Midterm 2 Review Document

CS61BL Summer 2016

Antares Chen

born too late to explore the earth

born too early to explore the galaxy

born in time to explore dank memes

errata. tree problem

errata. <? super T>

Introduction

Let me preface this packet with the following statement: this packet is a *monster*. Don't even think **FOR A SECOND** that it would make sense to sit down and do all of it in one go. The purpose of this packet is to give you a compendium of targeted supplementary problems ranked by difficulty.

Like the first document,it reflects all material that you will have already seen in labs and lecture. Do not use this as a "be all end all" guide! It is still highly recommended that you review previous and external course material.

For example, you should still do MANY midterm 2s from previous semesters both 61BL and 61B. Use the midterms a heuristic for your knowledge of the material, then use the lab guides and textbook to relearn the material.

There are still three modes: (easy) which represents basic understanding which you should achieve after doing the lab, (medium) which consists of midterm difficulty level problems, and (hard) which has problems not meant to be trivially solvable!

REMEMBER if you're feeling down about things, take a step back and just breathe. Maybe take a walk, buy a soda and stress cook some turkey soup (trust me it's actually really cathartic). No matter what believe in yourself, and if you don't do that then at least believe in me who believes in you.

Organization

- The organization of this document is much like the MT1 review doc!
  - *Abstract Interfaces*
    - Material on using abstract classes and interfaces
  - *Iterators*
    - Material on using and defining iterators
  - *Generics*
    - Material on using generics
  - *Streaming Lambdas*
    - Material on higher order functions, lambda functions, streams and other miscellaneous Java 8 things.
  - *Tree Structures*
    - Includes generic practice problems for trees
  - *Binary Search Trees*
    - Material on binary search trees
  - *Balanced Search Trees*
    - Material on balanced search trees such as left leaning red-black trees, 2-3 tree, 2-3-4 trees, and tries
  - *Hashbrowns*
    - All things related to hashing

(Abstract Interfaces) Easy Mode

Warm-up Questions

1) What are the differences between abstract classes and interfaces? Why might we wish to use one over the other?

abstract classes → extends, can extend only one
for generalizing partial implementations
interfaces → implements, can implement many, a contract of methods

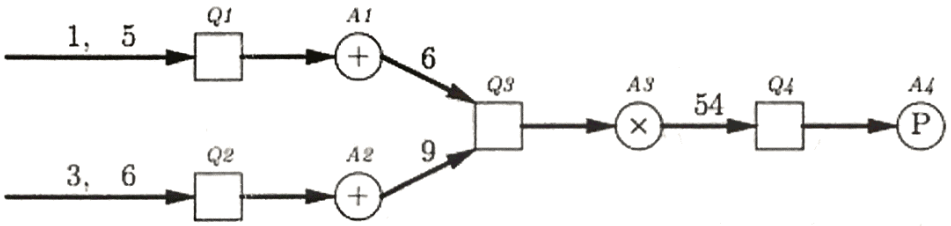
2) Why is it okay to write List<T> list = new ArrayList<>(); but not ArrayList<T> list = new List<T>();

List<T> is an interface. to use a variable of a type from an interface you need a concrete implementation such as ArrayList.

(Abstract Interfaces) Medium Mode

Computation Network (From FA15 MT1)

A simple computational network is composed of *value queues* (represented by squares in the example below) and *computation nodes* (circles in the example below). Both value queues and computation nodes have inputs and outputs. Integer values come into each value queue, which accumulates them until the computation node on its output indicates there are enough values for its computation. At that point, the value queue sends its accumulated values to its computation node, which computes a value that is sent to another value queue. In the example below, '+' nodes wait for two inputs and compute their sum; '×' nodes wait for two inputs and compute their product; and 'P' nodes wait for one input and print it (producing nothing):



We'll represent value queues with the class *ValueQueue* and describe computations with an interface called *Computation*. Here is *ValueQueue*:

```
import java.util.ArrayList;

public class ValueQueue {

    private Computation sink;
    private ArrayList<Integer> queue = new ArrayList<>();

    public void attach(Computation sink) {
        this.sink = sink;
    }

    public void accept(Integer value) {
        queue.add(value);
        if (sink.enabled(queue.size())) {
            sink.consume(queue);
            queue.clear();
        }
    }
}
```

two methods in interface!

And the code to set up the above network and inputs.

```
ValueQueue Q1 = new ValueQueue(), Q2 = new ValueQueue()
           Q3 = new ValueQueue(), Q4 = new ValueQueue();
Computation A1 = new Adder(Q3), A2 = new Adder(Q3),
           A3 = new Multiplier(Q4), A4 = new Printer();
Q1.attach(A1); Q2.attach(A2);
Q3.attach(A3); Q4.attach(A4);

Q1.accept(1); Q2.accept(3);
Q2.accept(6); Q1.accept(5);
```

Working backwards from the code, fill in the Suitable definition for the Computation interface and the Adder class

```
public interface Computation {
    void consume(ArrayList<Integer> list);
    boolean enabled(int length);
}
public class Adder implements Computation {
    private ValueQueue q;
    public Adder(ValueQueue q) {
        this.q = q;
    }
    public boolean enabled(int length) {
        return length >= 2;
    }
    public void consume(ArrayList<Integer> list) {
        int value = list.remove(0) + list.remove(1);
        q.accept(value);
    }
}
```

(Iterating Collections) Easy Mode

Warm-up Questions

1) ~~What are the three methods for iterators. What is the interface you need to implement?~~

implement Iterable → has method iterator() returning an Iterator : Iterator is an interface w/ hasNext(), next() remove()

2) It's bad for hasNext() to change the state of the iterator. How could hasNext() change the iterators state and why is it bad?

changes state → setting a value of an instance variable

bad because hasNext should only return status of iterator not change it.

3) Why is Collection an interface?

There are many Collections such as List, Set, ~~Order~~ SortedList, etc.

each has a different implementation.

