Midterm 2 Review Document Solution

CS 61B Spring 2017

Antares Chen + Kevin Lin

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 practice midterms from previous semesters both CS 61BL and CS 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.

Iterating Collections

Easy Mode

Warm-up Questions

1) How do you make an object iterable? What are the three methods for iterators? What is the interface that you need to implement?

   Implement `Iterable<T>` which requires a method `Iterator<T> iterator()`. `Iterator` contains method declarations for `hasNext()`, `next()`, and (optionally) `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?

   Changing the value of an instance variable is a way of changing state. This is bad because `hasNext()` should only return the status of the iterator, not change any component of it.

3) Why is `Collection` an interface?

   There are many different types of `Collections` like `List`, `Set`, `Map`, `SortedList`, etc. each with their own implementations

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.

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

    /** The number of elements in this SimpleCollection. */
    protected 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`.

```java
public class ArrayStack<E> extends SimpleCollection<E> {
    private E[] data;
    private int currIndex;

    public ArrayStack() {
        data = (E[]) new Object[2];
        currIndex = 0;
    }

    public boolean add(E elem) {
        if (currIndex >= data.length) {
            resize();
        }
        data[currIndex] = elem;
        currIndex += 1;
        size += 1;
        return true;
    }

    public boolean remove(E elem) {
        if (currIndex <= 0) {
            throw new NoSuchElementException();
        } else if (!elem.equals(data[currIndex - 1])) {
            throw new IllegalArgumentException();
        }
        currIndex -= 1;
        size -= 1;
        return true;
    }

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

Medium Mode

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

```java
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[currY].length == 0) {
           return false;
        }
        return currY < image.length && currX < image[currY].length;
    }

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

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

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 non-negative numbers.

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

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

    public Repeaterator(int[] array) {
        data = array;
        index = 0;
        repeats = 1;
        advance();
    }

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

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

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

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

Generics

Easy Mode

Warm-up Question Why do we use generics?

Because casting is a pain! By using generics, we no longer have to cast objects within other collections which results in better type safety.

Make This Generic Rewrite the `Node` class below to allow for generic types.

```java
public class Node {
    Object first;
    Node rest;

    public Node(Object first, Node rest) {
        this.first = first;
        this.rest = rest;
    }
}
```

```java
public class Node<T> {
    T first;
    Node<T> rest;

    public Node(T first, Node<T> rest) {
        this.first = first;
        this.rest = rest;
    }
}
```

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` to have `Number` as a super class.

2) What kind of types can you place in a `NumericSet` instance? Be specific in terms of the classes existing within the Java standard library.

   `Integer, Double Float, Long, Short, Byte, BigInteger, BigDecimal, ...`

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

   `NumericSet` already makes the generic type declaration `<T extends Number>`. Any usage of `T` after this refers to the same particular `T` that extends `Number`.

Generic Binary Search Trees 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?

   `int 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?

No.

Hard Mode

Fill 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` is a `List<? super T>`. x is of type `T`.

2) What is the type that `fill` returns?

   `void` return type. The list is mutated.

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

   `list` holds objects of type `T` or a super class of type `T`.

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

   It's possible that we want to fill a `List<Number>` with `Integers`. Declaring the `list` as `List<T>` would eliminate this possibility.

Binary Search The `Arrays` class also has a binary search method.

```
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>>` and `T`.

2) What is the return type for this method?

   `int`, the index of the element.

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

   `list` holds objects implementing `Comparable`. The `Comparable` type may be a super class of the list's elements. For example, in a `List<Integer>`, we want to be able to compare those `Integers` with other `Numbers` like `Floats`.

Asymptotics

Easy Mode

For the following code block, step through its execution and determine the tightest bound on its runtime.

```java
public static void easyMethod1(int[] array) {
    MultipleFunction mf = new MultipleFunction(4);
    for (int i = 0; i < array.length(); i += 1) {
        mf.setArg(array[i]);
        array[i] = mf.apply();
    }
}
```

```java
// Let N = arr.length and suppose mf.apply() takes Theta(N) time.
easyMethod1(arr);
```

Theta(N^2)

```java
public static void easyMethod2(int[] array, int low, int high) {
    if (low <= high) {
        if (array[low] == 0) {
            for (int i = low; i < high; i += 1) {
                System.out.println(array[i]);
            }
        }
        easyMethod2(array, low, low + (high - low) / 2);
        easyMethod2(array, low + (high - low) / 2, high);
    }
}
```

```java
// Let N = arr.length
easyMethod2(arr, 0, N);
```

O(N log N), Omega(N)

Medium Mode

For the following code block, step through its execution and determine the tightest bound on its runtime.

```
public static void mediumMethod1(int n) {
    for (int i = 1; i < n; i *= 2) {
        int j = 0;
        while (j < n) {
            j += 1;
        }
    }
}
```

```
// Let N be some number
mediumMethod1(N);
```

Theta(N log N)

```
public static void mediumMethod2(int[] arr) {
    for (int i = 0; i < arr.length; i += 1) {
        int j = i + 1;
        while (j < arr.length) {
            if (arr[i] == arr[j]) {
                return;
            }
            j += 1;
        }
    }
}
```

```
// Let N = arr.length
mediumMethod2(arr);
```

O(N^2), Omega(1)

Hard Mode

```java
public static int hardMethod1(int n) {
    return hardMethod1(n, n);
}

public static int hardMethod1(int x, int n) {
    if (x == 1) {
        return x;
    } else {
        int sum = 0;
        for (int i = 0; i < n; i += 1) {
            sum += hardMethod1(x - 1, n);
        }
        return sum;
    }
}
```

```java
// Let N be some number
hardMethod1(N);
```

Theta(N^{N-1})

```java
public static void bogosort(int[] arr) {
    if (!isSorted(arr)) {
        shuffle(arr);
        bogosort(arr);
    }
}
```

