Midterm 1: CompSci 201
Form A

Prof. Alex Steiger

February 12, 2024

General Directions

You should have a separate answer sheet. Before you begin, **make sure to indicate your name and NetID**, and **verify that your form matches the answer sheet**. Do this in addition to signing this exam copy.

This exam contains multiple choice questions of equal weight. Some questions have as few as two choices for answers, but every problem has five bubbles on the sheet. **Make sure you bubble answers appropriately.** You are welcome to use the exam itself for scratch work, but **only your answer sheet will be graded**. If you erase to change an answer, be sure to erase completely.

For all problems you should assume that any necessary libraries (for example, from 'java.util') are imported.

You may not communicate with anyone while completing this 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 that you bring with you.

You will have **60 minutes** to complete the exam. When you are finished, **turn in your answer sheet, your signed exam copy, and your 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. _____ **NetID.** _____

Signature. _____ **Date.** _____

This Exam is Form A, please mark your answer sheet accordingly.

```
1  public class Student {
2    String name;
3    String[] majors;
4
5    public Student() {
6      this.name = new String("unnamed");
7      this.majors = new String[] { new String("undeclared") };
8    }
9
10   public Student(String givenName, String[] majors) {
11     this.name = givenName;
12     this.majors = majors;
13   }
14
15   public static void main(String[] args) {
16     String name = "cora";
17     String[] majors = new String[] { "math", "dance" };
18     Student c = new Student(name, majors);
19     Student u = new Student();
20   }
21 }
```

For the next two problems, consider the `Student` class defined above.

PROBLEM 1: In which of lines 18 and 19 is memory allocated for `Student` objects?

A. No memory is allocated in line 18 nor line 19

B. Line 18 only

C. Line 19 only

D. Both lines 18 and 19

PROBLEM 2: Which of the following modifications affects the behavior of code using this class?

A. Removing "`this.`" from line 6

B. Removing "`this.`" from line 7

C. Removing "`this.`" from line 11

D. Removing "`this.`" from line 12

E. None of the modifications affect the behavior

PROBLEM 3: Private instance variables can only be assigned inside constructor methods.

A. True

B. False

PROBLEM 4: Calling the `.equals` method on an object of a class that has not overriden it will cause a `NoSuchMethodException`.

A. True

B. False

PROBLEM 5: Classes are not required to have non-static methods.

A. True

B. False

```
1   public static boolean chocula(ArrayList<String> words) {
2         int count1 = 0;
3         int count2 = 0;
4         HashSet<String> set = new HashSet<>(words);
5         for (String s : words) {
6           if (set.contains(s)) {
7             count1++;
8           }
9         }
10        for (String s : set) {
11          if (words.contains(s)) {
12            count2++;
13          }
14        }
15        return count1 <= count2;
16    }
```

PROBLEM 6: Which of the following statement correctly describes the behavior of the `chocula` method above?

A. It always returns `true`

B. It always returns `false`

C. For some inputs it returns `true` and for some inputs it returns `false`

size	methodA	methodB
1000	506	0.298
2000	520	0.306
4000	572	0.354
8000	772	0.418
16000	1556	0.546
32000	4660	0.802
64000	17012	1.314
128000	66292	2.538
256000	263156	5.186

The table above shows timings (in milliseconds) for calling two methods with `ArrayList` parameters with sizes ranging from 1,000 elements to 256,000 elements. Consider the table in the following two questions.

PROBLEM 7:

Based on these empirical timings, which best characterizes the asymptotic runtime complexity of `methodA` when called with a parameter containing N values?

A. $O(1)$

B. $O(\log N)$

C. $O(N)$

D. $O(N^2)$

E. $O(N^3)$

PROBLEM 8:

Based on these empirical timings, which best characterizes the asymptotic runtime complexity of `methodB` when called with a parameter containing N values?

A. $O(1)$

B. $O(\log N)$

C. $O(N)$

D. $O(N^2)$

E. $O(N^3)$

The method `highFreq` returns an `int` equal to the number of distinct words in the given sentence (words separated by spaces) `String input` that appear at least the given `int freq` times. **This method is only intended to work properly when `input` contains only lowercase letters and spaces and `freq` is positive (at least 1)**.

