Midterm 2: CompSci 201

Prof. Astrachan

November 1, 2023

Name: _____

netid: _____

In submitting this test, I affirm that I have followed the Duke Community Standard.

Community standard acknowledgement (signature) _____

You should bubble in answers for 27 questions on this exam.

The bubble sheet is for multiple choice questions. On the other/back side you'll find areas for fill-in-the blank questions. Please bubble in an answer for every question, choosing **Option A** for fill-in-the-blank questions as directed.

The definition of `ListNode` unless another is specified. This is the same definition as used in linked-list APT problems.

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

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

The method `fromArray` below should return a pointer to the first node of a linked list in which each node contains one value from the parameter `array` in the same order. For example, the call `fromArray(new int[]{1,2,3})` should return a pointer `front->1->2->3->null`

```
36  public ListNode fromArray(int[] array) {
37      if (array.length == 0) return null;
38      int lastIndex = array.length-1;
39      ListNode front = new ListNode(array[lastIndex]);
40      for(int k=lastIndex-1; k >= 0; k--) {
41          // one statement here
42      }
43      return front;
```

One line is missing from the body of the `for` loop. Consider the invariant *front points to the first node of the linked list that has been constructed so far from indexes* `(k..lastIndex]`

PROBLEM 1:

Which line of code establishes the loop invariant as true before the first time the loop body executes?

A. 37

B. 38

C. 39

PROBLEM 2:

The missing line should re-establish the loop invariant as true by creating a new node using the two-parameter constructor for `ListNode` and ensuring that this node is linked into the list being constructed.

What is the missing line? Write it in the appropriate fill-in-the blank area of the back of the bubble answer sheet. **Bubble A for this question on the front of the answer sheet.**

PROBLEM 3:

The missing statement calls `new` to create one new node. If the parameter `array` contains N values, what is the runtime of `fromArray(array)`?

A. $O(N)$

B. $O(N^2)$

C. $O(N^3)$

PROBLEM 4:

The method `count` should return the number of nodes in parameter `list`. The method is missing one statement. What is the missing statement? Write it in the appropriate fill-in-the blank area of the back of the bubble answer sheet. **Bubble A for this question on the front of the answer sheet.**

```
30  public int count(ListNode list){
31      int total = 0;
32      while (list != null) {
33          total += 1;
34          // missing statement
35      }
36      return total;
37  }
```

PROBLEM 5:

The method `reCount` is a recursive method to return the number of nodes in parameter `list`. The method is missing one (return) statement. What is the missing statement? Write it in the appropriate fill-in-the blank area of the back of the bubble answer sheet. **Bubble A for this question on the front of the answer sheet.**

```
38  public int reCount(ListNode list) {
39      if (list == null) return 0;
40      // missing statement
41  }
```

The next five questions are about the `IDnaStrand.charAt` method used in the *P3/DnaStrand* project.

The `charAt` method for classes implementing `IDnaStrand` was simple for the classes `StringStrand` and `StringBuilderStrand` and runs in constant, $O(1)$ time. The implementation below is the same in both `StringStrand` and `StringBuilderStrand` though `myInfo` has type `String` and `StringBuilder`, respectively, in those classes.

```
78    @Override
79    public char charAt(int index) {
80        return myInfo.charAt(index);
81    }
```

PROBLEM 6:

For the method below, what is the runtime of the call `getReverse(strand)` if parameter `strand` is a `StringStrand` object?

```
2   public String getReverse(IDnaStrand strand){
3       StringBuilder sb = new StringBuilder();
4       for(int k= (int) strand.size()-1; k >= 0; k--) {
5           sb.append(strand.charAt(k));
6       }
7       return sb.toString();
8   }
```

A. $O(N)$

B. $O(N^2)$

C. $O(N^3)$

PROBLEM 7:

For the same method, what is the runtime of the call `getReverse(strand)` if `strand` is a `StringBuilderStrand` object?