```java
// Let N = arr.length
// Suppose isSorted runs in O(N) time and shuffle also runs in O(N) time
// Assume each shuffle returned is unique!
int[] arr = {n, n - 1, n - 2, ..., 2, 1};
bogosort(arr);
```

O(NN!), Omega(N): There are O(N!) unique permutations of N elements and each call takes O(N) time.

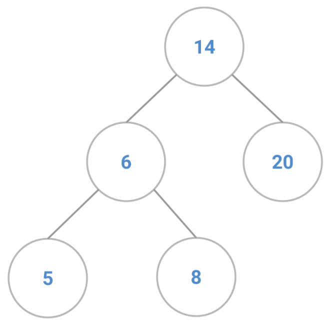
Tree Structures

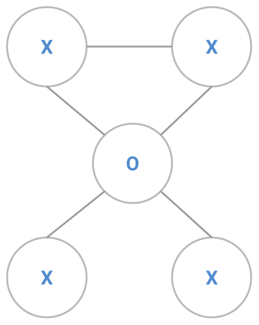
Easy Mode

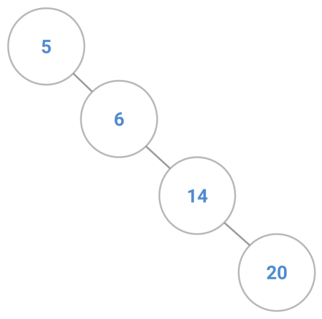
Warm-up Question How do you define a tree?

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.

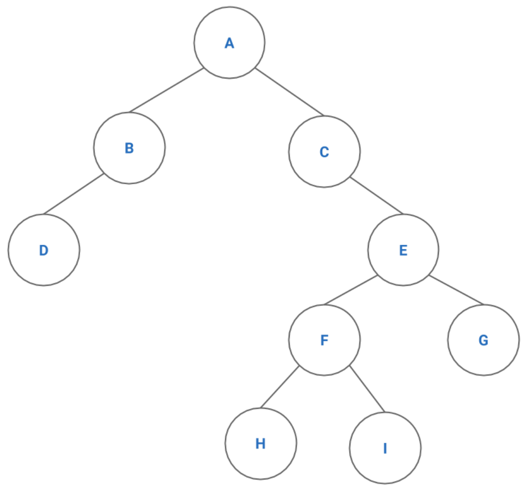






Yes, No, Yes

Order Order Order



In-order D B A C H F I E G

Pre-order A B D C E F H I G

Post-order D B H I F G E C A

Medium Mode

Good Ole Amoeba For the next questions, consider the `AmoebaFamily` class definition below.

```
public class AmoebaFamily {
    public Amoeba root;

    public static class Amoeba {
        public String name;
        public Amoeba parent;
        public List<Amoeba> children;

        public Amoeba(String name, Amoeba parent) {
            this.name = name;
            this.parent = parent;
            this.children = new ArrayList<Amoeba>();
        }
    }
}
```

Amoeba Search Define `AmoebaFamily::findMoeba`, a method that will search for an `Amoeba` with the given `name`. Assume that the `root`, `name`, and every child of an `Amoeba` are not null.

```
/** Returns true if this AmoebaFamily has an Amoeba with NAME as name. */
public boolean findMoeba(String name) {
    return findMoebaHelper(this.root, name);
}

private boolean findMoebaHelper(Ameoba node, String name) {
    if (name.equals(node.name)) {
        return true;
    }
    for (Amoeba child : node.children) {
        if (findMoebaHelper(child, name)) {
            return true;
        }
    }
    return false;
}
```

Amoeba Sum Let's say each Amoeba object contains an additional int instance variable, value. Write a method that will determine the sum of all values in an AmoebaFamily. Assume that the root, name, and every child of an Amoeba are not null.

```
/** Returns the sum of all Amoeba values in this AmoebaFamily. */
public int sumoeba() {
    return sumoebaHelper(this.root);
}

