lines 3 and 4 are moved inside the loop to lines 6 and 7 (with the comment `// consider`). Then what would be printed by lines 11 and 12 respectively?

Answer: A. Both print 1. s and c are both created before the loop and have the same hashCode and are equals every time lines 8 and 9 try to add them the set, so they are treated as duplicates after the first iteration.

B. 11 still prints 1 but line 12 prints 10. String hashCode and equals are based on the characters in the string, which are still the same on every iteration. Circle only inherits the default Object hashCode and equals based on memory location, so each new object created in the loop is treated as unique from the others.

```
1  public class DIYArrayList {
2      private int nextOpen;
3      private String[] values;
4      public DIYArrayList() {
5          nextOpen = 0;
6          values = new String[1];
7      }
8
9      public void add(String toAdd) {
10         if (nextOpen >= values.length) {
11             grow(2, 0);
12         }
13         values[nextOpen] = toAdd;
14         nextOpen++;
15     }
16
17     private void grow(int multiplier, int additional) {
18         String[] oldValues = values;
19         values = new String[values.length * multiplier + additional];
20         for (int i=0; i<oldValues.length; i++) {
21             values[i] = oldValues[i];
22         }
23     }
```

Figure 3: ArrayList Growth

4. (6 points). Consider the code in Figure 3. Suppose we `add` n elements to an initially empty `DIYArrayList` one at a time. Answer the following and briefly explain your answers.

A. Suppose the average `add` (out of the n total) takes T_{avg} milliseconds (ms) using the code as shown. About how long should you expect it to take **in the worst case to add a single element**? Answer in terms of T_{avg} and/or n.

B. Suppose it takes T_{tot} milliseconds (ms) **total to add all** n elements using the code as shown. About how long should you expect it to take to add the same n elements to an initially empty `DIYArrayList` if line 11 is changed to `grow(1, 1)`? Answer in terms of T_{tot} and/or n.

Answer: A. Something between $\frac{T_{avg} \cdot n}{2}$ and $T_{avg} \cdot n$, because in the worst case, grow will be called by add when the current array has $n/2$ to n elements, copying all of those elements.

B. Something like $T_{tot} \cdot n$ because, using grow(1,1), the entire previous array will have to be copied on each of the n adds.

```
1     public int calc(int n) {
2         int iters = 0;
3         for (int i=0; i<n; i++) {
4             for (int j=i; j<n; j++) {
5                 iters++;
6             }
7         }
8         for (int k=0; k<2*n; k++) {
9             iters++;
10        }
11        return iters;
12    }
```

Figure 4: calc method

5. (4 points). Consider the code shown in Figure 4. What is the Big O runtime complexity of `calc()` as a function of the parameter n? Briefly explain your answer referencing the code.

Answer: $O(n^2)$. Line 2 is $O(1)$. The loop on line 3 iterates n times, and on the ith iteration, the nested loop on line 4 iterates $n-i$ times, for a total of $n + (n-1) + (n-2) + \ldots + 3 + 2 + 1 \approx \frac{n^2}{2}$ iterations, and the loop body on line 5 is $O(1)$. The other loop is only $O(n)$ so the nested $O(n^2)$ loop dominates the runtime complexity.

```
1 public String foo(int k) {
2     String s = "";
3     for (int i=0; i<k; i++) {
4         s = s + "Duke";
5     }
6     return s;
7 }
```

```
1 public int bar(int m) {
2     int result = 0;
3     for (int i=0; i<m; i++) {
4         result += m;
5     }
6     return result;
7 }
```

```
1 public String foobar(int n) {
2     return foo(bar(n));
3 }
```

Figure 5: foo, bar, and foobar methods

6. (4 points). Consider the code shown in Figure 5. What is the Big O runtime complexity of `foobar()` as a function of the parameter n? Briefly explain your answer referencing the code.

