UC Berkeley – Computer Science
CS61BL: Data Structures

Midterm 1, Summer 2016

This test has 8 questions worth a total of 30 points. The exam is closed book, except that you are allowed to use one double-sided page of notes as a cheat sheet. No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided.

Write the statement out below in the blank provided and sign. You may do this before the exam begins. **Any plagiarism, no matter how minor, will result in points deducted from your exam.**

"I have neither given nor received any assistance during the taking of this exam."

_____

_____

Signature: _____

Write your name and student ID on the front page. Write the names of your neighbors. Write and sign the given statement. Once the exam has started, write your login in the corner of every page.

Name: _____ **Your Login:** cs61bl-_____

SID: _____ **Name of person to left:** _____

TA: _____ **Name of person to right:** _____

Tips:

- There may be partial credit for incomplete answers. Write as much of the solution as you can, but bear in mind that we may deduct points if your answers are much more complicated than necessary.
- There are a lot of problems on this exam. Work through the ones with which you are comfortable first. Do not get overly captivated by interesting design issues or complex corner cases you're not sure about.
- Not all information provided in a problem may be useful.
- Unless otherwise stated, all given code on this exam should compile. All code has been compiled and executed before printing, but in the unlikely event that we do happen to catch any bugs during the exam, we'll announce a fix. Unless we specifically give you the option, the correct answer is not 'does not compile.'

Optional. Mark along the line to show your feelings on the spectrum between ☹ and ☺.

Before exam: [☹ _____ ☺]

After exam: [☹ _____ ☺]

1. Hanamura (4 pts)

Specify the full output of attempting to compile and run the following programs. If an error or exception occurs at any point, you only need to specify whether it is a runtime or compile-time error, rather than writing the output of the Java interpreter. You should still write the output of lines that execute before any **runtime** exceptions occur.

Code	Output
`public class Fastest {` `  String map;` `  public Fastest copy() {` `  map = "Possible";` `  return this;` `  }` `  public static void main(String[] args) {` `  Fastest map1 = new Fastest();` `  map1.map = "BGH";` `  Fastest map2 = map1.copy();` `  System.out.println(map1.map);` `  map2.map = "Lost Temple";` `  System.out.println(map2.map);` `  }` `}`	`Possible` `Lost Temple`
`public class Ketchup {` `  public void friend(int ketchup) {` `  int tomato = ((ketchup + 4) / 2);` `  System.out.println(tomato);` `  }` `  public static void main(String[] args) {` `  Ketchup ash = new Ketchup();` `  friend(4);` `  }` `}`	`Compile-Time error: Cannot access a non-static method from a static context`
`public class Thyme {` `  public static String zoo = "zoo";` `  public void gul(String dan) {` `  dan = dan + this.zoo;` `  zoo = dan;` `  System.out.println(dan);` `  }` `  public static void main(String[] args) {` `  Thyme herb = new Thyme();` `  herb.gul("hand");` `  herb.gul("hand");` `  }` `}`	`handzoo` `handhandzoo`

`` public class Gateway { int id; double location; Object[] productionQ = new Object[3]; public void processQueue() { for (int i = 1; i <= productionQ.length; i++) { produce(productionQ[i]); } } public void produce(Object unit) { if (unit != null) System.out.println("Produced " + unit.toString()); } public static void main(String[] args) { Gateway gw = new Gateway(); gw.productionQ[0] = "zealot"; gw.productionQ[1] = "stalker"; gw.processQueue(); } } ```	``` Produced stalker Runtime error: ArrayIndexOutOfBoundsExceptio n ```

2. Dorado (7 pts)

a. Provide the tightest bound you can on these functions. Use Big-Theta notation if possible, otherwise just use Big-O. Reduce the expression as much as possible. Unreduced expressions will not receive credit. N and M are positive integers, representing the size of the input(s).

Function	Bound
N	Θ(N)
$10000 + N^2$	$\Theta(N^2)$
$2N + 4N^4$	$\Theta(N^4)$
$2 * 3^N$	$\Theta(3^N)$
42 + (3 / N)	Θ(1)

For parts b and c, suppose we have a linked list of length N. Provide a bound on the **worst case** running time of the following operations in terms of N. Use Big-Theta notation. (These linked lists are from lab06.)

b. If our list is singly-linked with a pointer to the front of the list:

Operation	Bound
`size()`	Θ(1)
`get(int index)`	Θ(N)
`insertFront(Object o)`	Θ(1)
`/* Inserts a node before n, containing o */ insertBefore(Object o, ListNode n)`	Θ(N)
`remove(int index)`	Θ(N)
`/* Assume n is a ListNode inside this list */ remove(ListNode n)`	Θ(N)
`reverse()`	Θ(N)

c. If our list is doubly-linked with pointers to the front and back of the list:

Operation	Bound
`size()`	Θ(1)
`get(int index)`	Θ(N)
`insertFront(Object o)`	Θ(1)
`/* Inserts a node before n, containing o */ insertBefore(Object o, ListNode n)`	Θ(1)
`remove(int index)`	Θ(N)
`/* Assume n is a ListNode inside this list */ remove(ListNode n)`	Θ(1)
`reverse()`	Θ(N)

d. Provide simplified and tight asymptotic bounds on the methods below as a function of the value of **N**, the length of the input arrays. If possible, give a Big-Theta bound for the overall runtime. Otherwise, provide a Big-Theta bound for both the best case and worst case runtime. **Clearly mark your final answer by placing a box around it.**

Code	Bound
`int[] convoluted(int[] a, int[] b) {` `  assert a.length == a.length;  // Assume true` `  int[] result = new int[a.length];` `  for (int i = 0; i < result.length; i++) {` `  for (int j = 0; j <= i; j++) {` `  result[i] += a[j] * b[i];` `  }` `  }` `  return result;` `}`	$\Theta(N^2)$
`boolean debugBinarySearch(int[] a, int target) {` `  int start = 0;` `  int end = a.length - 1;` `  while (start <= end) {` `  int mid = ((end - start) / 2) + start;` `  if (a[mid] == target) {` `  return true;` `  } else if (a[mid] < target) {` `  start = mid + 1;` `  } else {` `  end = mid - 1;` `  }` `  System.out.print("Searching: [ ");` `  for (int i = start; i <= end; i++) {` `  System.out.print(a[i] + " ");` `  }` `  System.out.println("]");` `  }` `  return false;` `}`	Worst Case: $\Theta(N)$ (This is a geometric sum: $1 + 2 + 4 + \ldots + N \sim= 2N$) Best Case: $\Theta(1)$

3. Volskaya Industries (4 pts)

a. Baddice Einer wants to update his `SuperArray` data structure to add the `trim()` method, which makes null values at the end of the array disappear, similar to trimming an `ArrayList` down to size. For example, `{1, 2, null, null}` would trim to `{1, 2}`.

However, there's one issue. If the array is fragmented, which means there are null values in between non-null values rather than just at the end (e.g. `{1, null, 2, 3}`), trimming the array will not remove all the null values. Help him `throw` a `FragmentationException` with an error message when this happens. This exception should be handled by the user of `SuperArray` and should not directly cause the program to exit. (You may not need all lines.)

```
public class SuperArray {
    private Object[] arr;

    public SuperArray(int size) { arr = new Object[size]; }
    public int length() { return arr.length; }
    public Object get(int i) { return arr[i]; }
    public void set(Object o, int i) { arr[i] = o; }

    public class FragmentationException extends Exception {
        public FragmentationException (String msg) {
            super(msg);
        }
    }
    public void trim() throws FragmentationException {
        boolean fragmented = false;
        int trim_to = -1;
        // Finds if the array is fragmented. Assume this works properly.
        for (int i = arr.length - 1; i >= 0; i--) {
            if (arr[i] != null && trim_to == -1) trim_to = i;
            if (trim_to != -1 && arr[i] == null) fragmented = true;
        }
        if (fragmented) {
            String messageToPrint = "OMGZOR Fragmentation!!!!";
            throw new FragmentationException(messageToPrint);
        }
        arr = Arrays.copyOfRange(arr, 0, trim_to);
    }
}
```

b. Now Baddice Einer wants to test his latest feature. Fill in the JUnit test on the next page that verifies the behavior of `trim()` when the array is fragmented. The test should fail if anything except a `FragmentationException` is thrown, or if nothing is thrown. You may not need all lines.
Note that you can use the `fail(String msg)` method to instantly cause a JUnit assertion error.

```
import static org.junit.Assert.*;
import org.junit.Test;
...
    @Test
    public void testFragmentedTrim() {
        SuperArray s = new SuperArray(4);
        s.set("hi", 2); // Calling trim() on this array should throw the exception
        try {
            s.trim();
            fail("Calling trim should have thrown an exception");
        } catch (FragmentationException e) {
            return;
        }
    }
...
```