private int sumoebaHelper(Amoeba node) {
    int value = node.value;
    for (Amoeba child : node.children) {
        value += sumoebaHelper(child);
    }
    return value;
}
```

Hard Mode

Amoeba Path Implement pathMoeba which returns the length of the shortest path between two Amoeba in an AmoebaFamily. Amoeba are identified by their name. The AmoebaFamily is guaranteed to contain both Amoeba.

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

Binary Search Trees

Easy Mode

Define That BST Though

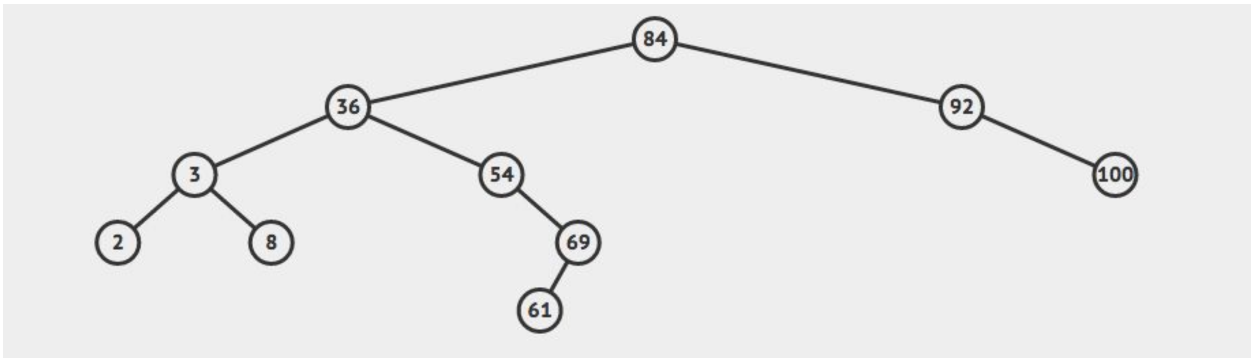
1) List the two properties that define a binary search tree.

   Each node has at most two children, with the left child having a value less than the node and the right child having a value greater than the node.

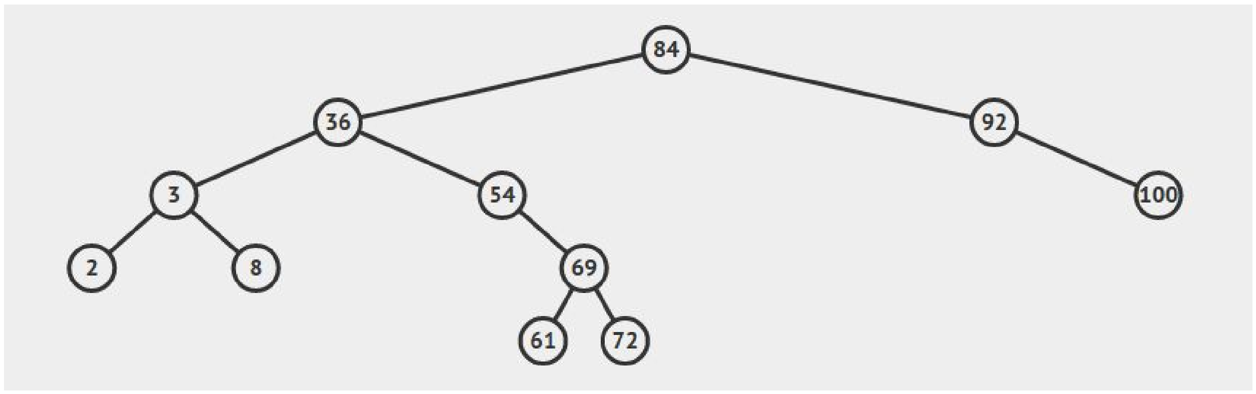
2) For the purpose of this class, do we care about inserting two elements into a BST of the same value?

   No.

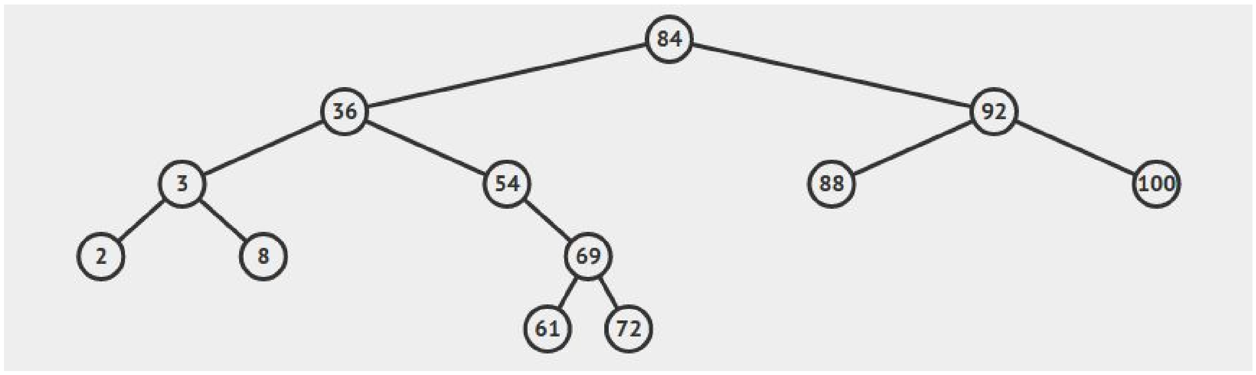
Do BST Things Given the following BST, perform the following operations.



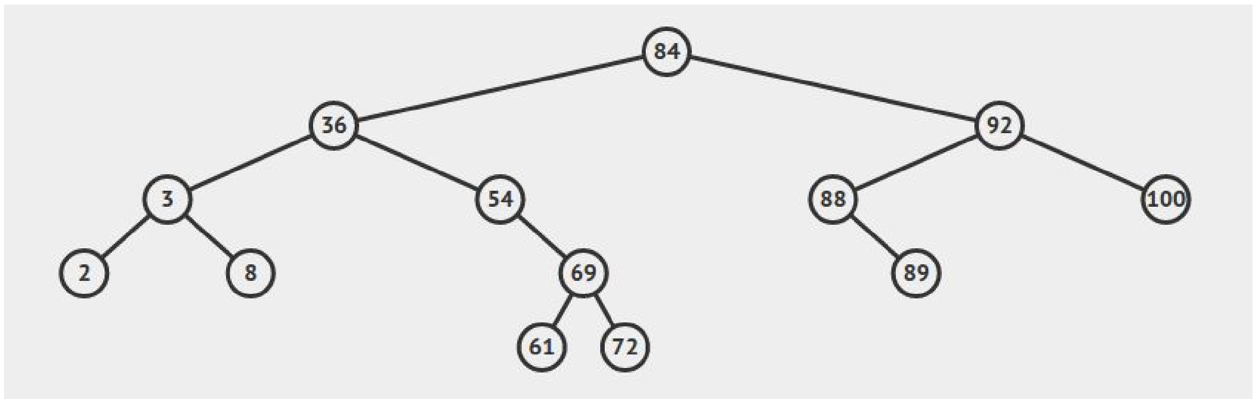
1) Insert 72



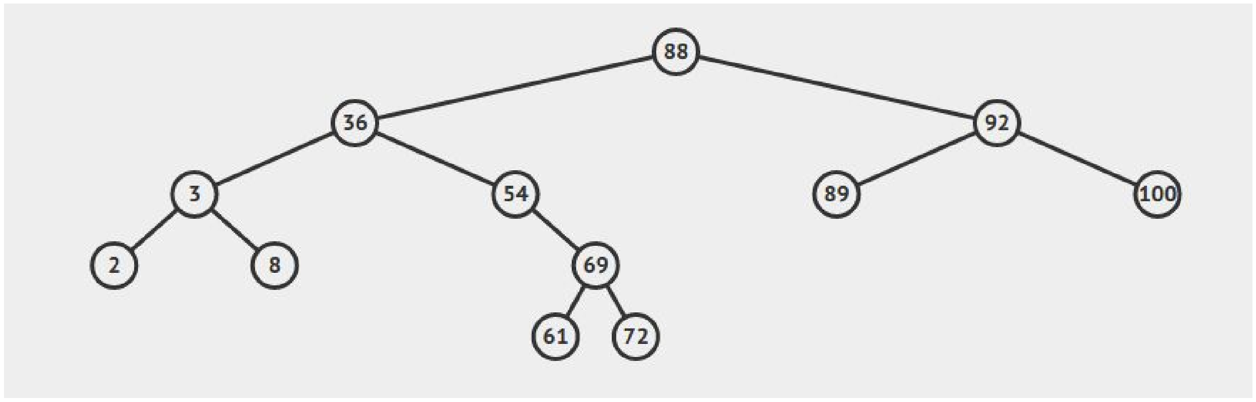
2) Insert 88



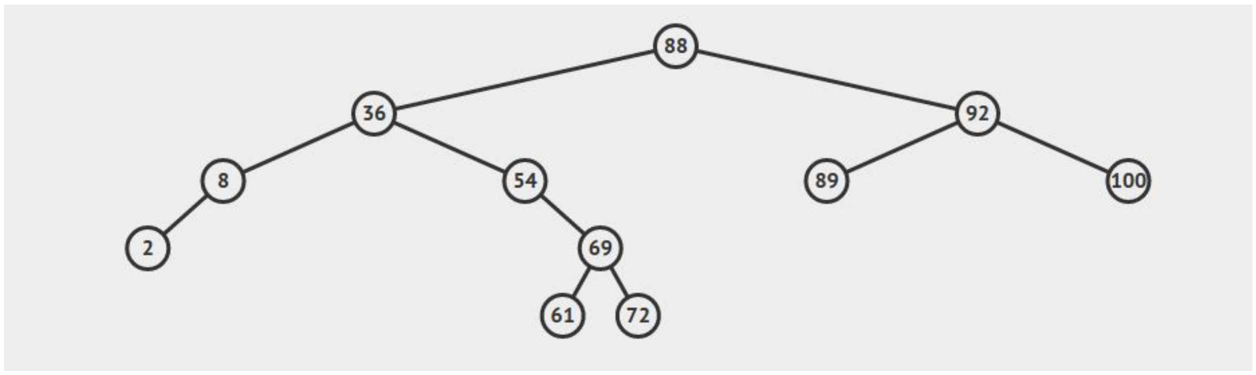
3) Insert 89



4) Remove 84 (promote right subtree)

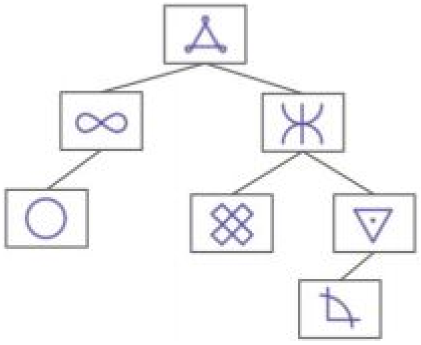


5) Remove 3

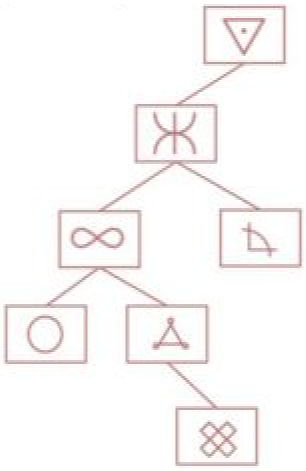


Medium Mode

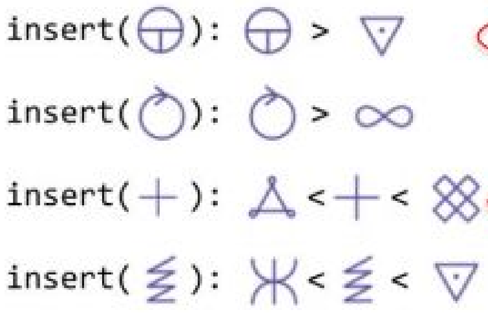
Symbol Tree Consider the binary search tree below. Each symbol has an underlying meaning. For example the root node may represent “snowman” while its immediate left child may represent “overthrow the capitalist regime”.



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 **printed example tree above** by drawing the object (and any needed links) onto the tree. Assume the objects are inserted in the order shown below. 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”.



[✓ Drawn In Tree Above] Not Enough Information

Drawn In Tree Above [✓ Not Enough Information]

[✓ Drawn In Tree Above] Not Enough Information

Drawn In Tree Above [✓ Not Enough Information]

Hard Mode

Linked Lists Are Back Convert a given Binary Search Tree into a sorted linked list.