Stack Times

Some of the operations in the Collection interface can be implemented generally without knowledge of the underlying Collection mechanics. To make it simpler, we make an abstract class that implements some of these functionalities.

```
public abstract class SimpleCollection<E> implements Collection<E> {

    /** The number of elements in this SimpleCollection. */
    private int size;

    public SimpleCollection() {
        size = 0;
    }

    /** Returns true if ELEM was added. */
    public abstract boolean add(E elem);

    /** Returns true if removing ELEM changed the collection. */
    public abstract boolean remove(E elem);

    /** Returns the size of this collection. */
    public int size() {
        return size;
    }

    /** Returns true if all elements in C were added. */
    public boolean addAll(Collection<? extends E> c) {
        boolean added = true;
        for (int i = 0; i < c.size(); i += 1) {
            added = added && add(c.get(i));
        }
        return added;
    }

    /** Returns true if the collection was changed. */
    public boolean removeAll(Collection<?> c) {
        boolean removed = false;
        for (int i = 0; i < c.size(); i += 1) {
            removed = removed || remove(c.get(i));
        }
        return removed;
    }

    // some more methods that I'm too lazy to write
}
```

Given this implementation of SimpleCollection, you are now to implement a Stack that is backed by an Array. Remember that a Stack only allows for adds and removes from the top of the stack. If remove is called with an element not at the top of the stack, you may throw an **IllegalArgumentException**

```
public class ArrayStack<E> extends SimpleCollection<E> {

    // some code may go here...
    private E[] data;
    private int currIndex;
    public ArrayStack() {
        data = new E[2];
        currIndex = 0;
    }
    public boolean add(E elem) {
        if (currIndex >= data.length) {
            resize();
        }
        data[currIndex] = elem
        currIndex += 1; return true;
    }
    public boolean remove(E elem) {
        ~~if (!elem.equals(~~
        if (currIndex <= 0) {
            throw new NoSuchElementException();
        } else if (!elem.equals(data[currIndex-1])) {
            throw new ~~NoSuch~~IllegalArgumentException();
        }
        currIndex -= 1;
        return true;
    }
}
```

```
    private void resize() {
        E[] newData = new E[data.length * 2];
        for(int i=0; i<data.length; i+=1) {
            newData[i] = data[i]; } data = newData; }
    ~~data = newData; }~~
```

(Iterating Collections) Medium Mode

ImageIterator

We can represent an image as a 2D array of Color objects (check the javadocs if you're interested). Now suppose we wish to sequentially process the image pixel by pixel, row by row. Code the ImageIterator to do just that.

```
public class ImageIterator implements Iterator<Color> {

    Color[][] image;
    int currX;
    int currY;

    public ImageIterator(Color[][] image) {
        this.image = image;
        currX = 0;
        currY = 0;
    }

    public void hasNext() {
        if (image.length == 0 || image[0].length == 0) {
            return false;
        }
        return currX < image.length && currX < image[currY].length;
    }

    public Color next() {
        Color value = image[currX][currY];
        currY = (currY+1) / image.length;
        currX = (currX+1) % image.length;
        return value;
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```

currY += (currX + 1) / image[currY].length

currX = (currX + 1) % image[currY].length

(Iterating Collections) Hard Mode

Repeaterator

The Repeaterator is an iterator that iterates through an int array, repeating each element the value number of times. A Repeaterator for [1, 2, 3] would return the sequence 1, 2, 2, 3, 3, 3. Fill out the following implementation for Repeaterator. Assume that the given array only holds positive numbers.

```
public class Repeaterator implements Iterator<Integer> {

    private int[] data;
    private int index;
    private int repeats;

    public Repeaterator(int[] array) {
        data = array; repeats = data[0]; index = 0;
        advance();
    }

    private void advance() {
        repeats -= 1;
        while (hasNext() && repeats==0){
            index += 1;
            repeats = data[index] - 1;
        }
    }

    public boolean hasNext() {
        return index < data.length;
    }

    public int next() {
        int prev = index;
        advance();
        return data[prev] - 1;
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```

(Streaming Lambdas) Easy Mode

Can You Read?

In Java 8, we can now pass Function objects as arguments and even define lambda functions. The following example allows us to easily express sorting in reverse order.

```
int[] arr = {1, 2, 3};
Arrays.sort(arr, (o1, o2) -> -(Integer.compare(o1, o2)));
System.out.println(Arrays.toString(arr)); // [3, 2, 1]
```

Streams allow us to compose a sequence of operations on each element of the stream. We can think of many Stream methods as analogous to their Function counterparts.

`Stream.map(mapper)`	`List.replaceAll(operator)`
`Stream.filter(predicate)`	`Collection.removeIf(!predicate)`
`Stream.forEach(action)`	`Iterable.forEach(action)`
`Stream.peek(action)`	`Iterable.forEach(action)`

Stream operations are stratified into three portions: creation, intermediate, and collection. You open a stream by calling stream(). To edit the stream, you perform intermediate operations such as map() and filter(). This will directly manipulate the data in the stream to create a new output stream. Finally, you close the stream by using some collector. For example forEach() or collect()

Warm-up Questions

1) What is the difference between == and .equals()? Why use one over the other?

relates
== ~~is~~ address identity while .equals ~~is~~ logical identity.
relates

Objects use .equals while primitives can use ==

(Streaming Lambdas) Medium Mode

Sort It Three Ways

You are given a list of strings and asked to sort the list by the length of each string. Write code that will perform this task in three different ways: using an anonymous class, using lambdas, and using streams.

anonymous class

```
public void sortMe(List<String> strings) {
    strings.sort(
    );
}
```

lambda

```
// you really need only one line
public void sortMe(List<String> strings) {
    strings.sort(
    );
}
```

stream

```
// you may not need all the lines
public List<String> sortMe(List<String> strings) {
    strings.stream()
    ;
}
```

(Streaming Lambdas) Hard Mode

The No Such Agency

As part of the larger MCDOUBLE project, the mysterious No Such Agency has effectively wiretapped a huge fraction of the internet, recording every page request and their associated IP addresses. After significant cross-referencing and pattern analysis, the No Such Agency has identified a select few IP addresses that require further processing to identify the true source. Apply the following transformations using Java Functions.

1. Filter out IP addresses that end in an even digit.
   You may find charAt(int index) and length() useful.
2. Replace each remaining IP address with the IP that can be found in data[first]. The network bounces requests through other addresses, so perform this process again except this time replacing the address with data[second].
   For example, the first address in darknet, 139.37.199.165, bounces to data[1], or 22.141.173.205, which then bounces to data[2], or 92.126.192.54. This new address replaces the original 139.37.199.165 in darknet.
3. Remove duplicates and identify the source addresses.
   Hint: Which data structure can help identify unique elements?
4. Print the resulting addresses.

```
String[] data = { "139.37.199.165" , "22.141.173.205" , "92.126.192.54"  ,
                  "92.130.25.157"  , "215.228.227.233", "253.104.111.68" ,
                  "132.57.118.180" , "202.134.155.208", "153.42.23.139"  ,
                  "224.253.18.171" };
```

```
ArrayList<String> danknet = new ArrayList<>(Arrays.asList(data));
danknet.removeIf(s -> (s.charAt(s.length()-1) - '0') % 2 == 0)
danknet.replaceAll(s -> data[data[(s.charAt(0) - '0')].
                                  charAt(1) - '0']);

Set<String> dankset = new HashSet<String>();
dankset.forEach(dankset::add);
dankset.forEach(System.out::println);
```

Now do the same thing using streams!

```
String[] data = { "139.37.199.165" , "22.141.173.205" , ... };
ArrayList<String> danknet = new ArrayList<>(Arrays.asList(data));
danknet.stream()
. filter (s -> (s.charAt(s.length()-1)-'0')%2 != 0)
. map (s -> data[data[(s.charAt(0)- '0')]
                  .charAt(1) - '0'])
. distinct()
. forEach ( System.out::println);
```

(Generics) Easy Mode

Warm-up Question

We spent a little time talking about generics in lab 10 and 11. Why the heck do we use generics?

because casting is a pain...

by using generics, we no longer have to cast Objects held by other collections.

