CS 61B Spring 2017 Guerrilla Section 4 Worksheet

11 March 2017

Directions: In groups of 4-5, work on the following exercises. Do not proceed to the next exercise until everyone in your group has the answer and *understands why the answer is what it is*. Of course, a topic appearing on this worksheet does not imply that the topic will appear on the midterm, nor does a topic not appearing on this worksheet imply that the topic will not appear on the midterm.

1 Asymptotic Approach

Give a tight asymptotic runtime bound on `foo(n)`.

```
1  int foo(int n) {
2      if (n == 0) {
3          return 0;
4      }
5      baz(n);
6      return foo(n/3) + foo(n/3) + foo(n/3);
7  }
8
9  int baz(int n) {
10     for (int i = 0; i < n; i += 1) {
11         System.out.println("Help me! I'm trapped in a loop");
12     }
13     return n;
14 }
```

Level	Number of Nodes	Work per Node	Total Amount of Work

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

2 Asymptotic Potpourri

For each of the following code snippets, give a tight asymptotic runtime bound with respect to n.

(a)
```
public void mystery1(int n) {
    for (int i = 0; i < n; i += 1) {
        for (int j = 0; j < n; j += 1) {
            i = i * 2;
            j = j * 2;
        }
    }
}
```

(b)
```
public void mystery2(int n) {
    for (int i = n; i > 0; i = i / 2) {
        for (int j = 0; j < i * 2; j += 1) {
            System.out.println("Hello World");
        }
    }
}
```

(c)
```
public void mystery3(int n) {
    for (int i = n; i > 0; i = i / 2) {
        for (int j = 0; j < i * i; j += 1) {
            System.out.println("Hello World");
        }
    }
}
```

(d)
```
public void mystery4(int n) {
   int i = 1, s = 1;
   while (s <= n) {
        i++;
        s = s + i;
        System.out.println(s);
    }
}
```

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

3 Stacks of Fun

An SQueue is a queue implemented using two stacks. Below is the code for a definition of this class.

```
1  public class SQueue {
2      private Stack inStack;
3      private Stack outStack;
4      public SQueue() {
5          inStack = new Stack();
6          outStack = new Stack();
7      }
8
9      public void enqueue(int item) {
10         inStack.push(item);
11     }
12
13     public void dequeue() {
14         if (outStack.isEmpty()) {
15         while (!inStack.isEmpty()) {
16             outStack.push(inStack.pop());
17         }
18     }
19     return outStack.pop();
20     }
21 }
```

Now, suppose we construct an SQueue and enqueue 100 items:

(a) How many calls to push and pop result from the next call to dequeue?

(b) How many calls to push and pop result from each of the next 99 calls to dequeue?

(c) How many calls to push and pop (total) were required to dequeue 100 elements? How many operations is this per element dequeued?

(d) What is the worst-case time to dequeue an item from an SQueue containing N elements? What is the runtime in the best case? Answer using $\Theta(\cdot)$ notation. You may assume that both push and pop run in $\Theta(1)$.

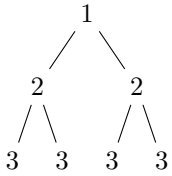
(e) What is the amortized time to dequeue an item from an SQueue containing N elements? Again, answer using $\Theta(\cdot)$ notation.

STOP!

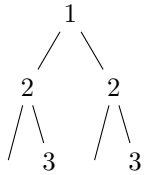
Don't proceed until everyone in your group has finished and understands all exercises in this section!

4 Symmetric Tree

Given a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center). For example, this binary tree is symmetric:



But this tree is not:



You may define additional methods to help solve the problem.

```
1  public class BinaryTree<T> {
2      protected Node root;
3      protected class Node {
4          public T value;
5          public Node left;
6          public Node right;
7      }
8
9      public boolean isSymmetric() {
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34     }
35 }
```

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

5 Greetings, Tree Traveler

(a) Draw a full binary tree that has the following pre-order and post-order. Each node should contain exactly one letter. A full binary tree is a tree such that all nodes except leaf nodes have exactly 2 children.

(i) Pre-order: C T U W X S A Z O

(ii) Post-order: W X U S T Z O A C

(b) What is the in-order traversal of this tree?