```java
/** 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 we define a Red-Black Tree?

   The root node is black. Every red node has at most two black children. Every path from a node its descendent leaf has the same number of black nodes.

2) How do we 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 number 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?

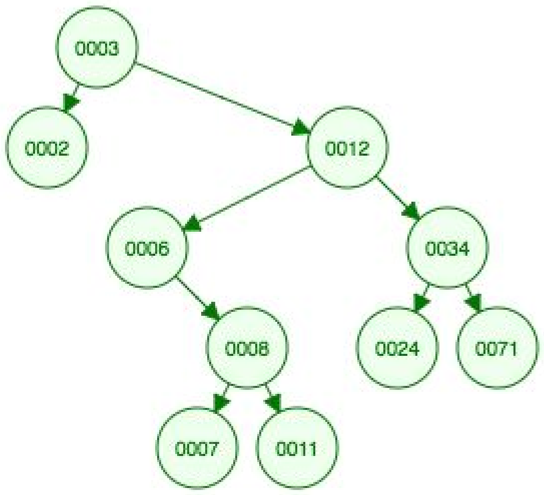
   2-3 Trees do not have 4 nodes.

Fill In the Table
Give a tight asymptotic runtime bound for each cell in the table below.

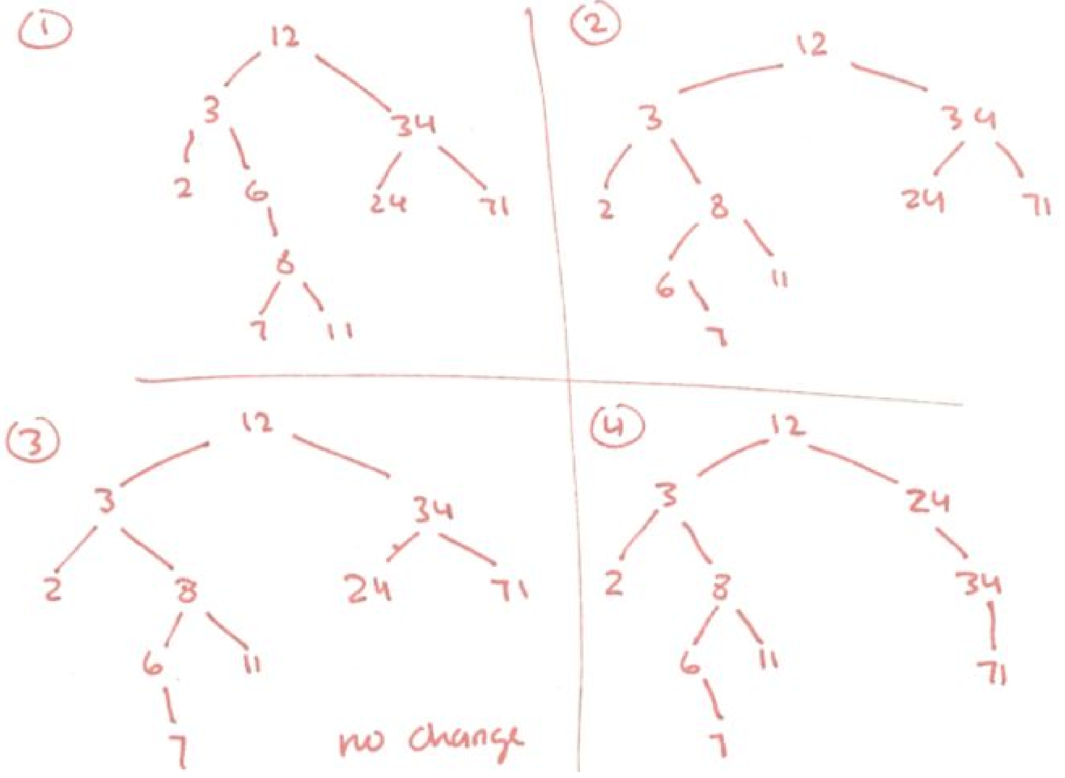
	Binary Search Tree		Red-Black Tree	
	Best	Worst	Best	Worst
Find	Theta(1)	Theta(N)	Theta(1)	Theta(log N)
Insert	Theta(1)	Theta(N)	Theta(log N)	Theta(log N)
Delete	Theta(1)	Theta(N)	Theta(1)	Theta(log N)