Make This Generic

Rewrite the Node class below to allow for generic types

```
public class Node {

    Object item;
    Node next;

    public Node(Object item, Node next) {
        this.item = item;
        this.next = next;
    }

}
```

```
public class Node <T> {
    T item;
    Node <T> next;
    public Node( T item, Node <T> next) {
        this.item = item;
        this.next = next;
    }
}
```

(Generics) Medium Mode

NumericSet

Below is a snippet of *NumericSet*. For the code below, answer the following questions. As a hint, *Number* is the super class of *Double*, *Integer*, *Float*, etc.

```
public class NumericSet<T extends Number> extends HashSet<T> {
    /** implementation goes here */
}
```

1) What does the extends keyword do here?

Forces all elements of NumericSet have Number as a superclass.

2) What kind of types can you place in a NumericSet instance? Be specific in terms of the Objects existing within the java standard library. Javadocs may be helpful here!

Integer, Double, Float, Long, Short

Byte, BigInteger, BigDecimal, AtomicInt..., AtomicDec...

3) Why ~~does~~ is the generic type for HashSet <T> and not <T extends Number>?

The type local to NumericSet is already declared with "<T extends Number>. It now follows that the generic for HashSet needs also to be T but should not be redeclared.

Generic Binary Search Trees

Below is a snippet of *BinarySearchTree*. For the code below, answer the following questions.

```
class BinarySearchTree<T extends Comparable<T>> implements Comparable<T> {
    /** implementation goes here */
}
```

1) What kind of elements does an instance of BinarySearchTree hold?

Objects that implement comparable.

2) What method do all elements stored in a BinarySearchTree instance have in common?

compareTo(T other)

3) Is the Comparable<T> in the generic declaration for BinarySearchTree in any way related to the Comparable<T> implemented by the BinarySearchTree class?

nah.

(Generics) Hard Mode

Streams Lab Lol

Lol remember the stream lab? Good times. Answer the following questions for the snippet of code below.

```
public <R extends Comparable<R>> List<T> getOrderedBy(
                                    Function<T, R> getter) {
    // implementation goes here
}
```

1) Why do we need to declare a generic R in the method header and what method are objects of type R guaranteed to have?

we need to access R in the context of the method getOrderedBy. All Rs have compareTo

2) Where did we get the generic type T from?

The class declaration

3) What are the input and return types for getter?

input : T

return : R

The Fill Method

The Arrays class in Java utils has a fill method that has the following declaration.

```
public static <T> void fill(List<? super T> list, T x) {
    // implementation goes here
}
```

1) What are the types for both arguments?

list ⟶ List < ? super T >

x ⟶ T

2) What is the type that fill returns?

void → none. Still works because we mutate a list which is an object (pass by value)

3) What can you say about the type of objects List list holds?

List holds type T or any superclass of T.

4) Why didn't the library designers just write this as static <T> void fill(List<T> list, T x) { ... }

because it is possible that I want to fill a list <Number> with integers. Declaring like above would only allow me to with fill List<Int.> with integers.

Binary Search

The Arrays class also has a binary search method with the following declaration.

```
public static <T> int binarySearch(List<? extends Comparable<? super T>> list, T key) {
    // implementation goes here
}
```

1) What are the types for both arguments?

List<? extends Comparable<? super T>>

T

2) What is the return type for this method?

int : index of element

ask in comments section for more detail.

3) What can you say about the type of objects List list holds?

list holds objects implementing comparable

But there is also the <? super T>, thus allowing elements in the list to use compareTo parameterized by a supertype of T. Imagine if T was Integer, then list's objects may extend Comparable<Number>

Streams Lab Legendary Mode
What's going on this thre... OH LAWD

```
public static <T, K, A, D> Collector<T, ?, Map<K, D>> groupingBy(
                                Function<? Super T, ? extends K> classifier,
                                Collector<? Super T, A, D> downstream) {
    // the heck is going on here...
}
```

This is a legendary mode question. It's okay if you can't answer these questions.

1) The Collector class has three generics, what does each generic type do?

no

2) What are the input and return types for the Function classifier?

Solution

3) groupingBy also accepts a Collector downstream. What are the generic types for downstream? Why is it okay to have <? Super T>

for

4) What is the return type of groupingBy? What are the generics of that return type?

you

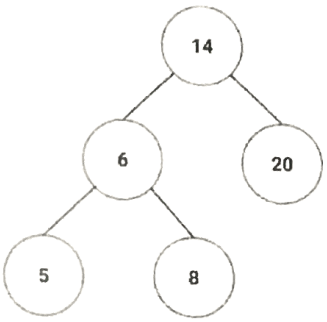
lol.

(Tree Structures) Easy Mode

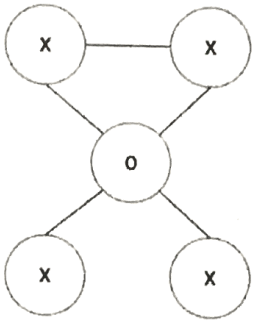
Warm-up Question?
How do you define a tree?

A tree is acyclic and fully connected or if there are n nodes, there are n-1 edges

Valid Trees
For each of the following images state if they are valid tree structures.

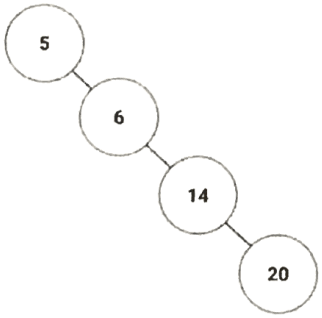


yes!



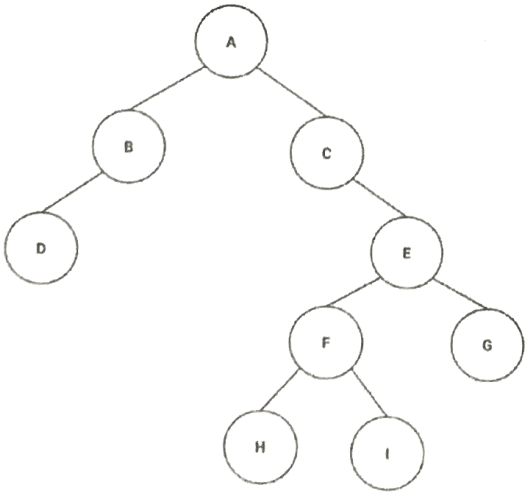
no!

- not acyclic
- two parents for a node.



yes!

