

## Midterm Exam Solutions

Name:	Sample Solutions	
ID:	bovik	@washington.edu
TA:	The Best	Section: A9

## INSTRUCTIONS:

- You have **60 minutes** to complete the exam.
- You *will* receive a deduction if you keep working after the instructor calls for papers.
- This exam is closed-book and closed-notes. You may not use any computing devices including calculators.
- Code will be graded on proper behavior/output and not on style, unless otherwise indicated.
- Do not abbreviate code, such as “ditto” marks or dot-dot-dot (“...”) marks. The only abbreviations that are allowed for this exam are: `S.o.p` for `System.out.print` and `S.o.pln` for `System.out.println`.
- You do not need to write import statements in your code.
- You may not use extra scratch paper on this exam. Use the provided spaces for extra work.
- If you write work you want graded on a strange page, clearly label it.
- If you enter the room, you must turn in an exam before leaving the room.
- You must show your Student ID to a TA or instructor for your exam to be accepted.
- If you get stuck on a problem, move on and come back to it later.

Problem	Points	Score	Problem	Points	Score
1	15		4	20	
2	15		5	20	
3	15		6	15	
			$\Sigma$	100	

Question #1: Scratch Work

*This page left intentionally almost Adam.*

## Mechanical Missions.

This section tests whether you are able to trace through code of various types in the same way a computer would.

### 1. Recursive Tracing [15 points]

```
1 public static void mystery(int n) {
2     if (n <= 1) {
3         System.out.print(n);
4     }
5     else {
6         System.out.print(n + ", ");
7         mystery(n / 2);
8         System.out.print(", " + n);
9     }
10 }
```

For each of the following, fill in the String *printed* when the given `mystery` method call is run.

	mystery Call	String Printed
(a)	mystery(1)	1
(b)	mystery(3)	3, 1, 3
(c)	mystery(4)	4, 2, 1, 2, 4
(d)	mystery(6)	6, 3, 1, 3, 6
(e)	mystery(12)	12, 6, 3, 1, 3, 6, 12

*This page left intentionally almost Adam.*

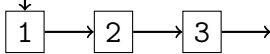
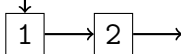

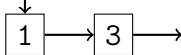
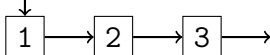

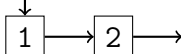
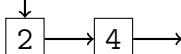
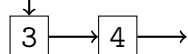
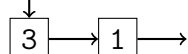
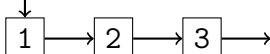
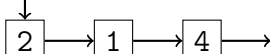
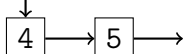
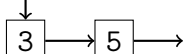
## 2. ListNode Before & After [15 points]

Your task in this question is to write code to turn the “before” picture into the “after” picture. You may not change any existing node’s data value. You may not construct any new nodes. In each part, you may declare *up to two* new variables of type ListNode. Recall the ListNode class from lecture:

```

1 public class ListNode {
2     public final int data;
3     public ListNode next;
4
5     public ListNode(int data) { ... }
6     public ListNode(int data, ListNode next) { ... }
7 }

```

	Before & After	Your Code
(a)	<p>list1</p> <p>Before: </p> <hr/> <p>list1</p> <p>After: </p>	<p>list2</p> <p>↓</p> <pre>list2 = list1.next.next; list1.next.next = null;</pre> <p>list2</p> <p>↓</p> <p></p>
(b)	<p>list1</p> <p>Before: </p> <hr/> <p>list1</p> <p>After: </p>	<p>list2</p> <p>↓</p> <p></p> <pre>list2.next = list1.next; list1.next = list2; list2 = null;</pre> <p>list2</p> <p>↓</p>
(c)	<p>list1</p> <p>Before: </p> <hr/> <p>list1</p> <p>After: </p>	<p>list2</p> <p>↓</p> <p></p> <pre>list1.next.next = list2.next; list2.next = list1; list1 = list1.next; list2.next.next = null;</pre> <p>list2</p> <p>↓</p> <p></p>
(d)	<p>list1</p> <p>Before: </p> <hr/> <p>list1</p> <p>After: </p>	<p>list2</p> <p>↓</p> <p></p> <pre>ListNode temp = list2; list2 = list1.next.next; list1.next.next = list1; list1 = list1.next; list1.next.next = temp; list2.next = temp.next; temp.next = null;</pre> <p>list2</p> <p>↓</p> <p></p>



## Programming Pursuits.

This section tests whether you synthesized various topics well enough to write novel programs using those topics.

### 3. AryIntLst [15 points]

Add a method `collapse` to the `ArrayIntList` class that collapses the contents of the `ArrayIntList` of integers by replacing each successive pair of integers with the sum of the pair. If the list stores an odd number of elements, the final element should not be collapsed.

#### Example Output

Before <code>list.collapse()</code>	After <code>list.collapse()</code>
[7, 2, 8, 9, 4, 13, 7, 1, 9, 10]	[9, 17, 17, 8, 19]
[1, 2, 3, 4, 5]	[3, 7, 5]

#### Implementation Restrictions

- You may not call any other methods on the `ArrayIntList` object (e.g., `add`, `remove`)
- You may not use a `foreach` loop on the `ArrayIntList`.
- You may not use any other **data structures such as arrays, lists, queues, etc.**
- Your solution should run in  $\mathcal{O}(n)$  time, where  $n$  is the number of elements in the list.

```
public class ArrayIntList {
    private int size;
    private int[] elementData;

    // Write Your Solution Here
```

```
}
```

### Question #3: Extra Solution Space

*Solution:*

```
1 public void collapse() {
2     for (int i = 0; i < size / 2; i++) {
3         this.elementData[i] = this.elementData[2 * i] + this.elementData[2 * i + 1];
4     }
5     if (this.size % 2 == 0) {
6         this.size = this.size / 2;
7     }
8     else {
9         this.elementData[this.size / 2] = this.elementData[this.size - 1];
10        this.size = this.size / 2 + 1;
11    }
12 }
```



#### 4. RecuRsion [20 points]

Write a *recursive* method called `commonChars` that takes two `Strings` as parameter and returns a new string representing the characters that they have in common. The characters that are different between the two strings should be represented by a "." in the new `String`. You may assume that the parameters do not contain any "." characters.