Medium Mode

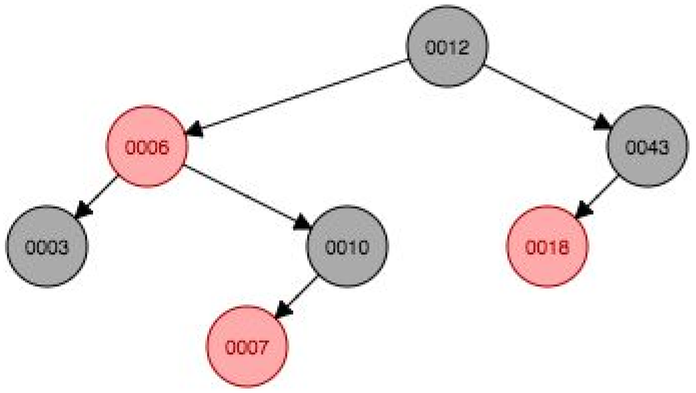
Rotations For Days Perform the following operations in order.



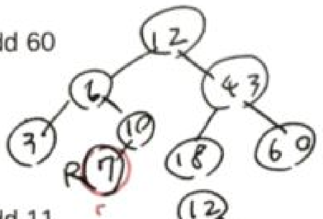
1) Rotate 3 left
2) Rotate 6 left
3) Rotate 2 right
4) Rotate 34 right



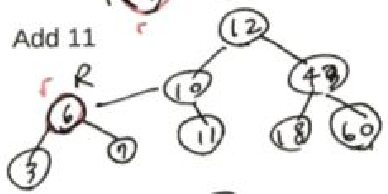
Add it To Me Perform the listed operation for the following Left Leaning Red-Black Tree.



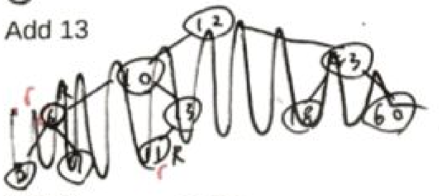
1) Add 60

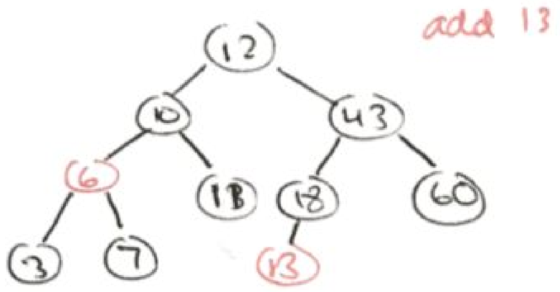


2) Add 11

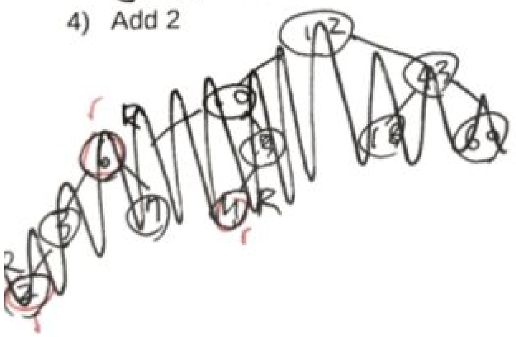


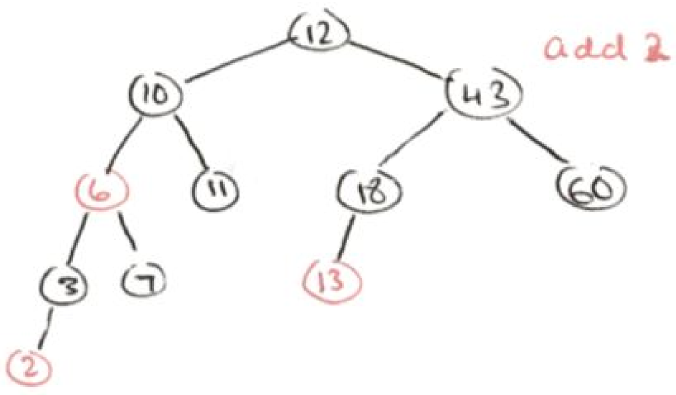
3) Add 13



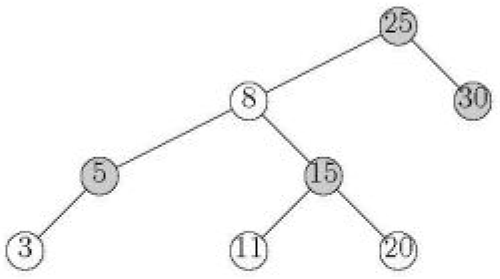


4) Add 2

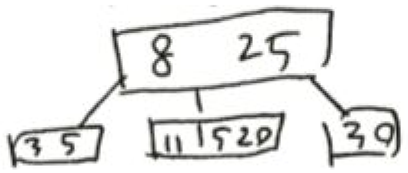




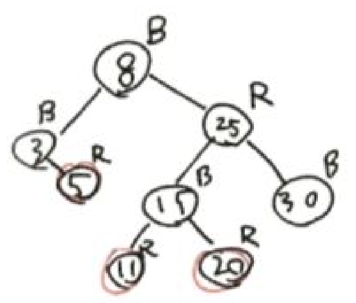
Conversion Times For the following red-black tree, perform the following operations (shaded nodes are black)