4. Route 66 (2 pts)

Write a method `flatten` that takes in a 2-D array x and returns a 1-D array that contains all of the arrays in x concatenated together. For example, `flatten({{1, 2, 3}, {}, {7, 8}})` should return `{1, 2, 3, 7, 8}`. You may not need all the lines below.

```
public static int[] flatten(int[][] x) {
    int totalLength = 0;
    for (int i = 0; i < x.length; i++) {
        totalLength += x[i].length;
    }
    int[] a = new int[totalLength];
    int aIndex = 0;
    for (int i = 0; i < x.length; i++) {
        for (int j = 0; j < x[i].length; j++) {
            a[aIndex] = x[i][j];
            aIndex++;
        }
    }
    return a;
}
```

5. Ilios (Class-y Critic) (2 pts)

Optional Story: After a long and arduous journey through his undergraduate years studying computer science, Alan has decided he has had enough. He has chosen to abandon his aspirations for a career in CS to pursue his lifelong dream of becoming a food critic. Despite his sudden change of heart, Alan knows that the skills he has acquired over the past four years could come in handy and decides that his first contribution to the field of culinary analysis will be a brand new, revolutionary platform for rating seafood restaurants: Kelp.

Instructions start: Consider the classes below for the duration of this question.

```
public class Review {
    private String reviewer;
    protected String content;

    public Review(String user, String content) {
        this.reviewer = user;
        this.content = content;
    }

    public String getContent() {
        return this.content;
    }
    public void edit(String newContent) {
        this.content = newContent;
    }
}

public class StarredReview extends Review {
    private int stars;

    public StarredReview(String user, String content, int stars) {
        super(user, content);
        this.stars = stars;
    }

    public String getContent() {
        return "Rating: " + this.stars + ", " + ((Review) this).getContent();
    }
    public String getContentText() {
        return this.content;
    }
    public void edit(String newContent, int stars) {
        super.edit(newContent);
        this.stars = stars;
    }
}
```

Consider the following test program.

```
public static void exciteReview(Review r) {
    StarredReview review = (StarredReview) r;
    review.edit(review.getContentText().toUpperCase(), 10000);
}

public static void main(String[] args) {
    Review r1 = new Review("Sarah", "Eggcellent!");
    exciteReview(r1);                              // (1)
    System.out.println(r1.getContent());           // (2)

    StarredReview r2 = new StarredReview("Antares", "Dude this place is lit", 5);
    exciteReview(r2);                              // (3)
    System.out.println(r2.getContent());           // (4)
    System.out.println(r2.getContentText());       // (5)
}
```

For each line with a number, write what should be printed on the corresponding blank below. If the line causes a compile error, write "Compile error". If the line causes a runtime exception, write "Exception". If the line will not produce any output, write N/A.

If a line causes an a compile-time or runtime error, assume the line is removed when determining what happens for the other lines of the `main` function.

(1) Runtime error: ClassCastException

(2) Eggcellent!

(3) N/A

(4) Runtime error: StackOverflowException

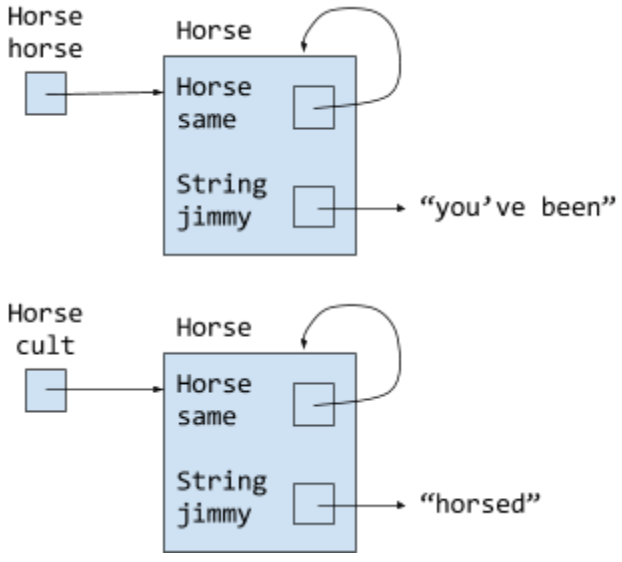
(5) DUDE THIS PLACE IS LIT

6. Temple of Anubis (Samehorse) (4 pts)

a. Given the following program, write the output of running the `main` method in the box on the right.

