UC Berkeley – Computer Science
CS61BL: Data Structures
Midterm 2, Summer 2017

This test has 9 questions worth a total of 45 points, and is to be completed in 110 minutes. The exam is closed book, except that you are allowed to use two double-sided page of notes as a cheat sheet (front and back). 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 above statement. Once the exam has started, write your class ID in the corner of every page.

Name: _____   **Your Class ID:** _____

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.'
- ○ indicates that only one circle should be filled in.
- □ indicates that more than one box may be filled in.
- For answers that involve filling in a ○ or □, **please fill in the shape completely.**

Optional. Mark along the line to show your feelings Before exam: [☹_____☺].
on the spectrum between ☹ and ☺.   After exam: [☹_____☺].

Reference Material

```
/** Collections */
All Collections implement Iterable and have size(), isEmpty() and stream()
methods.

public class HashSet<T> {
    void add(T item);
    boolean contains(T item);
}

public class TreeSet<T extends Comparable<T>> {
    void add(T item);
    boolean contains(T item);
}

public class LinkedList<T> implements Queue<T> {
    void addFirst(T item);
    void addLast(T item);
    T get(int index);
    T removeFirst();
    T removeLast();
}

public class Stack<T> {
    void push(T item);
    T pop();
}

public interface Queue<T> {
    void offer(T e);
    T poll();
    T peek();
}
```

```
/** Maps */

public class HashMap<K, V> {
    void put(K key, V value);
    V get(K key);
    boolean containsKey(K key);
}

public class TreeMap<K extends Comparable<K>, V> {
    void put(K key, V, value);
    V get(K key);
    boolean containsKey(K key);
}

/** Streams */

public interface Stream<T> {
    Stream<T> filter(Predicate<? super T> predicate);
    <R> Stream<R> map(Function<? super T, ? extends R> mapper);
    Optional<T> reduce(BinaryOperator<T> accumulator);
    Stream<T> sorted(Comparator<? super T> comparator);
    void forEach(Consumer<? super T> action);
    <R, A> R collect(Collector<? super T, A, R> collector);
}

public class Collectors {
    static <T> Collector<T, ?, List<T>> toList();
    static <T> Collector<T, ?, Set<T>> toSet();
    static <T,K> Collector<T, ?, Map<K, List<T>>>
        groupingBy(Function<? super T, ? extends K> classifer);
}

public class Optional<T> {
    boolean isPresent();
    T orElse(T other);
}
```
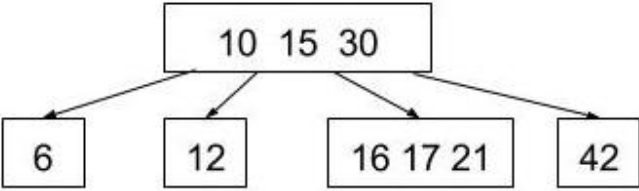
1. Stark: Winter is Coming (6 pts)

a. Draw the result of deleting 10 from the **BST** below using the standard deletion algorithm outlined in lab.

Original	delete(10)

b. Draw the **splay tree** below after `find(7)` has been called.

Original	find(7)

c. Draw the result of inserting 20 into the following **2-3-4 tree** using the insertion algorithm discussed in lab. Afterwards, draw the **red-black tree** that corresponds to the resulting 2-3-4 tree. **Clearly mark red nodes with the letter R to the right of the number.**

Original	insert(20)

Red-black tree that corresponds to the 2-3-4 tree after insert(20)

2. Tully: Family, Duty, Honor (6 pts)

Consider the following "real-world" situations. For each situation, select the data structure that would be best suited in terms of performance and ease-of use, taking into consideration the specific types of inputs listed in each problem. Give a **brief and succinct justification** (**2 lines or less**) for why that data structure is best suited. You may assume the LinkedList is a circular doubly-linked list with a sentinel node. Using Big-Theta notation, give the worst-case runtime bound of using your data structure with respect to the variables defined in each part. If there is more than one possible answer, choose **only one** data structure.

a. Kai has been tasked with scheduling autograder jobs in the order in which they arrive. He receives jobs one by one from the Professor Pretentious and must be able to provide the next job to the autograder when one is requested, removing that job from the pool of waiting jobs in the process. You may assume that there are never more than **N** jobs waiting at once. Choose the best data structure to store the waiting jobs in.