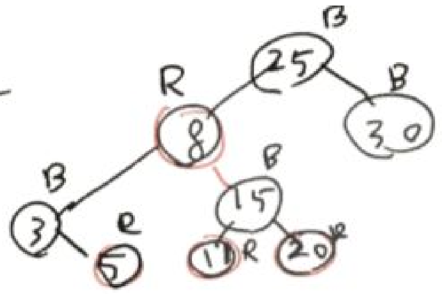
1) Draw the corresponding 2-3-4 Tree



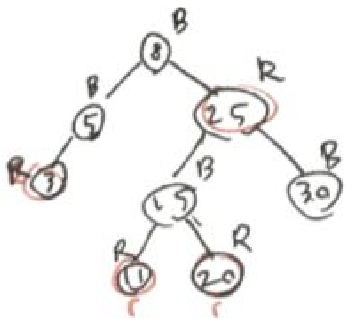
2) Draw a different Red-Black Tree that corresponds to that 2-3-4 Tree



or



or



Hard Mode

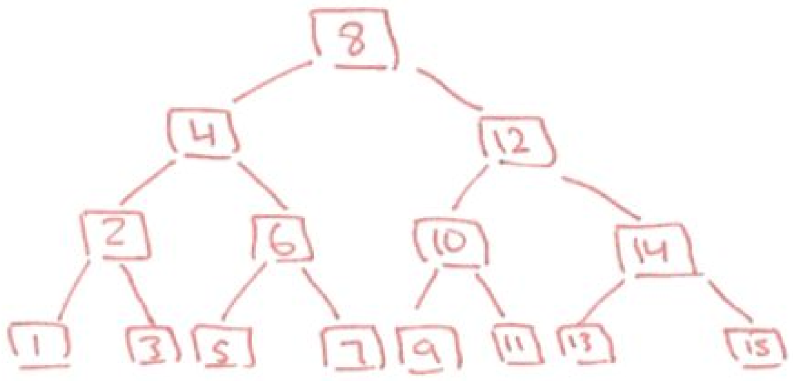
Grab Bag Questions

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.

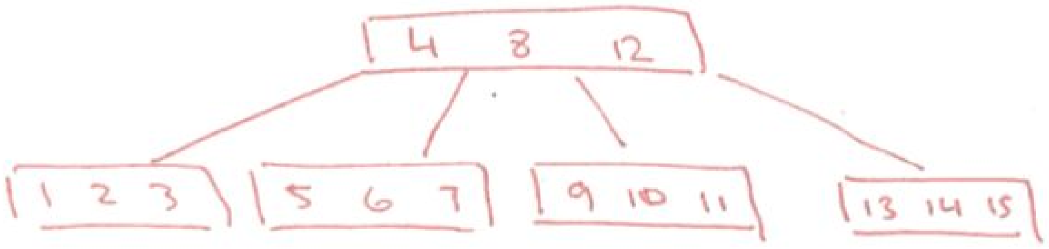
   Max = 2h + 1, Min = h

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

1) What three tenants must a good hashcode follow?

   Deterministic, good distribution, and the `equals()` contract: two objects equal to each other must have the same `hashCode()`.

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

   Return the memory address.

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

   Both `equals()` and `hashCode()`.

4) Suppose the hash code for `String` simply returns a numeric representation of the first letter. Why is this a bad hash code? Give a better hashing regime.

   Because any dataset whose words begin with the same letter will all hash to the same bucket.

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) with a good hash function. O(N^2) with a poor has function as the hash table will need to iterate down the entries to the end of the chain before inserting.

StringSet The `StringSet` class defines a set for strings. The set is backed by a hash table that resolves collisions by chaining. Implement `put` and `resize`.

```java
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);
    }
    /** 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 = key % entries.length;
        Entry curr = entries[index];
        if (curr == null) {
            entries[index] = new Entry(key, value, null);
        } else {
            while (curr.next != null) {
                if (key.equals(curr.key) && value.equals(curr.value)) {
                    return false;
                } else if (key.equals(curr.key)) {
                    curr.value = value;
                    return true;
                }
                curr = curr.next;
            }
            curr.next = new Entry(key, value, null);
            size += 1;
            load = size / entries.length;
            if (load >= MAX_LOAD_FACTOR) {
                resize();
            }
        }
        return true;
    }

    /** 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 = 0;
        for (int i = 0; i < old.length; i += 1) {
            for (Entry curr = old[i]; curr != null; curr = curr.next) {
                put(curr.key, curr.value);
            }
        }
    }
}
```