Code	Output
`public class Horse {` `  Horse same;` `  String jimmy;` `  public Horse(String lee) {` `  jimmy = lee;` `  }` `  public Horse same(Horse horse) {` `  if (same != null) {` `  Horse same = horse;` `  same.same = horse;` `  same = horse.same;` `  }` `  return same.same;` `  }` `  public static void main(String[] args) {` `  Horse horse = new Horse("you've been");` `  Horse cult = new Horse("horsed");` `  cult.same = cult;` `  cult = cult.same(horse);` `  System.out.println(cult.jimmy);` `  System.out.println(horse.jimmy);` `  }` `}`	`horsed` `you've been`

b. Draw the box and pointer diagram that results after the `main` method has executed:



7. Lijiang Tower (4 pts)

Professor Saralahan Kyaoim at Harbuvard University decided to improve the miserable indexing time for the usual singly linked list by adding more pointers to each node. You, as a faithful assistant, decide to help the professor implement the `FastList` class.

Each `FastList` node keeps up to 3 pointers stored in a `nexts` array. Each node has a pointer to the next node, every 5th node (0, 5, 10, 15, ...) has a pointer to the next 5th node, and each 10th node (0, 10, 20, 30, ...) has a pointer to the next 10th node. For example, the 0th node will have pointers to the 1st, 5th, and 10th nodes (if they exist). The 2nd node will only have a pointer to the 3rd node (if it exists). The 5th node will have pointers to the 6th and 10th nodes (if they exist). Fill in the following lines to meet the specification. You may not need all lines.

```
public class FastList {
    private Node head;
    public static class Node {
        Object item;
        Node[] nexts;  // Structure: {next node after, 5th node after, 10th node after}
        public Node(Object item) {
            this.item = item;
            this.nexts = new Node[3];
        }
    }
    /** Returns the item in the list at INDEX. Returns null if invalid index
     *  (e.g. negative index, index greater than or equal to list size). */
    public Object get(int index) {
        if (head == null || index < 0) {
            return null;
        } else {
            return get(head, index);
        }
    }
    /** An efficient solution will make the fewest possible recursive calls. */
    private static Object get (Node node, int index) {
        if (index == 0) {
            return node.item;
        }
        if (index >= 10 && node.nexts[2] != null) {
            return get(node.nexts[2], index - 10);
        } else if (index >= 5 && node.nexts[1] != null) {
            return get(node.nexts[1], index - 5);
        } else if (node.nexts[0] != null) {
            return get(node.nexts[0], index - 1);
        } else {
            return null;
        }
    }
}
```

8. Watchpoint: Gibraltar (3 pts)

Consider the following `SafeArray` class, which represents an array of `Objects` but does not have to handle null cases when being used. One operation we could use on our `SafeArray` is the `map` operation, which applies a function f: Object → Object to every element of the array and replaces it in-place.

We represent functions as follows:

```
public class Function {
    public Object apply(Object o) {
        return "Not Implemented";
    }
}
```

For a concrete example, we'll work with `ExplosionFunction`. Notice how our functions do not require any null checking.

```
public class ExplosionFunction extends Function {
    @Override
    public Object apply(Object o) {
        return "EXPLOSION!! " + o.toString();
    }
}
```

We use a `Container` to hold every element of the array. Fill in the code below so that the constructor and `map` successfully handle null objects in the input array so that calling apply on them causes nothing to happen.
Note: There are multiple ways to solve this problem. You may not need to use all the blank lines.

```
public class SafeArray {
    /* The following line is optional, and the problem may be completed without it.
     * However, it makes the solution more elegant. */

    private static final NullContainer NULL = new NullContainer();
    Container[] arr;

    static class Container {
        Object o;
        public Container() {}
        public Container(Object o) {
            this.o = o;
        }
        public void apply(Function f) {
            o = f.apply(o);
        }
        public Object get() {
            return o;
        }
    }
    ... (continued on next page)
```

```
    static class NullContainer extends Container {
        public NullContainer() {
            super();
        }
        public void apply(Function f) {
            return;
        }
    }

    public SafeArray(Object[] input) {
        arr = new Container[input.length];
        for (int i = 0; i < input.length; i++) {
            if (input[i] == null) {
                arr[i] = NULL;
                // Alternate solution without using NULL: arr[i] = new NullContainer();
            } else {
                arr[i] = new Container(input[i]);
            }
        }
    }

    public void map(Function f) {
        for (int i = 0; i < arr.length; i++) arr[i].apply(f);
    }

    public Object get(int index) {
        return arr[index].get();
    }

    public static void main(String[] args) {
        Object[] input = {null, "Heroes", null, "never", "die!", 9001, null, null};
        SafeArray a = new SafeArray(input);
        a.map(new ExplosionFunction());
        System.out.println(a.get(1));
        System.out.println(a.get(3));
        System.out.println(a.get(4));
        System.out.println(a.get(5));
    }
}
```

The output of running `java SafeArray` should be

```
EXPLOSION!! Heroes
EXPLOSION!! never
EXPLOSION!! die!
EXPLOSION!! 9001
```