CS 61B Homework 2

Spring 2021 March 15, 2021

1 Asymptotic Warm Up

Give the tightest asymptotic bound on `foo(n)`.

```java
public int foo(int n) {
    if (n == 0) {
        return 0;
    }
    bloop(n);
    return foo(n / 3) + foo(n / 3) + foo(n / 3);
}

public int bloop(int n) {
    for (int i = 0; i < n; i += 1) {
        System.out.println("Ah, loops too");
    }
    return n;
}
```

2 Asymptotic Potpourri

Note: These are hard problems. If you are stuck on it for a long time, move on to other problems, and post on Ed or come to Office Hours so we can help you.

For the following methods, give the runtime of in Θ notation. Your answer should be a function of N that is as simple as possible with no unnecessary leading constants or lower order terms.

(a) Give the runtime of `mystery1(n)` in Θ notation.

```java
public void mystery1(int n) {
    for (int i = n; i > 0; i = i / 2) {
        for (int j = 0; j < 100000000; j += 2) {
            System.out.println("Hello World");
        }
    }
}
```

(b) Give the runtime of `mystery2(n)` in Θ notation.

```java
public void mystery2(int n) {
    for (int i = 1; i < n; i += 1) {
        for (int j = 0; j < n; j += 1) {
            i = i * 2;
            j = j * 2;
        }
    }
}
```

(c) What sum represents the work done by `mystery3(n)`? No need to simplify the sum, just write out the first few terms and the last term.

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

(d) Give the runtime of `mystery4(n)` in Θ notation. Assume that the `SLList` constructor, and the `size` and `addFirst` methods take constant time.

```java
public void mystery4(int n) {
    SLList<Integer> list = new SLList<>();
    for (int i = 1; list.size() < n; i += 1) {
        for (int j = 0; j < i; j += 1) {
            list.addFirst(j);
        }
        System.out.print(list.size() + " + ");
    }
}
```

3 WQU

(a) Draw the Weighted Quick Union object on 0 through 10, that results from the following `connect` calls. Do not use path compression. What is the resulting underlying array? If we connect two sets of equal weight, we will tie-break by making the set whose root has a larger number the parent of the other (the opposite tie-breaking scheme as discussion 6).

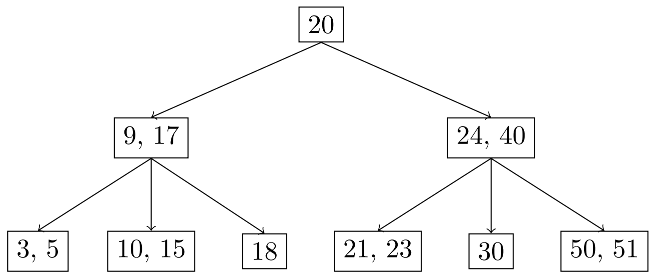
```
connect(0, 1);

connect(2, 3);

connect(9, 5);

connect(5, 7);

connect(7, 1);

connect(4, 2);

connect(3, 1);
```

(b) Assume that a single node has a height of 0. What are the shortest and tallest heights for a Quick Union object with 16 connected elements? What about for a Weighted Quick Union object?

(c) What are the best and worst runtimes for `connect` and `isConnected` in a Quick Union object with N connected elements? What about in a Weighted Quick Union object?

4 Switcheroo

(a) Consider the following 2-3 tree. Convert it to an LLRB, and describe the 6 LLRB operations to balance the tree after inserting the number 11. The LLRB operations are: `rotateRight(x)`, `rotateLeft(x)`, and `colorFlip(x)`.



(b) After inserting 11 and balancing the LLRB, how many nodes are on along the longest path from the root to a leaf.

(c) After inserting 11 and balancing the LLRB, how many red links are on along the longest path from the root to a leaf.

5 Mechanical Hashing

Suppose we insert the following words into an initially empty hash table, in this order: **kerfuffle**, **broom**, **hroom**, **ragamuffin**, **donkey**, **brekky**, **blob**, **zenzizenzizenzic**, and **yap**. Assume that the hash code of a String is just its length (note that this is not actually the hash code for Strings in Java). Use separate chaining to resolve collisions. Assume 4 is the initial size of the hash table's internal array, and double this array's size when the load factor is equal to 1. Illustrate this hash table with a box-and-pointer diagram.

For each index of the final hash table, specify what Strings are stored in it. If it is empty, write "none".