Midterm Exam 2, Compsci 201
Spring 2023, Duke University

March 22, 2023

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. If you continue writing outside of the box, you must clearly indicate where or your answers may 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 inch reference sheet, on which you should write your name and NetID and submit along with your exam when you are finished.

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.** _____

```
1     public LinkedList<Integer> copy (LinkedList<Integer> list) {
2         LinkedList<Integer> newList = EXPR_1;
3         for (int val : list) { EXPR_2; }
4         return newList;
5     }
```

Figure 1: copy method.

1. (4 points). Consider the `copy` method outlined in Figure 1. Suppose `list` is a `java.util LinkedList`. The method should create a separate copy of `list` in memory containing the same values in the same order, and return the copy.

There are two missing expressions in the code. What should these be so that the code works correctly?

A. `EXPR_1`:

B. `EXPR_2`:

```
1     public boolean hasDuplicates (LinkedList<Integer> list) {
2         for (int i=0; i<list.size(); i++) {
3             for (int j=i+1; j<list.size(); j++) {
4                 if (list.get(i)==list.get(j)) { return true; }
5             }
6         }
7         return false;
8     }
```

Figure 2: hasDuplicates method.

2. (4 points). Consider the `hasDuplicates` method defined in Figure 2. Suppose `list` is a `java.util LinkedList` with N elements. What is the asymptotic runtime complexity of `hasDuplicates(list)`? Briefly explain your answer.

```
1  public class ListNode {
2      int info;
3      ListNode next;
4      ListNode(int x){
5          info = x;
6      }
7      ListNode(int x,
       ListNode node) {
8          info = x;
9          next = node;
10     }
11 }
```

(a) ListNode class. Several questions that follow refer to the ListNode class.

```
1  public static void main(String[] args) {
2      ListNode node = new ListNode(1);
3      node.next = new ListNode(2);
4      node.next.next = new ListNode(3);
5      ListNode result = mutate(node);
6      ListNodeUtil.printList(result);
7      ListNodeUtil.printList(node);
8  }
9
10 static ListNode mutate(ListNode node) {
11     ListNode temp = node.next;
12     node.next = node.next.next;
13     node = node.next;
14     node.next = temp;
15     temp.next = null;
16     return node;
17 }
```

(b) mutate and main methods

Figure 3

3. (9 points). Suppose we run the `main` method defined in Figure 3b. The `mutate` method is defined in the same figure. The `ListNodeUtil.printlist(node)` method prints the values in a singly linked list of `ListNode` objects beginning at `node`.

A. What will be printed by line 6 of the `main` method, `ListNodeUtil.printList(result)`? For example, you could write (though it would not be correct), `[1, 2, 3]`. You do not need to explain your answer.

B. What will be printed by line 7 of the `main` method, `ListNodeUtil.printList(node)`? For example, you could write (though it would not be correct), `[1, 2, 3]`. You do not need to explain your answer.

C. If we comment out line 4 of the `main` method (`node.next.next = new ListNode(3);`) then running the code in the `main` method results in a null pointer exception during the execution of the mutate method called on line `5`. On what line of the `mutate` method does the null pointer exception occur? Briefly explain your answer.

```
1  public ListNode merge(ListNode listA, ListNode listB) {
2      ListNode first;
3      if (listA.info <= listB.info) {
4          first = listA;  listA = listA.next;
5      }
6      else {
7          first = listB;  listB = listB.next;
8      }
9
10     ListNode current = first;
11     while (listA != null && listB != null) {
12         if (EXPR_1) {
13             current.next = listA;  listA = listA.next;
14         }
15         else {
16             current.next = listB;  listB = listB.next;
17         }
18         current = EXPR_2;
19     }
20
21     if (EXPR_3) { current.next = listA; }
22     else { current.next = listB; }
23     return EXPR_4;
24 }
```

Figure 4: merge method

4. (8 points). An incomplete outline of the `merge` method is shown in Figure 4 (note that lines 4, 7, 13, and 16 each contain two assignment statements). The method takes as input `ListNode listA` and `ListNode listB`, each a reference to the first node of a nonempty singly linked list of `ListNode` objects sorted from least to greatest. The method should merge these into a single list sorted from least to greatest and return a reference to the first node of the merged list.