(c) Can a tree have the same in-order and post-order traversals? If so, what can you say about the tree?

(d) What about a tree with the same pre-order and post-order traversals?

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

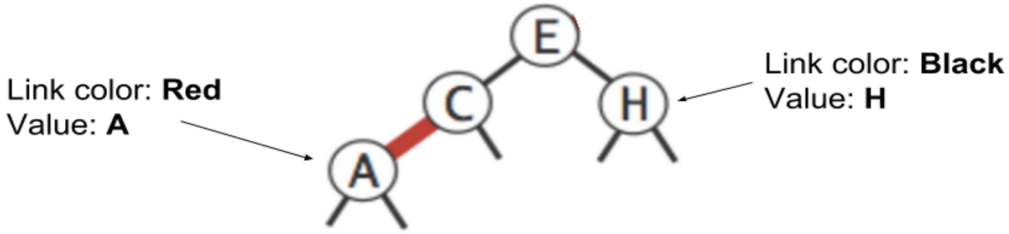
6 Balanced Search Tree Mechanics

(a) Insert the following numbers in order into a 2-3 tree: 20, 10, 35, 40, 50, 5, 25, 15, 30, 60

(b) Draw a red-black tree that corresponds to the 2-3 tree you drew in part a.

7 What color am I?

For each of the situations below in a valid LLRB tree, indicate whether the node's link to its parent is red, black, or either.



Questions:

(1) _____ The largest value in a tree with more than one node.

(2) _____ The smallest value in a tree with more than one node.

(3) _____ A node whose parent is red (parent's link color is red).

(4) _____ A node whose children are the same color.

(5) _____ A freshly inserted node after the insertion operation is completed.

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

8 Weight Times

A Quick Union data structure is used to handle set union and membership operations. The supported methods are:

(i) `connect(a, b)` - connects the set of a to the set of b

(ii) `isConnected(a, b)` - returns true if a and b are in the same set

Example:

```
connect(a, b)
connect(b, c)
connect(a, d)
isConnected(c, d)
isConnected(d, b)
```

Internally, a Quick Union's sets are represented using trees. Sets can be connected by adding one set's tree to the root of another set's tree. Note that Weighted Quick Union data structures are similar to Quick Union data structures, except that a Weighted Quick Union will always add the shorter tree to the root of the taller tree during connect operations.

(a) Draw the Weighted Quick Union object that results after the following four method calls:

```
connect(1, 3)
connect(0, 4)
connect(0, 1)
connect(0, 2)
```

(b) What is the resulting array of the Weighted Quick Union after the calls in (a) are executed?

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

9 Weighted Quick Union

(a) In terms of runtime, what is the worst way to place the integers 1, 2, 3, 4, and 5 into the same set? Your answer should be in the form of a series of calls to the connect method.

(b) Assume a single node has a height of 0, what is the shortest and tallest height possible for a Quick Union object with 10 elements?

(c) In general, what are the shortest and tallest heights possible for a Quick Union with N elements? What does this mean for the best and worst-case runtimes for `isConnected` and `connect`?

(d) What is the shortest and tallest height possible for a Weighted Quick Union with 10 elements? How about a Weighted Quick Union with N elements? What does this mean for the best and worst-case runtimes for `isConnected` and `connect`?

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

10 Mutual Recursion

Give a tight asymptotic runtime bound for `foo(p, q)` in terms of p and q. Assume p and q are positive.

```
1  public static int foo(int n, int m) {
2      if (m == 0) {
3          return bar(n - 1, n);
4      }
5      return foo(n, m - 1);
6  }
7
8  public static int bar(int x, int y) {
9      if (x == 0) {
10         return 1;
11     }
12     return foo(x, 3 * y);
13 }
```

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

11 Your Node is my Node

We've seen how a simple, encapsulated `Node` class can be used to power very versatile data structures like `LinkedLists` and `BinaryTrees`. Fill in the method, `treeToList`, below such that it destructively mutates the tree while still maintaining its relative ordering.

```
public class BinaryTree<T> {
    protected Node root;
    protected class Node {
        public T value;
        public Node left;
        public Node right;
    }
    public static Node treeToList(Node root) {









    }









}
```

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!