○ Array	○ LinkedList	○Red-Black Tree	○Splay Tree	○ HashSet

Justification:

_____

_____

Runtime (Schedule a job): _____ **Runtime (Get next job to run):** _____

b. Sarah has become the CEO of MeTube and wants a way to store the number of visits that various websites have. She wishes to construct a data structure that contains her websites and query this data structure for the top **K** most-visited websites. At any given moment, she may receive a new website along with its number of visits, thus forcing Sarah to update the data structure. Once a site is added, its number of visits cannot be updated. Choose the best data structure to store the websites. You may assume there are never more than **N** websites indexed at once and that query operations are much more common than add operations.

○ Array	○ LinkedList	○Red-Black Tree	○Splay Tree	○ HashSet

Justification:

_____

_____

Runtime (Add one website): _____ **Runtime (Query):** _____

c. You are in charge of storing a set of **N** student names on *PandaGrade*. When a user logs into *PandaGrade*, you must check that the user's name is in the set of student names. A small set of *PandaGrade* users typically make up the vast majority of login requests (since a few students use *PandaGrade* to submit a lot of regrade requests). In addition to checking for membership of users' names in the set of student names, we'd like to be able to produce a sorted list of student names. Choose the best data structure to handle these operations.

○ Array	○ LinkedList	○Red-Black Tree	○Splay Tree	○ HashSet

Justification:

_____

_____

Runtime (Check that user is a student): _____   **Runtime (Get sorted student list):** _____

d. Kevin wants to be more spontaneous. So, he decides to make sure that he never visits the same `Restaurant` twice. Choose a data structure to store each new `Restaurant` that Kevin visits. At any time, Kevin must be able to check if a given `Restaurant` has been visited. You can assume that Kevin will never visit more than **N** `Restaurant`s.

○ Array	○ LinkedList	○Red-Black Tree	○Splay Tree	○ HashSet

Justification:

_____

_____

Runtime (Insertion): _____   **Runtime (Query):** _____

Chocobo says hi

3. Arryn: As High As Honor (3 pts)

Consider the `Widget` class below. You may assume the `HashSet` class uses external chaining and the bucket-indexing formula discussed in class. `HashSet` stores no duplicates. `HashSet` has a maximum load factor of .75; it doubles in size when there are more than .75 items per bucket on average in the set.

```java
public class Widget {
    Integer doohickey;

    public Widget(Integer doohickey) {
        this.doohickey = doohickey;
    }

    public int hashCode() {
        return 1;
    }

    public boolean equals(Object o) {
        return true;
    }

    public static void hashParty(int M) {
        HashSet<Widget> pad = new HashSet<>();
        for (int i = 0; i < M; i++) {
            pad.add(new Widget(i));
        }
    }
}
```

a. Using Big-Theta notation, provide the worst-case runtime bound for `hashParty` in terms of its argument **M**.

Worst case: _____

b. Now, suppose we replace `Widget's equals` method above with the following:

```java
public boolean equals(Object o) {
    return false;
}
```

Using Big-Theta notation, provide the new worst-case runtime bound for `hashParty`, again in terms of **M**. The new runtime may be the same as before.

Worst case: _____

c. Finally, suppose we replace `Widget's equals` and `hashCode` methods as follows:

```
public int hashCode() {
    return this.doohickey;
}

public boolean equals(Object o) {
    return false;
}
```

Using Big-Theta notation, provide the new worst-case runtime bound for `hashParty`, again in terms of **M**. The new runtime may be the same as before.

Worst case: _____

Designated Drawing Space

Maybe take a second to breathe and draw a Moogle for the Moogle below. He wants a friend.

4. Tyrell: Growing Strong (4 pts)

Consider the `Patriot` class below:

```java
public class Patriot {
    public String name;
    public int intellect;
    public int wisdom;
    public Patriot(String name, int i, int w) {
        this.name = name; this.intellect = i; this.wisdom = w;
    }
    public int hashCode() {
        return intellect;
    }
    public boolean equals(Object o) {
        if (o instanceof Patriot) {
            Patriot other = (Patriot) o;
            return intellect == other.intellect
                && name.equals(other.name);
        }
        return false;
    }
}
```

You now execute the following lines of code:

```java
Patriot ham = new Patriot("Hamilton", 32, 15);
Patriot burr = new Patriot("Burr", 19, 10);
Patriot adams = new Patriot("Adams", 24, 12);
HashMap<Patriot, Patriot> enemyMap = new HashMap<>();
enemyMap.put(ham, burr);
enemyMap.put(burr, ham);
enemyMap.put(adams, ham);
```

Finally, you now consider the following four code snippets, each executed **independently** after the snippet above. For each snippet, write what is printed. If the snippet produces a `NullPointerException`, write *`NullPointerException`* and *justify your answer. No justification is necessary if a* `NullPointerException` *is not thrown*. None of the following code snippets produce compiler errors, nor do they produce runtime errors other than `NullPointerException`.