For example, if `listA` is `[1, 2, 4]` and `listB` is `[3, 5, 6]`, then `merge(listA, listB)` should return `[1, 2, 3, 4, 5, 6]`. No new nodes are created; `listA` and `listB` are merged directly.

There are four missing expressions in the code. What should these be so that the code works correctly? You do not need to explain your answer.

A. `EXPR_1`:

B. `EXPR_2`:

C. `EXPR_3`:

D. `EXPR_4`:

```
1     public static ListNode rec(ListNode list) {
2         if (list == null) { return null; }
3         if (list.next == null) { return new ListNode(list.info); }
4         ListNode after = rec(list.next);
5         ListNode current = after;
6         while (current.next != null) {
7             current = current.next;
8         }
9         current.next = new ListNode(list.info);
10        return after;
11    }
```

Figure 5: rec method

5. (6 points). Suppose we call the recursive `rec` method (shown in Figure 5) on the inputs as shown below. State what the resulting linked list returned would be. You do not need to explain your answers.

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. You should write your answers in the same format.

A. `[1]`:

B. `[0, 1]`:

C. `[2, 0, 1]`:

6. (3 points). State a recurrence relation of the form $T(N) = ...$ that describes the runtime complexity of the `rec` method as a function of N where N is the number of nodes of the input linked list. Briefly explain your answer, referencing the code. You do not need to solve the recurrence.

```
1     // Assumes nums is sorted from least to greatest
2     public boolean adjacentSum(int[] nums, int target) {
3         int low = 0;
4         int high = nums.length-1;
5         while (low < high) {
6             int mid = (low + high)/2;
7             int check = EXPR_1;
8             if (check == target) { return true; }
9             else if (EXPR_2) { high = mid; }
10            else { EXPR_3; }
11        }
12        return EXPR_4;
13    }
```

Figure 6: adjacentSum method

7. (8 points). An incomplete outline of the `adjacentSum` method is shown in Figure 6. The method takes as input an `int[] nums` that is sorted from least to greatest and an `int target`. The method should return `true` if there exist adjacent values (that is, values `nums[i]` and `nums[i+1]` for some index `i`) in `nums` that sum to `target` (that is, `nums[i] + nums[i+1] == target`), or `false` otherwise.

For example, if `nums` is `{1, 2, 4, 8, 9}` then `adjacentSum(nums, 3)` and `adjacentSum(nums, 17)` should both return `true`, but `adjacentSum(nums, 5)` should return `false`.

There are four missing expressions in the code. What should these be so that the code works correctly and efficiently? The algorithm should run in $O(\log(N))$ time where N is the length of `nums`. You do not need to explain your answers.

A. `EXPR_1`:

B. `EXPR_2`:

C. `EXPR_3`:

D. `EXPR_4`:

```
1  public class MatchComp implements Comparator<String> {
2      private String ref;
3      public MatchComp(String ref) { this.ref = ref; }
4
5      @Override
6      public int compare(String a, String b) {
7          return (-1) * (score(a) - score(b));
8      }
9
10     public int score(String strand) {
11         int common = 0;
12         int smallerLength = Math.min(ref.length(), strand.length());
13         for (int i=0; i<smallerLength; i++) {
14             if (ref.charAt(i)==strand.charAt(i)) { common++; }
15         }
16         return common;
17     }
18 }
```

Figure 7: MatchComp Comparator

```
1      public static void main(String[] args) {
2          MatchComp comp = new MatchComp("agtc");
3          String[] strands = new String[]{"aaaa", "ggtc", "agag"};
4          Arrays.sort(strands, comp);
5          System.out.println(Arrays.toString(strands));
6      }
```

Figure 8: main method

8. (4 points). What will be printed by line 5 of the `main` (Figure 8)? Note that it references the `Comparator` class defined in Figure 7 and prints the values of the array `strands`. For example, you could write (though it would not be correct) `{"aaaa", "ggtc", "agag"}`. Briefly explain your answer.

9. (4 points). Suppose `ref` is a `String` of length L and `comp = new MatchComp(ref);`. Suppose `myWords` is an array of N `String`s, and each individual `String` has length M. What is the asymptotic runtime complexity of `Arrays.sort(myWords, comp)`? Express your answer in big O notation with respect to some combination of L, N, and M. Briefly explain your answer.

This page left intentionally blank. Feel free to use it for scratch work if desired.