Order Order Order



Inorder D B A C H F I E G

Preorder A B D C E F H I G

Postorder D B H I F G E C A

(Tree Structures) Medium Mode

Amoeba Sum

Consider we have an AmoebaFamily object except that Amoeba now holds an int value instead of a String name. The Amoeba inner class now looks a bit like this.

```
public static class Amoeba {

    public int value; // amoeba's name
    public Amoeba parent; // amoeba's parent
    public ArrayList<Amoeba> children; // amoeba's children

    public Amoeba(int value, Amoeba parent) {
        this.value = value;
        this.parent = parent;
        children = new ArrayList<Amoeba>();
    }

    // more code goes here
}
```

Your task is to write a method that will determine the sum of all values in an AmoebaFamily below.

```
/** Returns the int sum of all Amoeba values in FAM. */
public static ~~void~~ sumoeba(AmoebaFamily fam) {
    return helper(fam.root);


}

// space for helper methods if wanted
public static int helper(Amoeba node) {
    int value = node.value;
    for(Amoeba child: node.children) {
        value += helper(child);
    }
    return value;
}
```

Amoeba Search

Given an AmoebaFamily, write code that will search for an Amoeba with the given name.

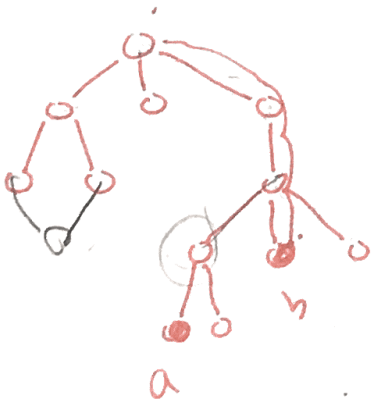
```
/** Returns true if FAM has an Amoeba with NAME as name. */
public static boolean findMoeba(AmoebaFamily fam, String name) {
    HELP
    return helper(fam.root, name);
}

// some space for helper methods
public static boolean helper(Amoeba node, String name){
    if (nam.equals(node.name)) {
        return true;
    }
    for (Amoeba child : node.children) {
        if (helper(child, name)) {
            return true;
        }
    }
    return false;
}
```

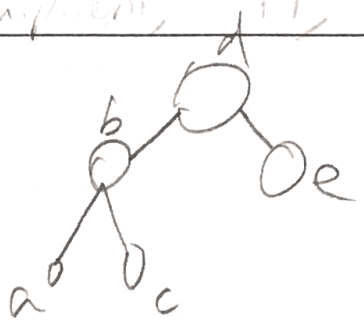
(Tree Structures) Hard Mode

Amoeba Path

Implement pathMoeba which returns the length of the shortest path between two Amoebas in an AmoebaFamily. Amoebas are identified by their name and the AmoebaFamily is guaranteed to have both Amoebas.

```
// go at it friends
public int pathMoeba(Amoeba a, Amoeba b) {
    if (b.equals(a)) {
        return 0;
    }
    Amoeba temp = b;
    for (int dist = 1; b.parent != null; dist += 1) {
        b = b.parent;
        if (a.name.equals(b.name)) {
            return dist;
        }
    }
    return pathMoeba(a.parent, temp) + 1;
}
```





(Binary Search Trees) Easy Mode

Define That BST Though

1) List the two properties that define a binary search tree.

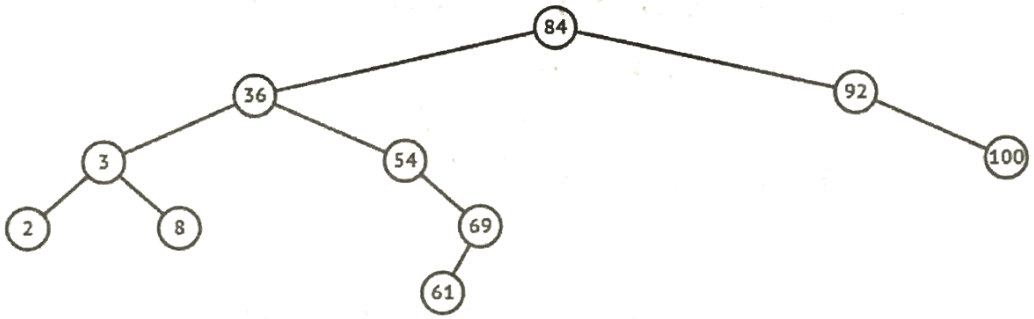
- left child is less than parent, right child is greater
- each node has at most two children.

2) For the purpose of this class, do we care about inserting two elements into a BST of the same value?

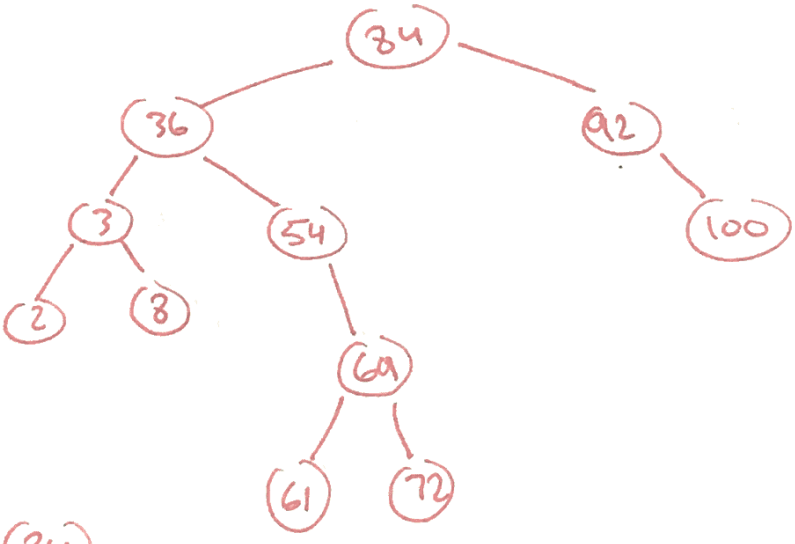
nope. avi

Do BST Things

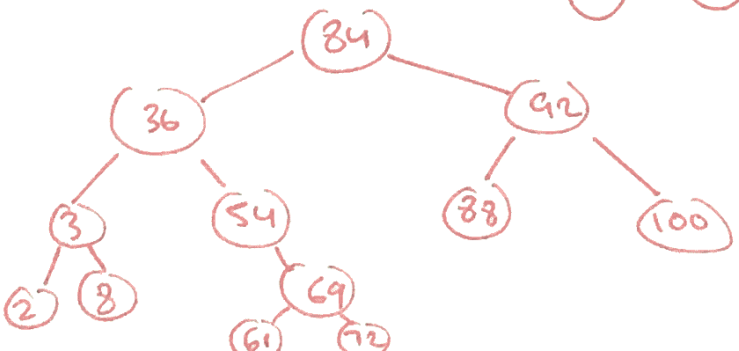
Given the following BST, perform the following operations.