You may assume the `HashMap` class uses external chaining and the bucket-indexing formula discussed in class. It stores no duplicates. `HashMap` initially has **16 buckets and a load factor of .75**. The `HashMap` doubles in size when there are more than .75 items per bucket on average in the table. You may assume `HashMap's get` method returns `null` when the provided key argument is not in the map.

```
ham.intellect = 33;
System.out.println(enemyMap.get(burr).name);
```

Print Output: _____

Justification: _____

```
adams.intellect = 26;
System.out.println(enemyMap.get(adams).name);
```

Print Output: _____

Justification: _____

```
ham.wisdom = 8;
System.out.println(enemyMap.get(ham).name);
```

Print Output: _____

Justification: _____

```
ham.intellect = 16;
System.out.println(enemyMap.get(ham).name);
```

Print Output: _____

Justification: _____

0. PNH (0 pts)

This author won the Nobel Prize in Literature for creating "an imagined world, where life and myth condense to form a disconcerting picture of the human predicament today."

5. Baratheon: Ours is the Fury (5 pts)

For each of the methods below, bound the runtime of the method using Big-Theta notation in terms of the input N. If not possible, write N/A. For full credit, your answer should be as simple as possible with no unnecessary leading constants or lower order terms.

```
_____   private static void f(int N) {
                 if (N < 1) {
                     return;
                 }
                 f(N/2);
             }


_____   private static void g(int N) {
                 if (N < 1) {
                     return;
                 }
                 g(N - 1);
             }


_____   private static void h(int N) {
                 if (N < 1) {
                     return;
                 }
                 h(N/2);
                 h(N/2);
                 h(N/2);
                 h(N/2);
             }


_____   private static void combined(int N) {
                 for (int i = 1; i < N; i *= 2) {
                     constant(N); // runs in constant time with respect to input
                     linear(N); // runs in linear time with respect to input
                     quadratic(N); // runs in quadratic time with respect to input
                 }
             }


_____   private static void challenge(int N) {
                 for (int i = 1; i < N; i *= 2) {
                     // runs in logarithmic time with respect to input
                     logarithmic(i);
                 }
             }
```

6. Lannister: Hear Me Roar (3 pts)

Consider the heavily redacted `Commit` class below. `Commit` objects contain a `nameToSHA HashMap`. `nameToSHA` contains a mapping of the names of the files tracked by this `Commit` to the hashes of the blob backups corresponding to those files. <u>You may not assume anything about `Commit's` instance variables or methods other than that which is provided below</u>. Fill in the `isFilePresent` method so that the method behaves according to its comment. You may assume the `Utils` class is available to the `Commit` class (via import or otherwise).

```
public class Commit {
    ...
    /**
     * A map of filename strings to the SHA-1 strings of those files'
     * contents.
     */
    public HashMap<String, String> nameToSHA;

    /**
     * Returns true if
     * a) given filename is tracked by this commit and
     * b) given contents matches the contents of this commit's blob for filename
     * with high probability.
     * Otherwise, returns false.
     */
    public boolean isFilePresent(String filename, byte[] contents) {
        if (_____) {

            _____;

            _____;

        }

        _____;

    }
}

public class Utils {
    ...
    /** Returns the SHA-1 hash of a byte array. */
    public static String sha1(byte[] content) { ... }
}
```

7. Greyjoy: We Do Not Sow (6 pts)

A `NestedList` is a list that contains either a `char`, or a `List` of `NestedLists`. Given a `NestedList`, complete the `DeepIterator`, an iterator that returns each `char` in the `NestedList` in order (essentially flattening the `NestedList`).