A. $O(N)$

B. $O(N^2)$

C. $O(N^3)$

In the P3 assignment the implementation of `LinkStrand.charAt` shown below was provided as a model. This implementation has a runtime of $O(N)$ for the call `charAt(N)`

```
79    @Override
80    public char charAt(int index) {
81        if (index < 0 || index >= mySize){
82            throw new IndexOutOfBoundsException(index+" is out of bounds");
83        }
84        int count = 0;
85        int dex = 0;
86        Node list = myFirst;
87        while (count != index) {
88            count++;
89            dex++;
90            if (dex >= list.info.length()) {
91                dex = 0;
92                list = list.next;
93            }
94        }
95        return list.info.charAt(dex);
96    }
```

PROBLEM 8:

For all three `IDnaStrand` implementations of `charAt` shown above (`StringStrand`, `StringBuilderStrand`, and `LinkStrand`) the call `strand.charAt(-1)` generates an exception:

A. True, they all generate an exception

B. False, at least one of them does not generate an exception

PROBLEM 9:

What is the runtime of the call `getReverse(strand)` (see method above) if `strand` is a `LinkStrand` object? and the `charAt` implementation above is used?

A. $O(N)$

B. $O(N^2)$

C. $O(N^3)$

PROBLEM 10:

You were asked to write an efficient implementation of `charAt` so that the code below executes in $O(N)$ time for a `LinkStrand` object `strand` to count the number of `'a'` characters in the strand.

```
int count = 0;
for(int k=0; k < (int) strand.size(); k++){
    char ch = strand.charAt(k);
    if (ch == 'a') count++;
}
```

Assume `LinkStrand.charAt` is efficient so that the code above to count the number of `'a'` values does run in $O(N)$ time. What is the runtime of `getReverse(strand)` for a `LinkStrand` object `strand` with this efficient implementation?

A. $O(N)$

B. $O(N^2)$

C. $O(N^3)$

The next three questions are about the *ListLastFirst* APT whose wording is:

> Write a method `move` that moves the last node of a linked list to the front of the list, leaving the order of the other nodes unchanged. You can create a new list with new nodes or you can simply move the last node of the ListNode parameter to the front and return the modified list.
>
> If `list->1->2->3->4->null` the call `move(list)` returns `ptr->4->1->2->3->null`

The code below is all-green if the missing statement is filled in correctly. The code is designed to find the last node, remove it from the list, and then make what was the last node the first node.

```
2   public ListNode move(ListNode list) {
3       if (list == null || list.next == null) return list;
4
5       ListNode first = list;
6       while (list.next.next != null) {
7           list = list.next;
8       }
9       ListNode last = list.next;
10      last.next = first;
11      first = last;
12      // missing statement
13      return first;
14  }
```

PROBLEM 11:

Which best categorizes what `list` points to when line 9 executes, after the loop has terminated.

A. `list` points to the last node of the list passed as a parameter

B. `list` points to the node before the last node of the list passed as a parameter (the second to last node)

C. `list` points to the node before the node before the last node of the list passed as a parameter (the third to last node)

PROBLEM 12:

If no statement is added on line 12, then when the APT tester runs the tests for an empty list and a one-node list are green, but all other tests *fail* with the result shown as expected `[correct list]` but got `[infinite list]`

What one line of code should be added on line 12 so that the solution is all-green for all cases? Write the answer in the fill-in-the-blank section on the back of the answer sheet. **Bubble A for this question on the front of the answer sheet.**

PROBLEM 13:

The code below is a recursive version of `move` to solve the APT. The code will be all-green if one statement is added on line 20. When run without the statement every test case for a non-null list shows the actual return value as a one-node list that contains the value that should be the first node of the returned list, e.g., if `list-1->2->3->4->null` the call `move(list)` returns `ptr->4->null`

```
16    public ListNode move(ListNode list) {
17        if (list == null || list.next == null) return list;
18        ListNode temp = move(list.next);
19        list.next = temp.next;
20        // missing statement
21        return temp;
22    }
```

Which statement should be added to line 20 to make the solution all green?

A. `temp = list.next`

B. `temp.next = list`

C. `temp.next = list.next`

PROBLEM 14:

The runtime of both the iterative solution and the recursive solution is the same. What is the runtime for an N-node list?