Answer: foo(k) is:
$4 + 8 + 12 + 16 + \ldots + 4k$
$= 4(1 + 2 + 3 + \ldots + k)$
is O(k^2)
adding up all concatenations and remembering strings are immutable.

[redacted]

bar(m) has linear runtime complexity but returns the value m^2.

foo(bar(n)) = foo(n^2)
has runtime complexity
$O((n^2)^2) = O(n^4)$
final answer

```
1     public List<Integer> inAll(List<List<Integer>> lists) {
2         List<Integer> inAll = new ArrayList<>();
3         for (int val : lists.get(0)) {
4             boolean valid = true;
5             for (int i=1; i<lists.size(); i++) {
6                 List<Integer> nextList = lists.get(i);
7                 if (EXPR1) {
8                     valid = false;
9                 }
10            }
11            if (valid && !inAll.contains(val)) {
12                EXPR2;
13            }
14        }
15        return inAll;
16    }
```

Figure 6: inAll method

7. (4 points). Consider the code shown in Figure 6. The `inAll` method takes as input a `List` of `Lists` of `Integers` and should return a `List` of `Integers` containing only values that appear in every input `List`. There should be no duplicates in the returned `List`.

Two pieces of code labeled `EXPR1` and `EXPR2` in the code on lines 7 and 12 respectively are missing. What should these be so that the method works correctly? You do not need to explain your answer.

- `EXPR1`: `!nextList.contains(val)`
- `EXPR2`: `inAll.add(val)`

8. (4 points). Again consider the code shown in Figure 6. Suppose the input `lists` contains m `Lists`, each of which contains n `Integers`. Further suppose that the the missing expressions are correctly implemented, the runtime complexity of line 7 / `EXPR1` is $O(n)$, and the runtime complexity of line 12 / `EXPR2` is $O(1)$.

What is the big O runtime complexity of `inAll()`? Briefly explain your answer.

Answer: $O(n^2m)$. The loop on line 3 loops over n values. The nested inner loop on line 5 iterates over $m-1$ lists. The problem assumes line 7, which runs on each of these $n(m-1)$ iterations, is $O(n)$, yielding $O(n^2m)$.

Assumes Lists are ArrayLists

```
1    public Map<String, Integer> count(Map<String, List<String>>
     destinations) {
2        Map<String, Integer> counts = new HashMap<>();
3        for (String p : destinations.keySet()) {
4            for (String s : EXPR1) {
5                counts.putIfAbsent(EXPR2, 0);
6                counts.put(EXPR3, EXPR4);
7            }
8        }
9        return counts;
10   }
```

Figure 7: count method

9. (8 points). The count method outlined above takes as input a `Map<String, List<String>> destinations` parameter and should return a `Map<String, Integer>` that counts the total number of occurrences of every String across all of the List values in `destinations`. For example, the tables below show an example of an input and the corresponding `Map` (in no particular order) that should be returned.

Key	Value
"Al"	["los angeles", "new york"]
"Xi"	["reno", "los angeles"]
"Jen"	["chicago", "austin"]

(a) Example destinations input

$\longrightarrow$

Key	Value
"austin"	1
"reno"	1
"los angeles"	2
"chicago"	1
"new york"	1

(b) Example of correct return Map

What should the missing expressions, `EXPR1`, `EXPR2`, `EXPR3`, and `EXPR4` be so that the code works as intended? You do not need to explain your answers.

- `EXPR1`: destinations.get(p)
- `EXPR2`: s
- `EXPR3`: s
- `EXPR4`: counts.get(s)+1

10. (4 points). Suppose that `destinations` has m key-value pairs and each list value has n Strings of at most a constant length. In the above example, m would be 3 and n would be 2. What is the big O runtime complexity of the `count` method as a function of m and n? Briefly justify your answer referencing the code.

Answer: O(mn). The loop on line 3 iterates over m keys, and for each, the nested loop on line 4 iterates over n elements of the value list. The HashMap operations on lines 5 and 6 are treated as O(1) for String keys.