Midterm 2: CompSci 201
Form A

Prof. Alex Steiger

March 20, 2024

General Directions

You should have an exam stapled with a separate answer sheet. Remove this answer sheet and verify it is a bubble sheet on the front side and fill-in-the-blank sheet on the back. 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 has **22 problems** which are multiple choice (MC) and fill-in-the-blank problems of equal weight. Some MC problems have as few as two choices for answers, but every problem has five bubbles on the sheet. **Fill in bubble "A" for every fill-in-the-blank problem, then write your actual answer on the appropriate space on the back of the answer sheet.** If you erase to change an answer, be sure to erase completely.

You are welcome to use the exam itself for scratch work, but **only your answer sheet will 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 be considered correct.

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.**

The back of this first page contains code for the `ListNode` class and common recurrences and their solutions which you may find useful.

By taking this exam, you are intending to, and promising to, adhere to the Duke Community Standard.

Print Name. _____ **NetID.** _____

Signature. _____ **Date.** _____

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

Common recurrences, their solutions, and the ListNode class used throughout the test (same as lecture/APTs)

```
T(N) =  T(N/2) + O(1) -> O(log N)
T(N) =  T(N/2) + O(N) -> O(N)
T(N) = 2T(N/2) + O(1) -> O(N)
T(N) = 2T(N/2) + O(N) -> O(N log N)
T(N) =  T(N-1) + O(1) -> O(N)
T(N) =  T(N-1) + O(N) -> O(N^2)
T(N) = 2T(N-1) + O(1) -> O(2^N)
```

```
public class ListNode {
    int info;
    ListNode next;
    ListNode(int val) {
        info = val;
    }
    ListNode(int val, ListNode node) {
       info = val;
       next = node;
   }
```

Throughout the exam, we represent the singly linked lists, for example, as `[2,0,1]`, meaning the input is a reference to a `ListNode` with `info` 2, which points to another `ListNode` with `info` 0, which points to another `ListNode` with `info` 1.

```
1  public static int countNegative(List<Integer> nums) {
2    int count = 0;
3    for (int i = 0; i < nums.size(); i++) {
4      if (nums.get(i) < 0) { count++; }
5    }
6    return count;
7  }
```

Figure 1: countNegative method

For the next two problems, consider the `countNegative` method (shown in Figure 1) above, whose parameter is any instance of class that implements the `java.util.List` interface.

PROBLEM 1: Suppose we call `countNegative` where the input list is a `java.util.ArrayList` with N elements. What is the asymptotic runtime complexity of `countNegative(list)`?

A. $O(1)$

B. $O(\log N)$

C. $O(N)$

D. $O(N^2)$

PROBLEM 2: Suppose we call `countNegative` where the input list is a `java.util.LinkedList` with N elements. What is the asymptotic runtime complexity of `countNegative(list)`?

A. $O(1)$

B. $O(\log N)$

C. $O(N)$

D. $O(N^2)$

Consider the following problem, which is similar to the `shiftAdd` method of the `WordGram` class in P2-Markov except that is defined for linked lists composed of `ListNode` objects instead of an array: Given a linked list **with at least one integer** and `int x`, return a list containing all but the first items in the given list, followed by `last`. The following method `shiftAdd` is missing two expressions. For example, if `list` is `[2,0,1]` and `last` is `7`, then `shiftAdd(list,last)` should return the linked list `[0,1,7]`.

```
1  public static ListNode shiftAdd(ListNode list, int last) {
2    ListNode current = list;
3    while (current.next != null) {
4      current = current.next;
5    }
6    current.next = EXPR_1;
7    return EXPR_2;
8  }
```

Figure 2: shiftAdd method for linked lists

The next three problems reference the `shiftAdd` method in Figure 2 above.

PROBLEM 3: What is the missing statement for `EXPR_1`? Write it in the appropriate fill-in-the blank area of the back of the answer sheet. **Bubble A for this problem on the front of the answer sheet.**

PROBLEM 4: What is the missing statement for `EXPR_2`? Write it in the appropriate fill-in-the blank area of the back of the answer sheet. **Bubble A for this problem on the front of the answer sheet.**

PROBLEM 5: What best describes the asymptotic runtime of the correctly-completed `shiftAdd` in terms of N, the size of the input linked list?