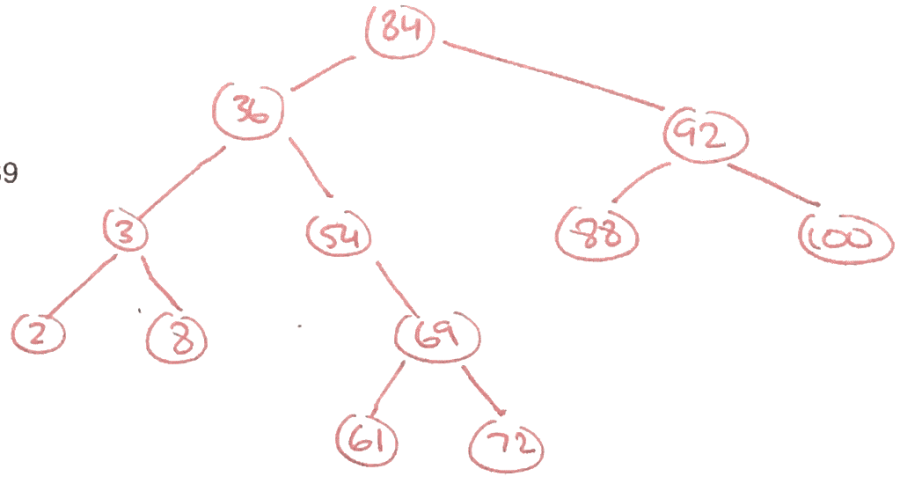
1) Insert 72



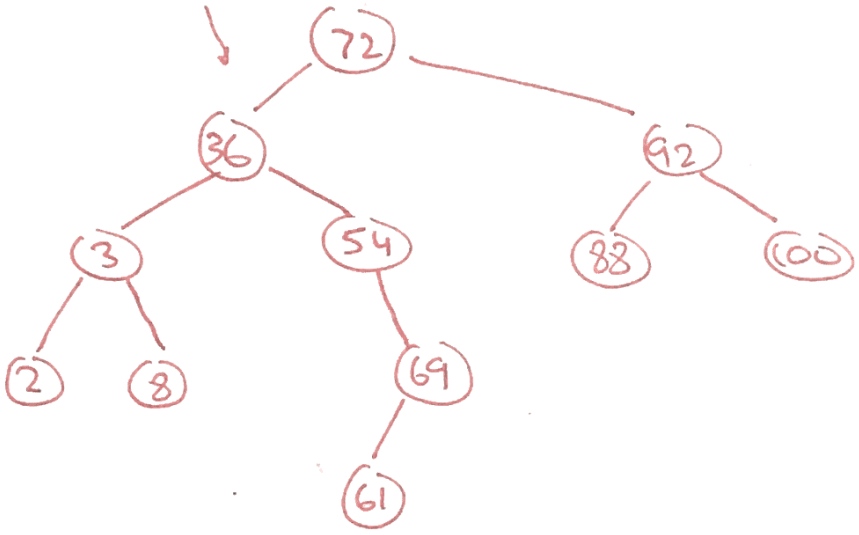
2) Insert 88



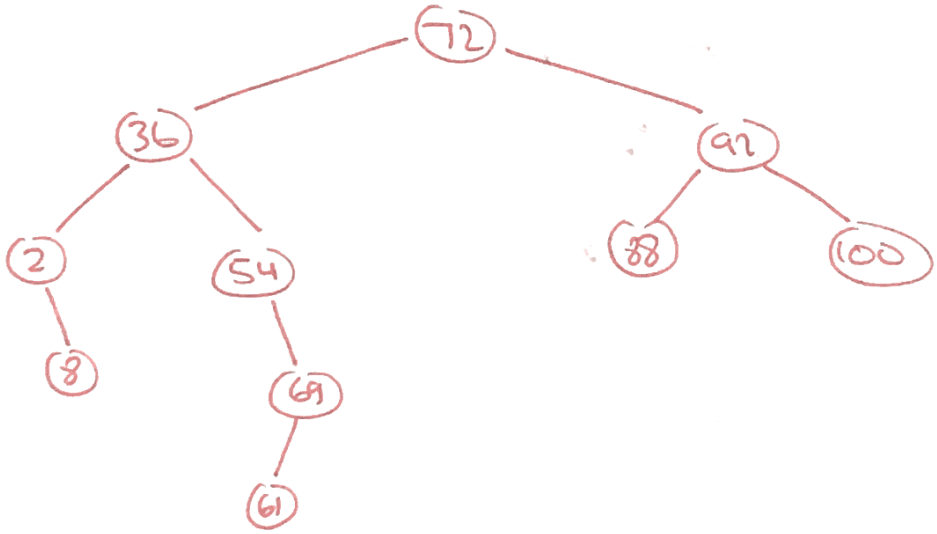
3) Insert 89



4) Remove 84 (use the left subtree)



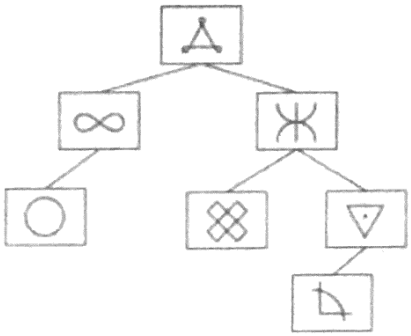
5) Remove 3



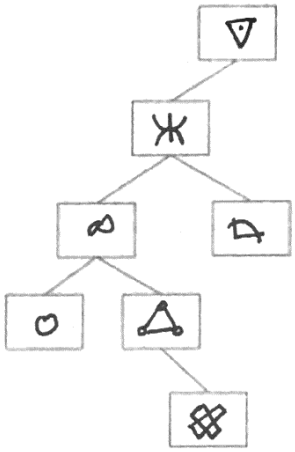
(Binary Search Trees) Medium Mode

Symbol Tree (Taken from SP16 MT2)

Consider the binary tree below. Each symbol has an underlying meaning. For example the root node may represent "snowman" while its immediate left child may represent "overthrow the capitalist regime"



1) Given the above symbols, fill out the following tree such that it is a valid BST on the underlying meaning of each symbol.