A. $O(1)$

B. $O(N)$

C. $O(N^2)$

D. $O(N^3)$

There are two problems relating to the (challenge) `SortedListRemoval` APT whose wording is:

> Write method `uniqify` to remove all duplicate values from a sorted linked list of integer values, leaving the unique values occurring once and in order. For example, the list represented by `[1,1,2,3,3,3,4]` would be changed to `[1,2,3,4]`. You should remove nodes and relink nodes rather than creating a new list.

PROBLEM 15:

The implementation below is all-green if one statement is added on line 6. No matter what is added as the missing statement, what does the code below return for the call `uniqify(list)` if `list->4->null`?

```
2   public ListNode uniqify(ListNode list) {
3       if (list == null || list.next == null) return list;
4       ListNode after = uniqify(list.next);
5       if (list.info == after.info) {
6           // code missing here
7           return list;
8       }
9       return list;
10  }
```

A. `null`

B. `4->null`

C. `4->4->null`

PROBLEM 16:

The missing code on line 6 in the code above should remove the node referenced by `after` from the linked list returned. Fill in the blank with the missing code on the fill-in-the-blank section on the back of the answer sheet. **Bubble A for this question on the front of the answer sheet.**

PROBLEM 17:

If the word `if` is replaced by `while` on line 5, the same result will be returned for the call `uniqify(list)` as when `if` is used, regardless of the statement added on line 6.

A. True, the behavior will be the same

B. False, with `while` the behavior is different than with `if`.

The next two questions are about the *ListSum* APT whose wording is:

> Write a method `sum` that returns the sum of the values greater than `thresh` in its list parameter, a linked list of int values.
>
> For example, if `list->1->2->3->4->5->null` the call `sum(list,3)` returns 9; the call `sum(list,5)` returns 0; and the call `sum(list,0)` returns 15.

The code below will be all green if `expression1` and `expression2` are replaced correctly. In that case, the value of `after` is the correct sum of all the nodes after `list` that are greater than `thresh`.

```
13  public int sum(ListNode list, int thresh) {
14      if (list == null) return 0;
15      int after = sum(list.next,thresh);
16      if (list.info <= thresh) return expression1;
17      return expression2;
18  }
```

PROBLEM 18:

Which of the following is the correct replacement for `expression1`?

A. `list.info`

B. `after`

C. `list.info + after`

PROBLEM 19:

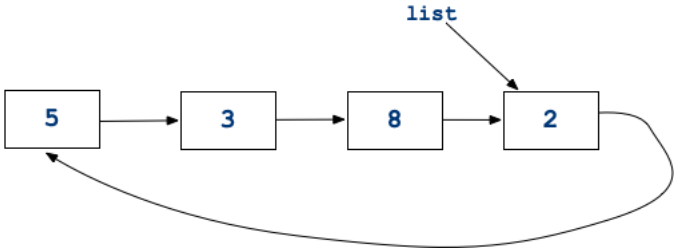
Which of the following is the correct replacement for `expression2`?

A. `list.info`

B. `after`

C. `list.info + after`

In a *circularly-linked* list, the last node of the list points to the first node rather than pointing at `null`. Such lists are typically represented by a pointer to the last node as shown, so that `list` references the last node and `list.next` references the first node. The diagram below represents the list `[5,3,8,2]` with 2 being the last node and 5 the firstnode.



If two missing statements are added, the method `makeCircular` below returns a pointer to the last node of a circulary-linked list so that the call `ListNode list = makeCircular(new int[]{5,3,8,2});` returns a pointer to the last node of the list pictured above.

```
13    public ListNode makeCircular(int[] a) {
14        ListNode first = new ListNode(a[0]);
15        ListNode last = first;
16        for(int k=1; k < a.length; k++) {
17            last.next = new ListNode(a[k]);
18            // missing statement 1
19        }
20        // missing statement 2
21        return last;
22    }
```

PROBLEM 20:

The code for *missing statement 1* on line 18 should ensure that the code on line 17 always adds a new node as the last node of the list being constructed in the loop. What is this statement for line 18?

A. `last = last.next;`

B. `last.next = first;`

C. `first.next = last;`

PROBLEM 21:

The code for *missing statement 2* on line 20 should ensure that the list being returned is circular. Assuming the correct statement has been added to line 18, what statement should be on line 20?