A. $O(1)$

B. $O(\log(N))$

C. $O(N)$

Below is a variant of the `cutAndSplice` method from P3-DNA that has three parameters: a String `dna`, a single character `base`, and a String `splicee`, all of which are composed of chemical bases (characters `'a'`, `'c'`, `'g'` and/or `'t'`) as in P3. The output of this method is a linked list represented by `StrListNode` objects (which are functionally identical to `ListNode` but using Strings instead of ints) such that the concatenation of the Strings in the list is equivalent to replacing every occurrence of `base` with `splicee`. (This method is less general than the one in P3—the part of the DNA sequence to replace must be a single character here (one chemical base), whereas in P3 the part to replace could be any String of chemical bases.)

For example, `cutAndSplice("gattaca", 'a', "ggg")` returns the linked list `["g", "ggg", "t", "t", "ggg", "c", "ggg"]`, and `cutAndSplice("gattaca", 'g', "tac")` returns the linked list `["tac", "a", "t", "t", "a", "c", "a"]`.

```
1     private class StrListNode {
2         String info;
3         StrListNode next;
4         StrListNode(String s) { info = s; }
5         StrListNode(String s, StrListNode node) { info = s; next = node; }
6     }
7
8     public StrListNode cutAndSplice(String dna, char base, String splicee)
      {
9         HashMap<String, String> map = new HashMap<>();
10        map.put(splicee, splicee);
11        StrListNode first = null;
12        StrListNode prev = null;
13        for (char c : dna.toCharArray()) {
14            String currStr = String.valueOf(c);
15            if (c == base) { // replace this base with splicee
16                currStr = splicee;
17            }
18
19            map.putIfAbsent(currStr, currStr);
20            currStr = map.get(currStr); // get ref. to equal String in map
21            StrListNode current = new StrListNode(currStr);
22
23            if (prev == null) {
24                first = current;
25            } else {
26                prev.next = current;
27            }
28            prev = current;
29        }
30        return first;
31    }
```

Figure 3: The cutAndSplice method and StrListNode class

The next two problems reference the `cutAndSplice` method in Figure 3. Consider calling `cutAndSplice(dna,base,splicee)`, and let N be the length of `dna`, let b be the number of occurrences of `base` in `dna`, and let S be the length of `splicee`.

PROBLEM 6: What best describes the number of `StrListNode` objects created during `cutAndSplice(dna,base,splicee)` in terms of N, b, and S?

A. $O(1)$

B. $O(bS)$

C. $O(N)$

D. $O(N + bS)$

E. $O(N + b^2S)$

PROBLEM 7: What best describes the number of distinct Strings pointed to by the `info` variables of the `StrListNode` objects created during `cutAndSplice(dna,base,splicee)` in terms of N, b, and S? (If a String is pointed to by multiple `StrListNode` objects, only count that String once.)

A. $O(1)$

B. $O(bS)$

C. $O(N)$

D. $O(N + bS)$

E. $O(N + b^2S)$

```
1     public static ListNode mystery(ListNode list) {
2         if (list == null || list.next == null) { return list; }
3         ListNode first = list;
4         ListNode current = first.next;
5         first.next = current.next;
6         current.next.next = null;
7         return first;
8     }
```

Figure 4: mystery method

For the next three problems, consider the `mystery` method (shown in Figure 4) above.

PROBLEM 8: Suppose we call `mystery` on input list `[0]`. What is the linked list returned by this call (if it does return)?

A. `[]`; that is, it returns `null`

B. `[0]`

C. `[0, 0]`

D. No list is returned because this input causes a `NullPointerException`

PROBLEM 9: Suppose we call `mystery` on input list `[1,2]`. This causes a `NullPointerException`. Which line causes the exception to occur?

A. 2

B. 3

C. 4

D. 5

E. 6

PROBLEM 10: Suppose we call `mystery` on input list `[3,4,5,6,7]`. What is the linked list returned by this call (if it does return)?

A. `[3,4]`

B. `[3,5]`

C. `[3,5,6,7]`

D. `[3,4,5,6,7]`

E. No list is returned because this input causes a `NullPointerException`

Two linked lists `a` and `b` are considered ***equal*** if they have the same number of nodes and each node in the i-th position in list `a` contains the same value is that contained in the i-th position of `b`. The ListsEqual APT asks one to write a method with two linked lists as parameters that returns `1` if the lists are equal and returns `0` otherwise.

