Midterm 2 Review Document

CS 61B Spring 2018

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?

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?

3) Why is `Collection` an interface?

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. */
    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`.

```
public class ArrayStack<E> extends SimpleCollection<E> {
    // some code may go here...









}
```

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.

```
public class ImageIterator implements Iterator<Color> {
    Color[][] image;
    int currX;
    int currY;

    public ImageIterator(Color[][] image) {
        ____________________________________;
        currX = ____________________________;
        currY = ____________________________;
    }

    public void hasNext() {
        if (__________________ || __________________) {
           return false;
        }
        return __________________ && __________________;
    }

    public Color next() {
        ____________________________________;
        currY += (_____________) / _____________;
        currX = (_____________) % _____________;
        ____________________________________;
    }

    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) {
        ____________________________________;
        ____________________________________;
        ____________________________________;
        advance();
    }

    private void advance() {
        repeats -= 1;
        while (_____________________________) {
            ____________________________________;
            ____________________________________;
        }
    }

    public boolean hasNext() {
        ____________________________________;
    }

    public int next() {
        ____________________________________;
        advance();
        ____________________________________;
    }

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

Generics

Easy Mode

Warm-up Question Why do we use generics?

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

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

    public Node(Object first, Node 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?

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.

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

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

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

Hard Mode

Fill Suppose we've defined a `fill` method with 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?

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

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

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

Binary Search Suppose we've also defined 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?

2) What is the return type for this method?

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

Asymptotics

Easy Mode

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

```
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();
    }
}
```

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

```
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);
    }
}
```

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

```
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);
```

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);
```

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

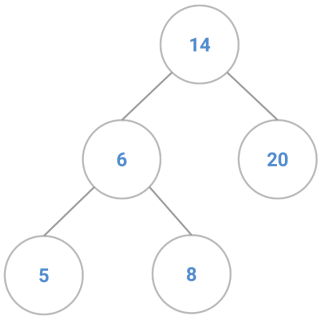
```java
// Let N = arr.length
// Suppose isSorted runs in O(N) time and shuffle also runs in O(N) time
// Assume each shuffle returned is unique!
int[] arr = {n, n - 1, n - 2, ..., 2, 1};
bogosort(arr);
```

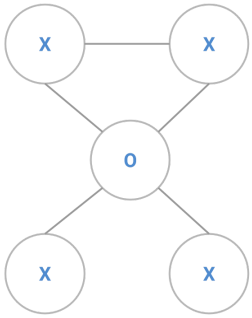
Tree Structures

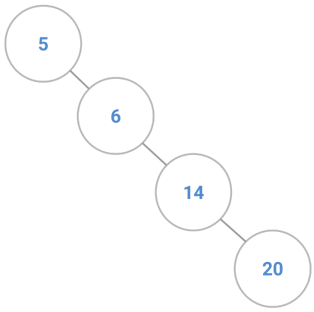
Easy Mode

Warm-up Question How do you define a tree?

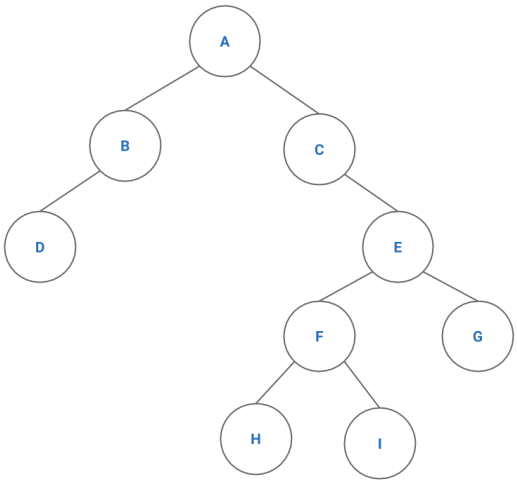
Valid Trees For each of the following images state if they are valid tree structures.







Order Order Order



In-order

Pre-order

Post-order

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) {



}
```

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() {



}
```

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`.

```
// go at it friends
```

Binary Search Trees

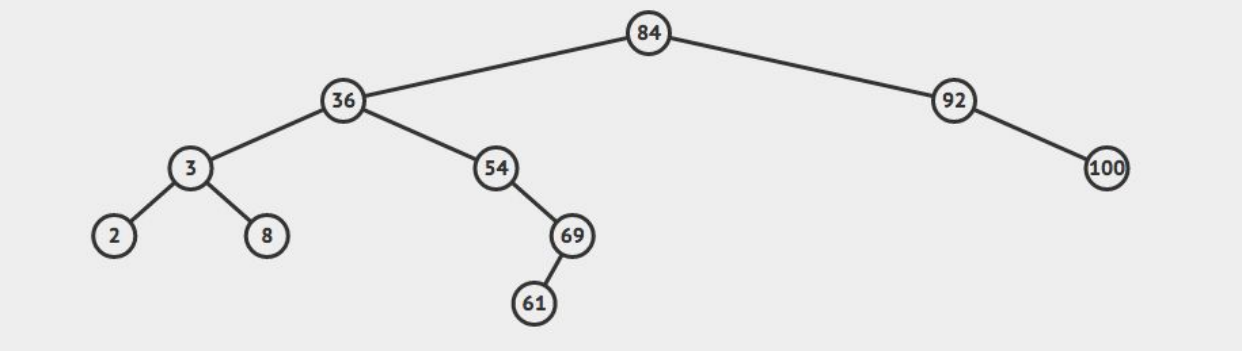
Easy Mode

Define That BST Though

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

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

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



1) Insert 72

2) Insert 88

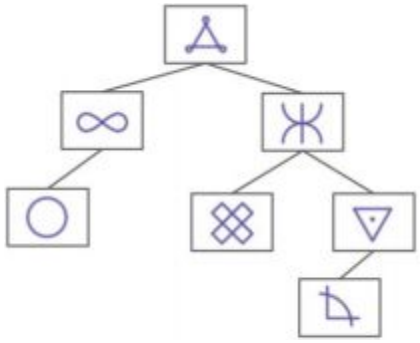
3) Insert 89

4) Remove 84 (promote right subtree)