A. `last = last.next;`

B. `last.next = first;`

C. `first.next = last;`

PROBLEM 22:

The code in `countCircular` below is intended to return the number of nodes in a circularly-linked list so that the call `countCircular(list)` returns 4 for the list diagrammed at the beginning of this problem.

```
2   public int countCircular(ListNode list) {
3       if (list == null) return 0;
4       int count = 1;
5       ListNode anchor = list;
6       while (expression) {
7           count += 1;
8           list = list.next;
9       }
10      return count;
11  }
```

What statement replaces `expression` as the loop test so that the method works correctly?

A. `list.next != anchor`

B. `anchor.next != list`

C. `anchor != list`

Two circularly-linked lists can be merged into one circularly-linked list in constant time. The call `append(a,b)`, where `a` and `b` each point to the last node of two different circularly-linked lists, should make the last node of list `a` point at the first node of list `b`, and make the last node of list `b` point at the first node of list `a`. The method returns `b`, a pointer the last node of the combined/merged lists (the two original lists are now a single list).

```
23  public ListNode append(ListNode a, ListNode b) {
24      ListNode first = a.next;
25      // missing statement 1
26      // missing statement 2
27      return b;
28  }
```

PROBLEM 23:

What is the first missing statement? It should be an assignment statement and **not reference** variable `first`. Write the answer in the fill-in-the-blank section on the back of the answer sheet. **Bubble A for this question on the front of the answer sheet.**

PROBLEM 24:

What is the second missing statement? It should be an assignment statement and **must reference** variable `first`. Write the answer in the fill-in-the-blank section on the back of the answer sheet. **Bubble A for this question on the front of the answer sheet.**

The next three problems are based on a problem from the *LeetCode* website whose wording is:

> Given an array of integers `nums` and an integer `target`, write method `twoSum` that returns indexes of the two numbers from `nums` such that they add up to target.
>
> *You may assume that there is exactly one solution*, and you may *not use the same element twice*. You can return the indexes in any order.
>
> For example, if `int[] nums = {2,7,11,15}` the call `twosum(nums,9)` should return `[0,1]` (or `[1,0]`) since 2+7 == 9; 2 has index zero in `nums` and 7 has index 1. The call `twoSum(nums,26)` should return `[2,3]` (or `[3,2]`) since 11+15 = 26.

The code below is all-green for `twosum`.

```
2     public int[] twoSum(int[] nums, int target) {
3         ArrayList<Integer> list = new ArrayList<>();
4         for(int val : nums) list.add(val);
5         for(int k=0; k < nums.length; k++) {
6             int val = nums[k];
7             int dex = list.indexOf(val);
8             if (dex >= 0) {
9                 int match = target-val;
10                int matchDex = list.lastIndexOf(match);
11                if (matchDex >= 0 && matchDex != dex) {
12                    return new int[]{dex,matchDex};
13                }
14            }
15        }
16        // never reached
17        return new int[]{-1,-1};
18    }
```

PROBLEM 25:

Which line of code in method `twoSum` guarantees that the same element is not used twice?

A. 8

B. 9

C. 10

D. 11

PROBLEM 26:

If `nums` has N-elements, what is the runtime complexity of `twoSum(nums,target)` for any value of `target`? Note that `indexOf` and `lastIndexOf`, which find the first and last index, respectively of their parameter, each must examine all values in the `ArrayList` to determine if the parameter is found and what its index is.

A. $O(N)$

B. $O(N \log N)$

C. $O(N^2)$

D. $O(N^3)$

PROBLEM 27:

The code below uses a map of each value in `nums` (as the key) to that value's index in the parameter `nums` (as the value).

```
2     public int[] twoSum(int[] nums, int target) {
3         HashMap<Integer,Integer> map = new HashMap<>();
4         for(int k=0; k < nums.length; k++) {
5             int current = nums[k];
6             int other = target-current;
7             if (map.containsKey(other)) {
8                 return new int[]{k,map.get(other)};
9             }
10            map.put(current,k);
11        }
12        return new int[]{-1,-1};
13    }
```

What is the runtime complexity of this version of `twoSum` when `nums` contains N elements?

A. $O(N)$

B. $O(N \log N)$

C. $O(N^2)$

D. $O(N^3)$