The following `iterEqual` method is a correct iterative implementation of the desired method, and the `recEqual` method is a nearly-complete recursive implementation of the desired method.

```
1     public int iterEqual(ListNode a, ListNode b) {
2         ListNode curr1 = a;
3         ListNode curr2 = b;
4         while (curr1 != null && curr2 != null) {
5             if (curr1.info != curr2.info) { return 0; }
6             curr1 = curr1.next;
7             curr2 = curr2.next;
8         }
9         return curr1 == null && curr2 == null;
10    }
11
12    public int recEqual(ListNode a, ListNode b) {
13        if (a == null && b == null) { return 1; }
14        if (a == null && b != null) { return 0; }
15        if (a != null && b == null) { return 0; }
16        if (EXPR_1)                 { return 0; }
17        return EXPR_2;
18    }
```

Figure 5: iterEqual and recEqual methods

PROBLEM 11: What is the missing statement for `EXPR_1` in `recEqual` so that the method works correctly? Write it in the appropriate fill-in-the blank area of the back of the answer sheet. **Bubble A for this problem on the front of the answer sheet.**

PROBLEM 12: What is the missing statement for `EXPR_2` in `recEqual` so that the method works correctly? Write it in the appropriate fill-in-the blank area of the back of the answer sheet. **Bubble A for this problem on the front of the answer sheet.**

PROBLEM 13: Assuming that both input lists have size at most N, which one of the following best describes the asymptotic runtime of `iterEqual`?

A. $O(N)$

B. $O(N \log(N))$

C. $O(N^2)$

PROBLEM 14: Assuming that both input lists have size at most N, which one of the following best describes the asymptotic runtime of `recEqual`?

A. $O(N)$

B. $O(N \log(N))$

C. $O(N^2)$

The implementation of mergesort described in lecture sorts any input array of integers. The following is a nearly-complete implementation to sort any linked list represented by `ListNode` objects. The overall implementation is similar, but lines 3-11, which identify the middle `ListNode mid` in the input list, are more complicated than when using an array. The main idea is to iterate through the list with `current`, and for every two nodes that `current` iterates over, we iterate `mid` to the next node. When the loop stops, `mid` is (roughly) at the middle node of the list.

```
1     public static ListNode mergesort(ListNode list) {
2         if (list == null || list.next == null) return list;
3         ListNode current = list.next;
4         ListNode mid = list.next;
5         ListNode prevMid = list;
6         // the loop moves current through list twice as fast as mid
7         while (current.next != null && current.next.next != null) {
8             current = current.next.next; // move current down twice
9             prevMid = mid;
10            mid = mid.next; // move mid down once
11        }
12        MISSING_STATEMENT; // replace
13        ListNode sortedLeft = mergesort(list);
14        ListNode sortedRight = mergesort(mid);
15        return merge(sortedLeft, sortedRight);
16    }
17
18    public static ListNode merge(ListNode listA, ListNode listB) {
19        if (listA == null) { return listB; }
20        else if (listB == null) { return listA; }
21        ListNode smaller = listA; ListNode larger = listB;
22        if (listA.info > listB.info) {
23            smaller = listB; larger = listA;
24        }
25        smaller.next = merge(smaller.next, larger);
26        return smaller;
27    }
```

Figure 6: mergesort and merge methods

Given two lists of length at most N, the `merge` subroutine "merges" two sorted linked lists into a single sorted linked list of their contents and then returns it. The asymptotic runtime of `merge` is $O(N)$, which is the same as the implementation for arrays. The full code for the `merge` method is included, which is correct and may be useful, however it is not required to answer the following problems.

The next two problems reference the `mergesort` implementation (shown in Figure 6) on the previous page.

PROBLEM 15: Line 12 is missing a statement. The missing statement should effectively split the input list into two separate lists—one that begins at the `ListNode list` and one that begins at `ListNode mid`. Which one of the following statements should replace `MISSING_STATEMENT` so that the method works correctly?

A. `prevMid.next = mid`

B. `prevMid.next = null`

C. `mid.next = prevMid`

D. `list.next = null`

E. `mid.next = null`

PROBLEM 16: The asymptotic runtme of the implementation of `mergesort` to sort an array of N integers, as described in lecture, was shown to be characterized by the recurrence $T(N) = 2T(N/2) + O(N)$.
True or False: This recurrence also characterizes the asymptotic runtime of this implementation of `mergesort` for linked lists of integers.