2) For each of the insertion operations below, use the information given to "insert" the element into the TOP TREE WITH PRINTED SYMBOLS, NOT THE TREE WITH YOUR HANDWRITTEN SYMBOLS by drawing the object (and any needed links) onto the tree. You can assume the objects are inserted in the order shown below. You should not change anything about the original tree; you should only add links and nodes for the new objects. If there is not enough information to determine where the object should be inserted into the tree, circle "not enough information". If there is enough information, circle "drawn in the tree above" and draw in the tree AT THE TOP OF THE PAGE.

insert():  >  (Drawn In Tree Above) Not Enough Information

insert():  >  Drawn In Tree Above (Not Enough Information)

insert():  <  <  (Drawn In Tree Above) Not Enough Information

insert():  <  <  Drawn In Tree Above (Not Enough Information)

(Binary Search Trees) Hard Mode

Linked Lists Are Back

In Lab 17, you converted a linked list into a balanced binary search tree. Now we're going to do the opposite. Convert a given Binary Search Tree into a sorted linked list. These two methods are placed into BinarySearchTree.java from lab 14.

```
/** Returns the head of a linked list created from BST rooted at NODE. */
public Node toLinkedList(TreeNode node) {
    node = helper(node);
    if (node != null) {
        while (node.left != null) {
            node = node.left;
        }
    }
    return node;
}

/** A helper method. */
private Node helper(TreeNode node) {
    if (node == null) {
        return node;
    }
    if (node.left != null){
        TreeNode left = helper(node.left);
        while (left.right != null) {
            left = left.right;
        }
        left.right = node;
        node.left = left;
    }
    if (node.right != null) {
        TreeNode right = helper(node.right);
        while (right.left != null) {
            right = right.left;
        }
        right.left = node;
        node.right = right;
    }
    return node;
}
```

(Balanced Search Trees) Easy Mode

Warm-up Questions

1) How do you define a Red-Black Tree?

- root node is black, every red node has at most two black children, every path from a node to its descendent leaf has same # of black nodes

2) How do you define a 2-3-4 Tree?

A search tree that has at most 3 keys per node and any non-leaf node has one more child than # of keys

3) Red Black Trees are to 2-3-4 Trees as Left Leaning Red Black Trees are to what?

2-3 trees

4) What is the difference between 2-3-4 Trees and 2-3 Trees?

2-3 trees do not have 4 nodes

Fill In the Table

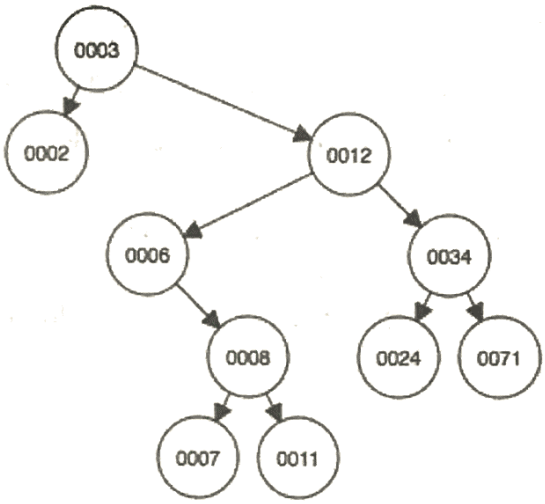
Write in the Big-O or Big-Theta time bounds for each cell in the table below.

	Binary Search Tree		Red-Black Tree	
	Best	Worst	Best	Worst
Find	$\Theta(1)$	$\Theta(N)$	$\Theta(1)$	$\Theta(\log N)$
Insert	$\Theta(1)$	$\Theta(N)$	$\Theta(\log N)$	$\Theta(\log N)$
Delete	$\Theta(1)$	$\Theta(N)$	$\Theta(1)$, $\Theta(\log N)$	$\Theta(\log N)$

(Balanced Search Trees) Medium Mode

Rotations For Days

For the tree below, perform the following operations in order.