For example, given the `NestedList`: `[([e], [([x], [o])])]`, a series of `next()` calls should return `e`, `x`, `o`. `()` denotes a `List` and `[]` denotes a `NestedList`. A `NestedList` can be arbitrarily deep (i.e. you could have to go through many lists before you get to a `char`). You may assume that the user of the iterator will always call `hasNext()` before `next()`.

```java
import java.util.List;
import java.util.Stack;

public interface NestedList {

    /**
     * Returns true if this NestedList holds a single char,
     * false if it contains a list of NestedLists.
     */
    public boolean isChar();

    /**
     * Returns the single char that this NestedList holds,
     * if it holds a single char, otherwise returns null.
     */
    public Character getChar();

    /**
     * Returns the List of NestedLists that this NestedList holds,
     * if it holds a List. Otherwise, returns null.
     */
    public List<NestedList> getList();
}

public class DeepIterator implements Iterator<Character> {

    Stack<NestedList> stack = new Stack<>();

    public DeepIterator(NestedList list) {

        _____;

    }
```

```
    @Override
    public boolean hasNext() {

            _____;

    }


    @Override
    public Character next() {
        while(!stack.isEmpty()) {

            NestedList curr = _____;

            if(_____) {

                _____;

            }

            _____;

            for(_____;_____;_____) {

                _____;
                _____;

            }

        }
        return null;
    }
}
```

8. Martell: Unbowed, Unbent, Unbroken (6 pts)

a. You are given a list of `Users` as defined below. Using streams, complete the methods such that they work as specified by their comments. Each method requires only a single statement (one semicolon only).

```java
public class User {
    ...
    public User(int age, String name) { ... }
    public List<User> getFriends() { ... }
    public int getAge() { ... }
    public String getName() { ... }

    /** Returns a List of the names of Users with age > 18. */
    public static List<String> olderThan18(List<User> lst) {

        _____

        _____

        _____;

    }

    /** Returns a list of Users ordered from least to most friends. */
    public static List<User> orderedByPopularity(List<User> lst) {

        _____

        _____

        _____;

    }
}
```

b. Write a method, `functionReducer`, which takes in a `List` of `Functions` and returns a single `Function` that is the composition of all the `Functions` in the list. For example, if the list contained *f(x)*, *g(x)*, and *h(x)*, `FunctionReducer` should return a function that would give the results of *f(g(h(x)))*. If the input list is empty, `FunctionReducer` should return `null`. This method requires only a single statement (one semicolon only).

Recall that a `Function<T, R>` takes in an argument of type `T` and returns an argument of type `R`. It is a functional interface that has a single `apply` method. Lambda statements and method references can be used in place of `Functions`.

```java
public static <T> Function<T, T> functionReducer(List<Function<T, T>> lst) {

    return _____

        _____;

}
```

This page is left intentionally blank.

Continue forward young traveler.

9. Targaryen: Fire and Blood (6 pts)

A `TrimTree` is a binary tree that contains values of type `T`. When `TrimTrees` get too overgrown, they must be trimmed. We will trim a `TrimTree` **destructively** around the edges in a **bottom-up manner**.

Trimming a node **N** means removing both of its children and replacing **N**'s item with the result of the `TrimTree's combiner`, called with **N**'s children passed as arguments. It is only possible to trim a node **N** if both of its children are leaves. To determine if a given node should be trimmed, a `TrimTree` has an `isBetter` predicate which will test to see if the current item of **N** is better than the result of `combiner`. Notice that when we trim a node **N**, it becomes a leaf and thus makes **N**'s parent available for trimming.

Complete the `trim` method on the next page, which does the work of recursively trimming the subtree rooted at a node. You may not need all lines. Each line should contain only one statement.

```java
import java.util.function.BiPredicate;
import java.util.function.BinaryOperator;

public class TrimTree<T> {

    /** Has a test(T a, T b) method that returns true if a is better than b. */
    BiPredicate<T, T> isBetter;

    /**
     * Has an apply(T x, T y) method that returns some result of type T.
     * Input order does not matter.
     */
    BinaryOperator<T> combiner;
    TreeNode root;

    public TrimTree(BiPredicate<T, T> isBetter, BinaryOperator<T> combiner) {
        this.isBetter = isBetter;
        this.combiner = combiner;
    }

    private class TreeNode {
        T item;
        TreeNode left;
        TreeNode right;

        public TreeNode(T item, TreeNode left, TreeNode right) { ... }

        public TreeNode(T item) { ... }
    }
```

```
private boolean isLeaf(TreeNode node) {
        return node != null && node.left == null && node.right == null;
}


public void trim() {
    root = trim(root);
}

private void trim(TreeNode node) {

    if (_____) {

        _____;

    }

    _____;

    _____;

    if (_____) {

        _____;

        if (_____) {

            _____;

            _____;

            _____;

        }

    }

}

}
```