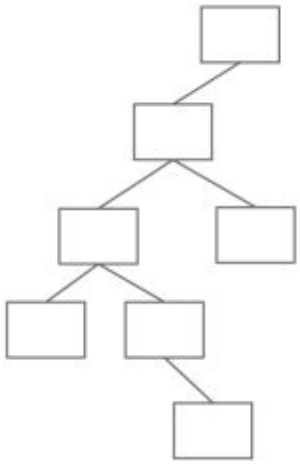
5) Remove 3

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”.

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

Hard Mode

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

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

/** A helper method. */
private Node helper(TreeNode node) {
    if (____________________) {
        ____________________________________________________;
    }
    if (____________________) {
        TreeNode left = helper(node.left);
        while (_____________________________) {
            ____________________________________________;
        }
        ____________________________________________________;
        ____________________________________________________;
    }
    if (____________________) {
        TreeNode right = helper(node.right);
        while (_____________________________) {
            ____________________________________________;
        }
        ____________________________________________________;
        ____________________________________________________;
    }
    return node;
}
```

Balanced Search Trees

Easy Mode

Warm-up Questions

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

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

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

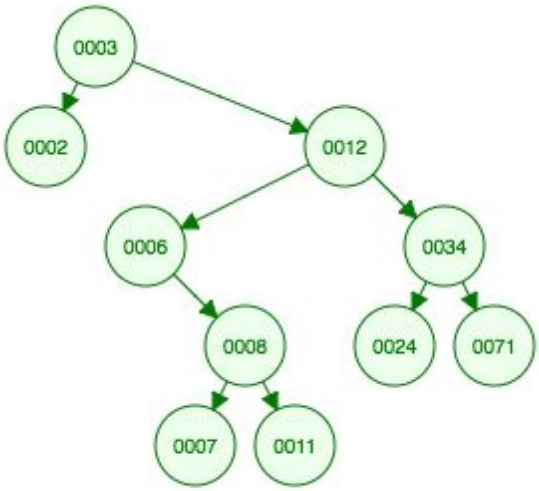
4) What is the difference between 2-3-4 Trees and 2-3 Trees?

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				
Insert				
Delete				

Medium Mode

Rotations For Days Perform the following operations in order.



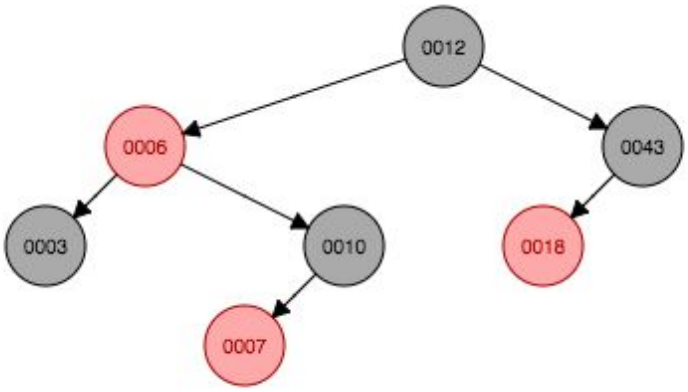
1) Rotate 3 left

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.



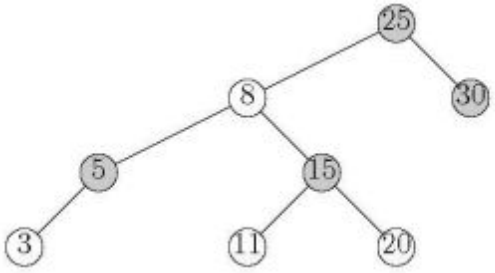
1) Add 60

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

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.

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

Hashbrowns

Easy Mode

Get Ready For Hashbrowns

1) What three tenets must a good hashcode follow?

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

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

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.

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?

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`.

```
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) {









    }

    /** If the load factor of the hash table is greater than
     *  MAX_LOAD_FACTOR then this method will double the size
     *  of the map. */
    private void resize() {









    }
}
```

Medium Mode

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`.

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.

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

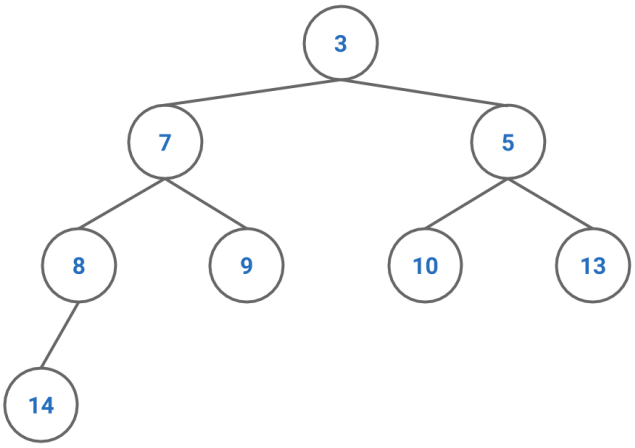
2) Generate a random number. If that number is even, then `hashCode1()`, else use `hashCode2()`.

3) -hashCode1() if hashCode1() is even else use hashCode1().

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?

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				
Worst case				

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.

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)
    }
}
```

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?

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?

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

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

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

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.

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?

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

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


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