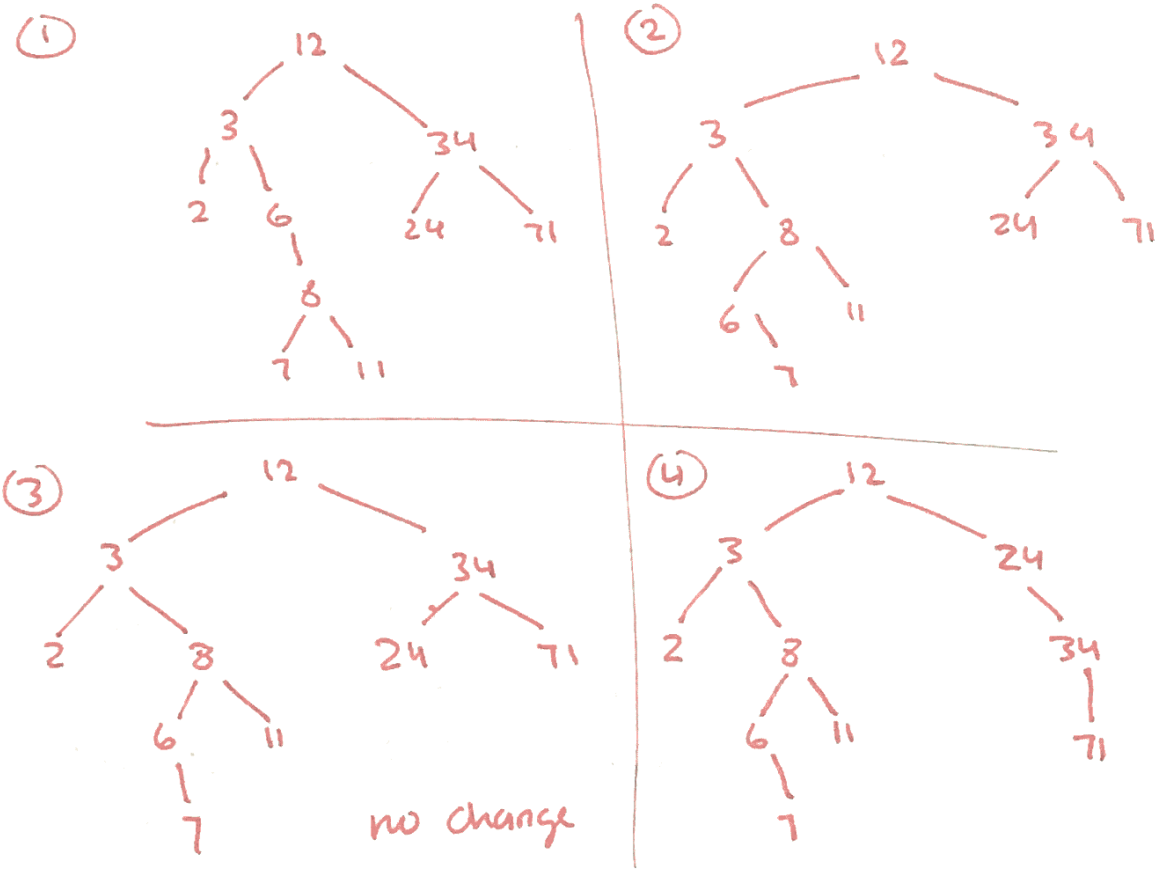


1) Rotate 3 left

2) Rotate 6 left

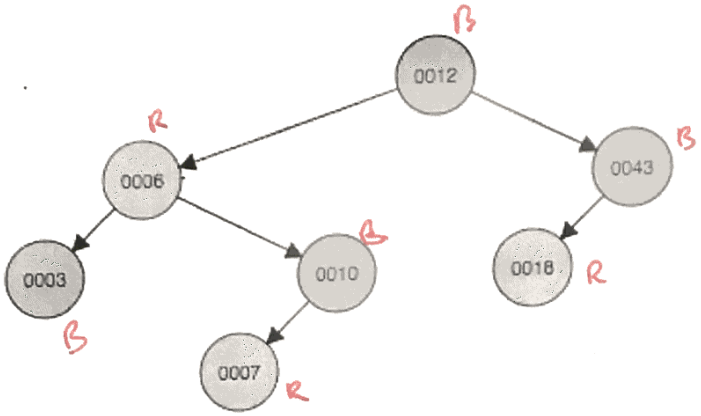
3) Rotate 2 right

4) Rotate 34 right

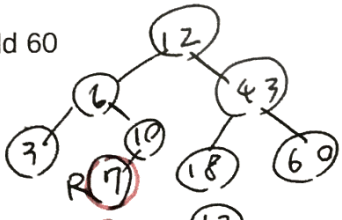


Add it To Me

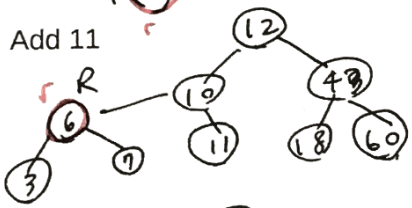
Perform the listed operation for the following Left Leaning Red-Black Tree. Draw out the tree for each.



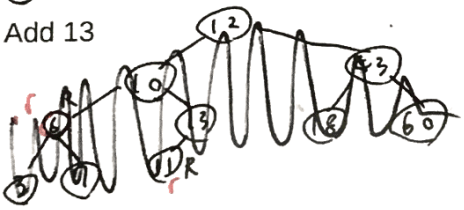
1) Add 60

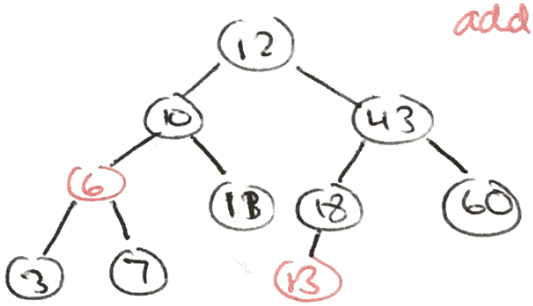


2) Add 11

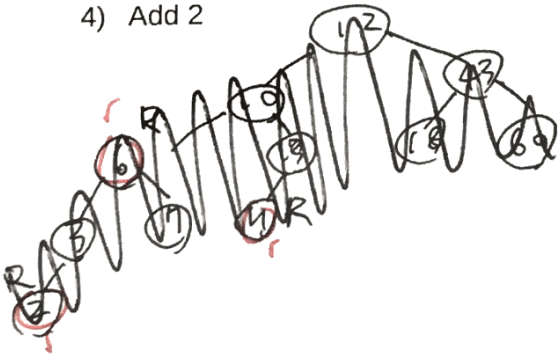


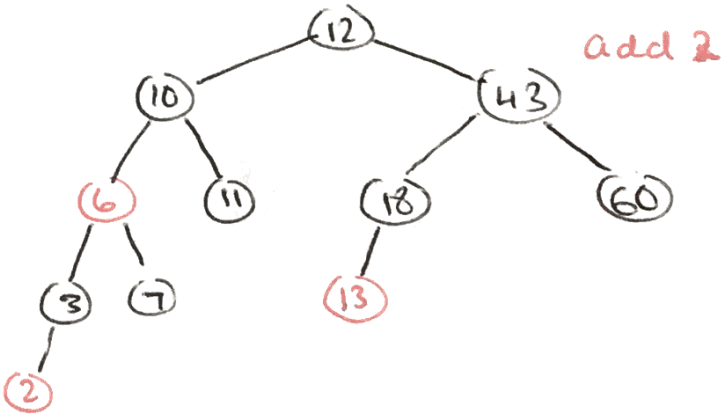
3) Add 13





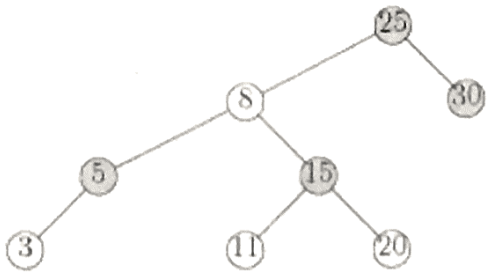
4) Add 2



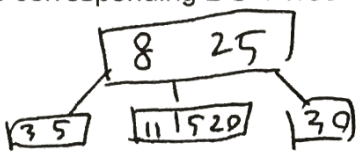


Conversion Times

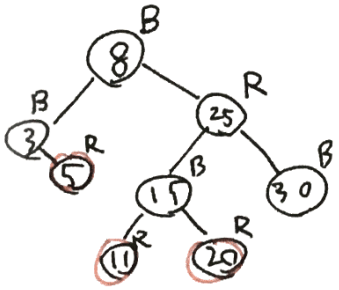
For the following red-black tree, complete the following operations (shaded nodes are black)



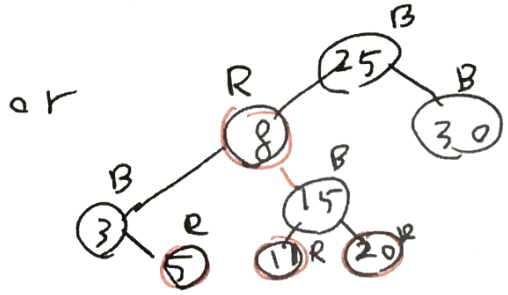
1) Create the corresponding 2-3-4 Tree



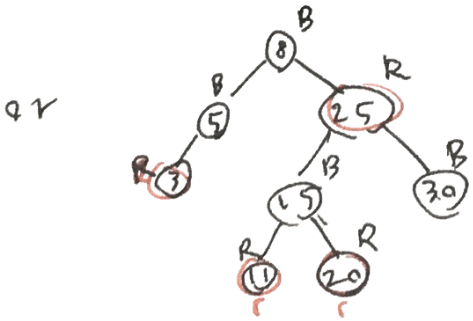
2) Create another Red-Black Tree that corresponds to that 2-3-4 Tree



or



or



(Balanced Search Trees) Hard Mode

Grab Bag Questions

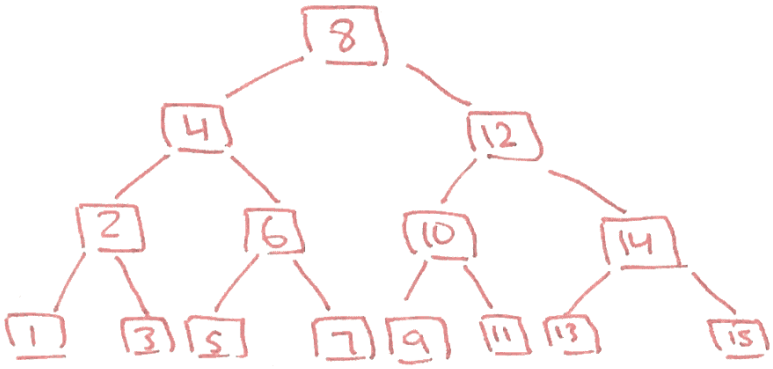
1) If a certain 2-3-4 Tree has height *h* meaning it has *h+1* levels, then what is the maximum and minimum height for the corresponding Red-Black Tree. Do not use asymptotics.