For example, suppose `input = "a is a vowel but b is not a vowel"`. Then `highFreq(input, 1) == 6` since every distinct word in `input` naturally appear `freq`=1 times or more. As another example, `highFreq(input, 2) == 3` since the three words `"a"`, `"is"`, and `"vowel"` each appear at least `freq`=2 times and the other words, `"but"`, `"b"`, and `"not"`, appear fewer than `freq`=2 times.

```
1   public static int highFreq(String input, int freq) {
2       int count = 0;
3       String[] wArray = input.split(" ");
4       ArrayList<String> words = new ArrayList<>(Arrays.asList(wArray));
5       HashMap<String, Integer> map = new HashMap<>();
6       for (String word : words) {
7           System.out.println(map);
8           if (!map.containsKey(word)) {
9               /* EXPR_1 */
10          }
11          map.put(word, map.get(word)+1);
12          if ( /* EXPR_2 */ ) {
13              count++;
14          }
15      }
16      return count;
17  }
```

PROBLEM 9: Which of the following can replace `EXPR_1` so that `highFreq` works as intended (specifically, when `input` only consists of lowercase letters and spaces and `freq` is positive).

A. `map.put(new String[]{ word }, new Integer[]{ 0 });`

B. `map.put(0, word);`

C. `map.put(Integer.toString(0), word);`

D. `map.put(word, 0);`

E. `map.put(word, new ArrayList<>(0));`

PROBLEM 10: Which one of the following can replace `EXPR_2` so that `highFreq(input,freq)` works as intended (specifically, when `input` only consists of lowercase letters and spaces and `freq` is positive).

A. `true`

B. `map.get(word) >= freq`

C. `map.get(Integer.toString(freq)) >= word.length()`

D. `map.get(word) == freq`

E. `map.get(Integer.toString(freq)) >= words.length`

```java
public class DIYHashSet {
    public static final int M = 200; // initial number of buckets
    private List<String>[] buckets;

    public DIYHashSet() {
        buckets = new ArrayList[M];
        for (List<String> bucket : buckets) {
            bucket = new ArrayList<>();
        }
    }

    private int calcHash(String s) {
        return Math.abs(s.hashCode()) % buckets.length;
    }

    public void add(String s) {
        if (!buckets[calcHash(s)].contains(s)) {
            buckets[calcHash(s)].add(s);
        }
    }

    public boolean contains(String s) {
        return buckets[calcHash(s)].contains(s);
    }

    public void grow() {
        List<String>[] oldBuckets = buckets;
        buckets = new ArrayList[buckets.length*2];
        for (List<String> bucket : oldBuckets) {
            for (String s : bucket) {
                add(s);
            }
        }
    }
}
```

Figure 1: DIYHashSet

For the next three problems, consider the `DIYHashSet` class defined on the previous page, which implements a Set ADT using a hash table for storing Strings, whose size is M buckets.

PROBLEM 11: Suppose you have an array `String[] strs` of length $N < M$ such that every String has at most 40 characters and no two Strings are equal to each other. Consider adding the N strings into an initially empty `DIYHashSet`, one by one. Then, under the simple uniform hashing assumption (SUHA), which of the following best describes the expected size of any bucket?

A. 1

B. N/M

C. N

D. N^2

E. MN

PROBLEM 12: Suppose you have an array `String[] strs` of length $N < M$ such that every String has **exactly** 40 characters and no two Strings are equal to each other. Furthermore, suppose we modify the `calcHash(String s)` method to return `s.length() % buckets.length` instead of its original return value. After inserting the N Strings, one by one, to an initially empty `DIYHashSet`, which of the following best describes the size of the bucket that `strs[0]` is placed in? (Note that this question does not assume SUHA unlike the previous problem.)