A. True

B. False

```
1  public class TimeComp implements Comparator<String> {
2
3    @Override
4    public int compare(String timeA, String timeB) {
5      String[] splitA = timeA.split(":");
6      String[] splitB = timeB.split(":");
7      Integer hoursA   = Integer.parseInt(splitA[0]);
8      Integer minutesA = Integer.parseInt(splitA[1]);
9      Integer hoursB   = Integer.parseInt(splitB[0]);
10     Integer minutesB = Integer.parseInt(splitB[1]);
11     int compVal = minutesA - minutesB;
12     if (compVal != 0) { return compVal; }
13     return hoursB - hoursA;
14   }
15
16   public static void main(String[] args) {
17     TimeComp comp = new TimeComp();
18     String[] times =
19       new String[] {"3:00", "4:15", "6:15", "6:45", "11:00", "12:45"};
20     Arrays.sort(times, comp);
21     System.out.println(Arrays.toString(times));
22   }
23 }
```

Figure 7: TimeComp class

The next three problems reference the `TimeComp` class (shown in Figure 7) above.

PROBLEM 17: What is printed out by the main method?

A. `[3:00, 4:15, 6:15, 6:45, 11:00, 12:45]`

B. `[12:45, 11:00, 6:45, 6:15, 4:15, 3:00]`

C. `[6:45, 12:45, 4:15, 6:15, 3:00, 11:00]`

D. `[3:00, 11:00, 4:15, 6:15, 6:45, 12:45]`

E. `[11:00, 3:00, 6:15, 4:15, 12:45, 6:45]`

F. None of the above

PROBLEM 18: Suppose we call `Arrays.sort(times)` on line 20 instead of `Arrays.sort(times,comp)`. Which one of the following best describes what happens?

A. This code would not compile because `Arrays.sort()` requires two parameters

B. The code would not compile because `TimeComp` does not implement `Comparable`

C. The contents of the printed array would be in the natural ordering for Strings (lexicographic order)

D. The contents of the printed array would be the same as in the original code with `Arrays.sort(times, comp)`

E. Cannot be determined from the information given

PROBLEM 19: Suppose we call `Arrays.sort(times, comp)` on an array of N Strings of the correct form, i.e., `"hours:minutes"` where `hours` is an integer from `1` to `12` and `minutes` is an integer, formatted as a two-digit number, from `00` to `59`. The `.split` calls will run in $O(1)$ time on such Strings. What best describes the asymptotic runtime of the call in terms of N?

A. $O(1)$

B. $O(\log(N))$

C. $O(N)$

D. $O(N \log(N))$

E. $O(N^2 \log(N))$

```
1     // assumes the input array is sorted in non-decreasing order
2     //  e.g., [-10,-2,6,6,9,42,201].
3     public static boolean make201(int[] nums, int x) {
4         int low = 0;
5         int high = nums.length-1;
6         while (low <= high) {
7             int mid = (low+high)/2;
8             if (EXPR_1) { return true; }
9             else if (EXPR_2) { low = mid+1; }
10            else { high = mid-1; }
11        }
12        return false;
13    }
```

Figure 8: make201 method

For the next two problems, consider the incomplete `make201` method (shown in Figure 8) above. This method takes as input an `int[] nums` that is sorted in non-decreasing order (from least to greatest, possibly with consecutive equal values) and an integer `x`. The method should return `true` if there exists an integer `y` in `nums` such that `x+y=201`, and otherwise it should return `false`.

For example, if `nums` is `[3,5,25,61,100]` then `make201(nums, 140)` should return `true` (because `140+nums[3] == 201`), and `[3,5,25,41,100]` should return `false`.

There are four missing expressions in the code. What should these be so that the code works correctly and efficiently? The method should run in $O(\log(N))$ time, where N is `nums.length`.

PROBLEM 20: What is the missing statement for `EXPR_1`? Write it in the appropriate fill-in-the blank area of the back of the answer sheet. **Bubble A for this problem on the front of the answer sheet.**

PROBLEM 21: What is the missing statement for `EXPR_2`? Write it in the appropriate fill-in-the blank area of the back of the answer sheet. **Bubble A for this problem on the front of the answer sheet.**

PROBLEM 22: Suppose `high = mid-1` is replaced with `high = mid` in line 10, assuming that `EXPR_1` and `EXPR_2` are correct for the code (before the change). Does this change affect the correctness of the method? If yes, how does it change?

A. Yes, the change causes the method to return an incorrect answer on some inputs

B. Yes, the change prevents the method from returning any answer on some inputs

C. No, the change **does NOT** affect the correctness of the method

This page intentionally left blank. You may use it for scratch if you wish, but you should submit it with the exam.