Medium Mode

Hashing Binary Trees Consider an implementation of `BSTMap` in lab with one instance variable, `root`, which was of type `TreeNode`. A `TreeNode` instance has four instance variables: `T value`, `TreeNode left`, `TreeNode right`, and `int size`. Override Java's `hashCode()` function to hash binary trees.

```
public int hashCode() {
    return helper(root);
}

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

Hashing Strings in Java 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 a `String` into the `HashSet`.

   Theta(N + L) in the worst case, Theta(L) in the best case. If we treat L as a constant, then Theta(1).

2) In Java, `HashSet` always ensures the size of the underlying array 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 `HashSet's` array to size *M* would break the method above.

   Choose M = 31.

Performance Hashing 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()`.

   No. Suppose `hashCode1() == hashCode2()`. Then the result is always 0.

2) `hashCode1()` if `hashCode2()` returns 0, else `hashCode2()`.

   Yes. Both hash codes are assumed to be good and we'll use at most one at a time.

3) Generate a random number. If that number is even, then `hashCode1()`, else use `hashCode2()`.

   No, this is non-deterministic.

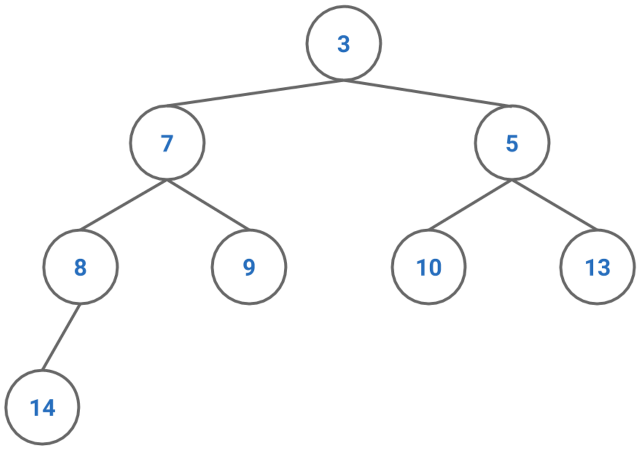
4) `-hashCode1()` if `hashCode1()` is even else use `hashCode1()`.

   Yes, values are distributed as evenly as `hashCode1()`, just differently.

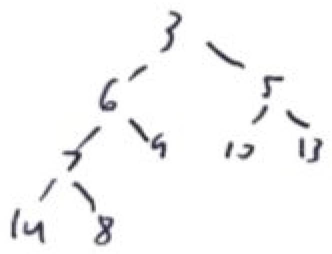
Heaps and Priority Queues

Easy Mode

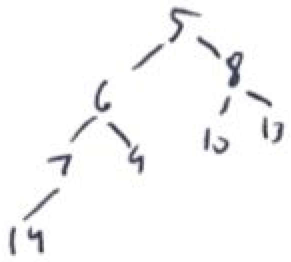
Insert Into Heaps The heap below holds integers with each integer's initial priority set to its value. Perform the following operations in order on the heap.



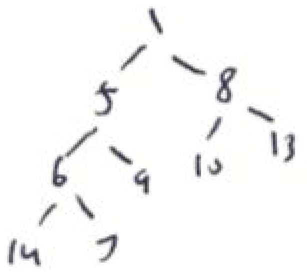
1) Insert 6



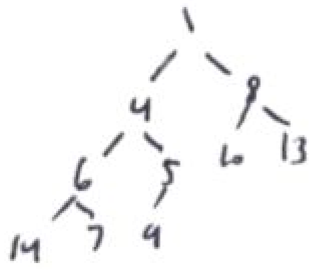
2) Remove-min



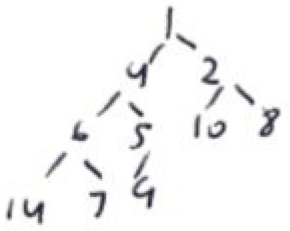
3) Insert 1



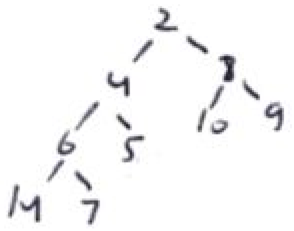
4) Insert 4



5) Change-priority 13 to 2



6) Remove min



What is the array representation of the final heap?

[2, 4, 8, 6, 5, 10, 9, 14, 7]

Runtime Funtime Fill in the table with the correct runtime bounds for a Min Binary Heap.

Operation	Get-min	Remove-min	Insert	Change-priority
Best case	Theta(1)	Theta(1)	Theta(1)	Theta(1)
Worst case	Theta(1)	Theta(log N)	Theta(log N)	Theta(log N)

Medium Mode

Evil Alan and Evil Sarah are Up To No Good
You're walking down the street one day and all of a sudden, Evil Alan jumps from a bush and challenges you to quickly implement an integer max-heap.

You, being the clever CS 61B student you are, say to yourself, "Ah ha! I'll just use my min-heap implementation as a template to write `MaxHeap.java`." But before you can begin coding, Evil Sarah deletes your min-heap implementation.

However, you notice that you still have the `MinHeap.class` file; could you use it to complete the challenge? You can still use methods from min-heap but you cannot modify them. If so, describe your approach. If not, explain why it is impossible.

Write a wrapper class that uses the min-heap by inserting the negative of each element. When popping off the heap, return the negation of the result which restores the original value.

Yet Another Runtime Question What is the best and worst case runtime for this block of code.