A. 1

B. N/M

C. N

D. N^2

E. MN

PROBLEM 13: Suppose you have an array `String[] strs` of length N (the Strings have any number of characters and the `calcHash(String s)` method is as originally stated in `DIYHashSet`). Consider adding each String in `strs` to an empty `DIYHashSet mySet` one at a time and then calling `mySet.grow()` once. What would be returned by calling `mySet.contains(strs[0])` afterwards?

A. True

B. False

C. This code does not always return because it may cause a `IndexOutOfBounds` error

D. This code always returns but the answer cannot be determined from the information given (e.g., it changes depending on unknown values of N, `strs[0].hashCode()`, etc.)

Suppose you have a `HashMap<String> myMap` implemented using a hash table as described in lecture. That is, `myMap` is represented by an array of `B` of buckets (lists), for an unspecified positive integer `B`, and the hash (bucket index) for any given `String key` is computed using the following method:

```
public static int calcHash(String key) {
      return Math.abs(key.hashCode()) % B; // B is the number of buckets
  }
```

For each statement, choose between True, False, or cannot be determined from what is given. If you choose *cannot be determined*, that means for some values of variables `s`, `t` and/or `B`, the statement could be True, but for other values it could be False. If you choose True, then the statement is True for any/all values of `s`, `t`, and `B`, and similarly if you choose False, the statement is False for any/all such values.

PROBLEM 14:

If two String variables `s` and `t` have `s.equals(t)` is **True**, what is the value of `s.hashCode() == t.hashCode()`?

A. True

B. False

C. Cannot be determined from the information given

PROBLEM 15:

If two String variables `s` and `t` have `s.hashcode() == t.hashCode()` is **False**, what is the value of `calcHash(s) == calcHash(t)`?

A. True

B. False

C. Cannot be determined from the information given

PROBLEM 16:

If two String variables `s` and `t` have different hashes (that is, `calcHash(s) == calcHash(t)` is **False**), what is the value of `s.equals(t)`?

A. True

B. False

C. Cannot be determined from the information given

PROBLEM 17:

Which of the following best characterizes the runtime complexity of the call `occlusion(num)` as a function of N, where $N =$ `num`?

```
public int occlusion(int num) {
    int sum = 0;
    for(int i=0; i < num; i++) {
        for(int j=0; j <= num; j++) {
            for(int k=0; k <= num; k++) {
               sum += i*j*k;
            }
        }
    }
    return sum;
 }
```

A. $O(1)$

B. $O(N)$

C. $O(N^2)$

D. $O(N^3)$

E. $O(N^8)$

PROBLEM 18:

Which of the following best characterizes the runtime complexity of the call `immersion(num)` as a function of N, where $N =$ `num`?

```
public int immersion(int num) {
      int sum = 0;
      for(int i=num; i < num*num; i += num) {
          sum += 1;
      }
      return sum;
  }
```

A. $O(1)$

B. $O(\log N)$

C. $O(N)$

D. $O(N^2)$

E. $O(N^3)$

PROBLEM 19:

Which of the following best characterizes the runtime complexity of the call `fission(s)` as a function of N, where $N =$ `s.length()`?

```
public int fission(String s) {
    String t = "";
    while (t.length() < s.length()) {
         t += "zZzZ";
    }
    return t;
 }
```

A. $O(\log N)$

B. $O(N)$

C. $O(N \log N)$

D. $O(N^2)$

E. $O(2^N)$

PROBLEM 20:

Which of the following best characterizes the runtime complexity of the call `eclosion(nums)` as a function of N, where $N =$ `nums.length`? *[Hint: How many iterations of each loop are actually executed? Running through an example with a small input array may be helpful.]*

```
public int eclosion(int[] nums) {
    List<Integer> list = new ArrayList<>();
    for (int i : nums) {
       if (!list.contains(i)) {
          for (int j : nums) {
             if (!list.contains(j)) {
                list.add(j);
             }
          }
       }
    }
    return list.size();
 }
```

A. $O(1)$

B. $O(N)$

C. $O(N^2)$

D. $O(N^3)$

E. $O(N^4)$