#### Example Output

Method Call	Return Value
<code>commonChars("", "")</code>	<code>""</code>
<code>commonChars("a", "a")</code>	<code>"a"</code>
<code>commonChars("AaA", "Aaa")</code>	<code>"Aa."</code>
<code>commonChars("abc", "cba")</code>	<code>".b."</code>
<code>commonChars("hello world", "heyya world")</code>	<code>"he... world"</code>

#### Implementation Restrictions

- You must use recursion.
- You may assume that both input `Strings` are the same length.
- You may assume that both input `Strings` are not null.
- You may only use the `String` methods on the cheatsheet.

## Question #4: Extra Solution Space

*Solution:*

```
1 public static String commonChars(String s1, String s2) {
2     if (s1.length() == 0) {
3         return "";
4     }
5     else {
6         if (s1.charAt(0) == s2.charAt(0)) {
7             return s1.charAt(0) + commonChars(s1.substring(1), s2.substring(1));
8         }
9         else {
10            return "." + commonChars(s1.substring(1), s2.substring(1));
11        }
12    }
13 }
```

### 5. Some Maps [20 points]

Write a method called `sumMaps` that takes two Maps from Strings to ints as parameters and returns a new map that associates each key from either/both of the inputs with the *sum* of the values in both maps. The map returned by your method should be as fast as possible.

#### Example Output

Input Maps	Return Value
{}, {a=1, e=3}	{a=1, e=3}
{a=1, b=2, c=3, d=4}, {a=2, c=5, e=6}	{a=3, b=2, c=8, d=4, e=6}

#### Implementation Restrictions

- You may assume that the arguments are not null
- Your method should not modify the parameters
- Your method should run in  $\mathcal{O}(n)$  time (e.g., you should not have any nested loops)

## Question #5: Extra Solution Space

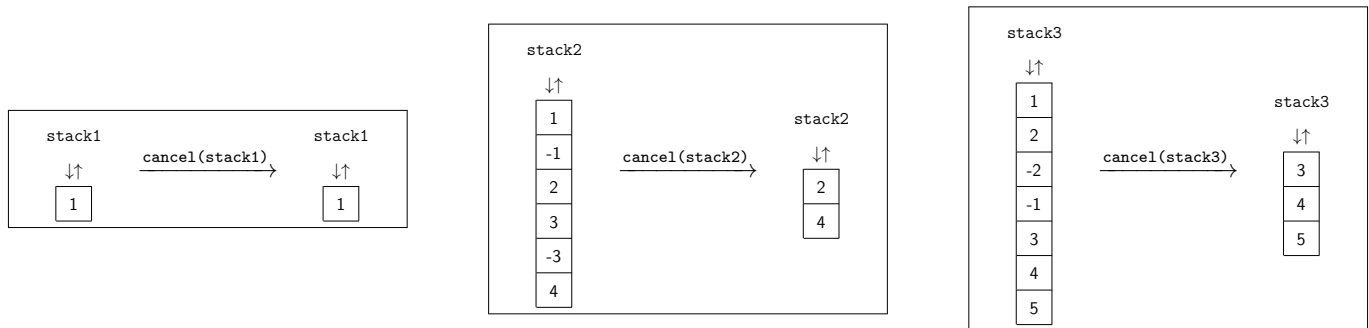
*Solution:*

```
1 public static Map<String, Integer> sumMaps(Map<String, Integer> m1, Map<String, Integer> m2) {
2     Map<String, Integer> output = new HashMap<String, Integer>();
3     for (String key : m1.keySet()) {
4         output.put(key, m1.get(key));
5     }
6     for (String key : m2.keySet()) {
7         if (!output.containsKey(key)) {
8             output.put(key, 0);
9         }
10        output.put(key, output.get(key) + m2.get(key));
11    }
12    return output;
13 }
```

## 6. Stacks and Queues... [15 points]

Write a method `cancel` that takes a `Stack` of integers as a parameter and modifies it by removing all consecutive pairs that sum to zero. Whenever removing one pair that sums to zero results in another consecutive pair that sums to zero, make sure to remove that pair as well.

### Example Output



### Implementation Restrictions

- You may assume the input stack is not null.
- You may create **at most one stack** and no other data structures.
- You may not use recursion to solve this problem.
- Your solution must run in  $\mathcal{O}(n)$  time, where  $n$  is the size of the stack.

Use This Box For Scratch Work

DO NOT WRITE YOUR SOLUTION HERE

A large, empty rectangular box with a thin black border, intended for scratch work. It occupies the majority of the page's width and height.

Solution:

### Question #6: Solution Space

```
1 public static void cancel(Stack<Integer> stack) {
2     Stack<Integer> tmp = new Stack<Integer>();
3     while (!stack.isEmpty()) {
4         int top = stack.pop();
5         if (!stack.isEmpty() && top + stack.peek() == 0) {
6             stack.pop();
7         }
8         else if (!tmp.isEmpty() tmp.peek() + top == 0) {
9             tmp.pop();
10        }
11        else {
12            tmp.push(top);
13        }
14    }
15    while (!tmp.isEmpty()) {
16        stack.push(tmp.pop());
17    }
18 }
```