maximum = 2h+1

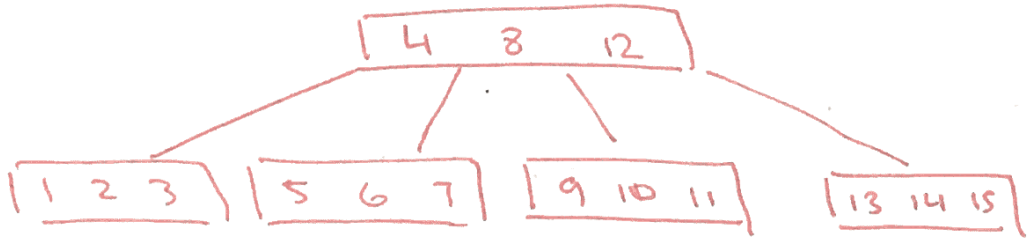
minimum = h

2) Show two 2-3-4 Trees containing values 1-15 having both minimum and maximum depth respectively.

max height.



min height



(Hashbrowns) Easy Mode

Get Ready For Hashbrowns

Answer the following questions that will test your understanding of hashing and java

1) What three tenants must a good hashcode follow?

1. deterministic 2. good distribution 3. two objects equal to eachother must have same hashcode.

2) What is the default hash java uses for any object?

hash via address.

3) When defining your own hash function for Java, what two methods must you override and why?

equals and hashcode to enforce behavior of tenant 3.

4) In lab 15, the first hashcode you wrote for strings involved hashing by the first letter of the string. Why is this a bad hashcode? Give a better hashing regime.

because all words w/ same first letter would collide!

5) Suppose you have a hash table with a perfect hashing function. What is the runtime for N insertions? What if the hash function is terrible?

$O(N)$ → good function, amortized!

$O(N^2)$ → bad hash function (always collide) (and inserts to end of chain).

---

actual java string hash

```
int hash = 0;
for (int i=0; i < str.length(); i+=1) {
    hash = hash * 31 + str.charAt(i);
}
return hash;
```

StringSet

The StringSet class defines a set for strings. The set is backed by a hash table that resolves collisions by chaining.

```
public class StringSet {
    /** The maximum load factor before resizing. */
    private static final double MAX_LOAD_FACTOR = 2;
    /** An array of hash table entries. */
    private Entry[] entries;
    /** The number of elements held in the set. */
    private int size;
    /** The load factor of the hash map. */
    private double load;

    class Entry {
        int key; String value; Entry next;

        Entry(int key, String value, Entry next) {
            this.key = key;
            this.value = value;
            this.next = next;
        }
    }

    /** Initializes a new StringSet with an initial size of SIZE. */
    public StringSet(int size) {
        this.entries = new Entry[size];
        this.size = 0;
    }

    /** Returns true if e exists in the set, adding e. Else
     *  returns false. */
    public boolean add(String e) {
        return put(e.hashCode(), e);
    }

    /** Returns true if e is contained in the set. */
    public boolean contains(String e) {
        return add(e);
    }
}
```

Now given the above code, implement the following functions.

```
/** Returns true if the key, value pair does not already exist
 *  in the hashmap and then places it in the map. */
private boolean put(int key, String value) {
    int index = ~~value~~ ~~key.hashCode()~~ % entries.length;
    Entry curr = entries[index];
    if (curr == null) {
        entries[index] = new Entry(key, value, null);
    } else {
        while (curr.next != null) {
            curr = curr.next;
        }
        curr.next = new Entry(key, value, null);}
}

/** If the load factor of the hash map is greater than
 *  MAX_LOAD_FACTOR then this method will double the size
 *  of the map. */
private void resize() {
    Entry[] old = entries;
    entries = new Entry[old.length * 2]; load
    load = 0;
    for (int i = 0; i < old.length; i += 1) {
        ~~for (int j = 0; j < old[i].length; j += 1)~~
        ~~while (curr != null)~~;
        for (Entry curr = old[i]; curr != null; curr = curr.next
            put(curr.key, curr.value);
        }
    }
}

// continued from above

size += 1;
load = size / entries.length;
if (load >= MAX_LOAD_FACTOR) {
    resize();
}
return true;
}
```

(Hashbrowns) Medium Mode

Hashing Binary Trees

Consider the implementation we used for BinaryTree in lab 14. It had one instance variable root which was of type TreeNode. A TreeNode instance had four instance variables T item, TreeNode left, TreeNode right, and int size.

We now wish to override Java's hashcode() function to hash BinaryTree instances. Fill in the blanks below.

```
public int hashcode() {
    return hashcodeHelper(___root_____);
}

// in java the ^ represents logical xor
private int hashcodeHelper(TreeNode node) {
    if (node == null ) {
        return 0;
    } else {
        return node.item. ^ hashcodeHelper(node.left) ^ hashcodeHelper(node.right);
    }
}
```

hashcode

Hashing Strings in Java (From the FA15 Final)

The hash function for Java's String class is as follows.

```
public int hashcode() {
    int h = 0;
    for (int i = 0; i < length(); i += 1) {
        h = 31 * h + charAt(i);
    }
}
```

1) Given a string of length *L* and a HashSet<String> containing N strings, give the worst and best-case running times of inserting String into the HashSet.

worst case => $\Theta(N)$

best case => $\Theta(L)$ if we treat L as a constant $\Theta(1)$

2) In Java, HashSet always ensures the size of the underlying hashmap is some power of 2. If this were not the case, the method above could potentially be a very poor hash function. Give a number M such that setting the hashmap's array to size M would break the method above.

note that choosing M = 31, strings with last characters matching would collide (adding into a hashmap you do % 31).

Performance Hashing (Taken from FA15 MT2)

Suppose a class has two hash functions *hashcode1()* and *hashcode2()* which both are good hash functions. For each of the hash functions below, state if it is a good regime. If not, provide a brief explanation why.

1) hashcode1() ^ hashcode2() (^ = XOR)

No. e.g. hashcode() and hashcode2() are the same, result will always be 0

2) hashcode1() if hashcode2() returns 0, else hashcode2()

Yes. both hashcodes are assumed to be good we will use hc1 most of the time

3) Generate a random number. If that number is even, then hashcode1(), else use hashcode2().

No, this is nondeterministic

4) -hashcode1() if hashcode1() is even else use hashcode1().

Yes values are distributed as evenly as for hashcode1() just differently