```
public static void foobar(PriorityQueue<Integer> heap, int[] in) {
    int n = in.length;
    for (x : in) {
        heap.add(x)
    }
}
```

Theta(N) in the best case, Theta(N log N) in the worst case.

Hard Mode

More Heap Questions Answer the following questions about heaps.

1) What are the minimum and maximum number of elements in a heap of height h?

   Assuming a leaf node is of height 1, the min is 2^{h-1} when there's only one element in the last row while the max is $2^h - 1$ when the last row is completely full.

2) Is it true that for any subtree of a max-heap, the root of the subtree contains the largest value occurring anywhere in that subtree?

   Yes because that follows from the definition of a max-heap.

3) Is an array in sorted ascending order a min-heap?

   Yes. Consider any value v at index i. Since the array is sorted ascending, the value at 2i and 2i + 1 must be greater than v which maintains the min-heap property for any i.

4) Where in a max-heap might the smallest element reside, assuming all elements are distinct?

   In a max-heap, the min element must be a leaf.

5) Given an array representation of a binary heap, where do all the leaf nodes of the heap reside?

   The leaf nodes in the heap reside in the second half of the array since heaps are complete binary trees.

Merging Sorted Lists Give an O(N log K) algorithm to merge K sorted lists into one sorted list, where N is the total number of elements in all input lists.

Create a min-heap of size K. For each of the K sorted lists, add the minimum element at the front of the list into the heap remembering from which list each element came from. Remove the min from the heap and insert it into the result, and replenish the heap with the next element from the sorted list. Each heap insertion and bubbling takes O(log K) time and we perform this operation N times, once for each element, yielding an overall runtime in O(N log K).

K-ary Heaps A K-ary heap is like a binary heap, but non-leaf nodes have K children instead of 2 children. Answer the following questions.

1) How would you represent a k-ary heap in an array?

   Use the same indexing scheme as in a binary heap, but let the children of the value at index n be given by the formula Kn + i.

2) What is the height of a k-ary heap of N elements in terms of N and K?

   height = ceil($\log_K$ N)

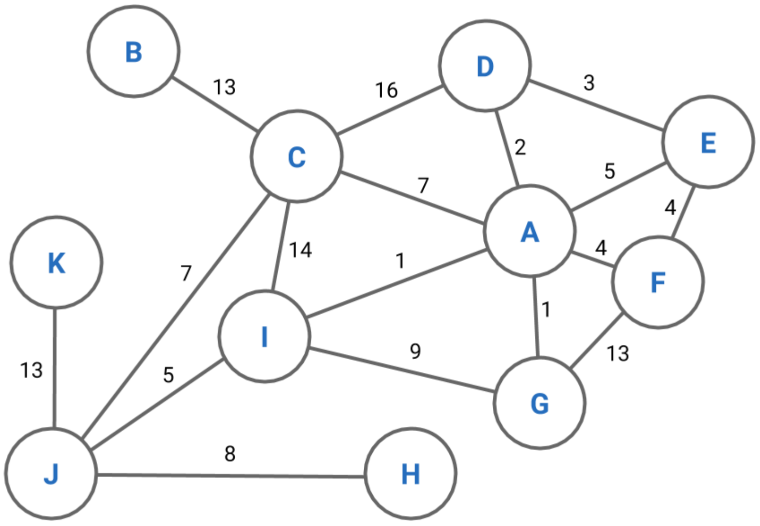
3) Give an efficient algorithm to insert an element into the heap and provide a runtime bound in terms of K and N.

   With a bubble-up strategy, the runtime is in O($\log_K$ N) time since each node has only up to 1 parent. With a bubble-down strategy, we need to consider up to K possible children which results in O(K $\log_K$ N) time since we spend Theta(K) time scanning across children and need to examine up to O($\log_K$ N) levels.

Graphs

Easy Mode

Traversal Time Write out the listed traversals starting at node A. Break ties alphabetically.



1) Breadth First Search

   BFS: A C D E F G I B J H K

2) Depth First Search

   DFS: A C B D E F G I J H K

3) Dijkstra's

   Dijkstra's: A G I D F E J C H K B

Runtime Funtime (More Fun Times) For each algorithm, list the worst case runtime.

BFS	DFS	Dijkstra's	A*	Topological Sort
Theta(V + E)	Theta(V + E)	Theta((V + E) log V)	Theta((V + E) log V)	Theta(V + E)

And for each representation fill in the runtimes.

	Storage size	Add vertex	Add edge	Remove vertex	Remove edge	Query if edge exists
Adjacency List	Theta(V + E)	Theta(1)	Theta(1)	Theta(V + E)	Theta(E)	Theta(E)
Adjacency Matrix	Theta(V^2)	Theta(V^2)	Theta(1)	Theta(V^2)	Theta(1)	Theta(1)

Last Question When would you use an adjacency matrix over an adjacency list?

When the graph is very dense, or where the number of edges, |E|, approaches $|V^2|$.

Medium Mode

Graph Questions Answer if each problem is always true, sometimes true, or never true.

1) Given a graph with V vertices and V-1 edges, adding an edge would introduce a cycle

   Always true by the definition of a tree.

2) A DFS traversal starting at vertex u ending at v always finds the shortest path in an unweighted graph.

   Sometimes true. If there is only a single path, then DFS will find that path (which happens to also be the shortest path). If there are multiple paths, then the returned path may not necessarily be the shortest.

3) Dijkstra's finds the shortest path for a graph with negative edge weights.

   Sometimes true. Try to construct several examples with three or four nodes with one negative edge weight.

4) A DFS on a directed acyclic graph produces a topological sort.

   Sometimes true. It works on a linear graph but not in general.

Dijkstra's Algorithm The runtime for Dijkstra's algorithm is O((V + E) log (V)); however this is specific only to binary heaps. Let's provide a more general runtime bound. Suppose we have a priority queue backed by some arbitrary heap implementation. Given that this unknown heap contains N elements, suppose the runtime of remove-min is f(N) and the runtime of change-priority is g(N).

1) What is the runtime of Dijkstra's in terms of f(V) and g(V)?

   In the worst case, we check every edge and decrease or increase the key of a node in the heap. In addition, we also always need to remove all vertices from the heap. The overall runtime is in O(Eg(V) + Vf(V))

2) Turns out the optimal version of Dijkstra's algorithm uses something called a fibonacci heap. The fibonacci heap has amortized constant time change-priority, and log time remove-min. What is the runtime of Dijkstra's algorithm?

   O(E + V log V)

Hard Mode

Algorithm Design Provide an algorithm matching the given time bound for each problem.

1) Suppose you are terribly lost in downtown SF (basically a maze), but happen to have an infinite amount of pennies (you made a killing investing in Kelp, a hot new Yelp-for-seafood startup that recently IPO'd). Discuss how you would get out of SF by solving the following problem.

   Let G be an undirected connected graph. Give an O(V + E) time algorithm to compute a path in G that traverses each edge in E exactly once in each direction.

   Modify depth first search. First, mark each edge with directions taken. If both directions exist, then remove the edge. To ensure all unexplored edges are explored, check unexplored edges before exploring singly traversed edges. Finally, when the algorithm gets stuck, backtrack to a predecessor node and try again.

2) A directed graph G is singly connected if for all vertices u, v, u connected to v implies that there is at most one simple path from u to v. Provide an efficient algorithm to determine whether or not a directed graph is singly connected.

```
for all unvisited vertices:
    visit vertex:
        run DFS from the vertex
        if vertex is visited twice:
            return False
return True
```

Regular Expressions

Easy Mode

1) Write the regex that matches any string that has an odd number of a's and ends with b.

   ```
   ^[^a]*a([^a]*a[^a]*a)*[^a]*b$
   ```

2) Write a regex that matches any binary string of length exactly 8, 16, 32, or 64.

   ```
   ^(0|1){64}|(0|1){32}|(0|1){16}|(0|1){8}$
   ```

3) Write the regex that matches any string with the word "banana" in it.

   ```
   ^.*banana.*$
   ```

Medium Mode

Bracketed List A certain type of bracketed list consists of words composed of one or more lower-case letters alternating with unsigned decimal numerals and separated with commas.

```
[]
[wolf, 12, cat, 41]
[dog, 9001, cat]
```

As illustrated, commas may be followed (but not preceded) by blanks, and the list may have odd length (ending in a word with no following numeral.) There are no spaces around the '[]' braces. Write the **Java pattern** that matches an alternating list.

```
\[(([a-z]+, *\d+, *)*[a-z]+(, *[0-9]+)?)?\]
```

IT'S... IT'S...
imgflip.com
IT'S LIT
ifunny.co