CS 61B Spring 2023 — MT2 Exam Review Session 1

March 8, 2023

1 A-simp-totics

a) What is the runtime of `loopsA`?

```java
public void loopsA(int N) {
    for (int i = N / 2; i < N; i += 1) {
        for (int j = 0; j < N / 2; j += 1) {
            System.out.println("");
        }
    }
}
```

○ $\Theta(1)$ ○ $\Theta(log(logN))$ ○ $\Theta((logN)^2)$ ○ $\Theta(logN)$ ○ $\Theta(N)$
○ $\Theta(NlogN)$ ○ $\Theta(N^2)$ ○ $\Theta(N^2logN)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3 \text{ log N})$
○ $\Theta(N^4)$ ○ $\Theta(N^4 \text{ log N})$ ○ Worse than $\Theta(N^4logN)$ ○ Never terminates

b) What is the runtime of `loopsB`?

```java
public void loopsB(int N) {
    for (int i = 0; i < N / 2; i += 1) {
        for (int j = 0; j < i; j += 1) {
            System.out.println("ello");
        }
    }
}
```

○ $\Theta(1)$ ○ $\Theta(log(logN))$ ○ $\Theta((logN)^2)$ ○ $\Theta(logN)$ ○ $\Theta(N)$
○ $\Theta(NlogN)$ ○ $\Theta(N^2)$ ○ $\Theta(N^2logN)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3 \text{ log N})$
○ $\Theta(N^4)$ ○ $\Theta(N^4 \text{ log N})$ ○ Worse than $\Theta(N^4logN)$ ○ Never terminates

c) What is the best and worst case runtime of `alpha`?

```
public void alpha(int N) {
    if (N % 4 == 0 || N <= 0) {
        return;
    }
    alpha(N - 4);
}
```

Best Case:

○ $\Theta(1)$ ○ $\Theta(log(logN))$ ○ $\Theta((logN)^2)$ ○ $\Theta(logN)$ ○ $\Theta(N)$
○ $\Theta(NlogN)$ ○ $\Theta(N^2)$ ○ $\Theta(N^2logN)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3 \text{ log N})$
○ $\Theta(N^4)$ ○ $\Theta(N^4 \text{ log N})$ ○ Worse than $\Theta(N^4logN)$ ○ Never terminates

Worst Case:

○ $\Theta(1)$ ○ $\Theta(log(logN))$ ○ $\Theta((logN)^2)$ ○ $\Theta(logN)$ ○ $\Theta(N)$
○ $\Theta(NlogN)$ ○ $\Theta(N^2)$ ○ $\Theta(N^2logN)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3 \text{ log N})$
○ $\Theta(N^4)$ ○ $\Theta(N^4 \text{ log N})$ ○ Worse than $\Theta(N^4logN)$ ○ Never terminates

d) What is the best and worst case runtime of `gamma`?

```
public void gamma(int N) {
    if (N % 4 == 0 || N <= 0) {
        return;
    }
    if (N % 2 == 0) {
        gamma(N - 6);
    } else {
        gamma(N - 5);
    }
}
```

Best Case:

○ $\Theta(1)$ ○ $\Theta(log(logN))$ ○ $\Theta((logN)^2)$ ○ $\Theta(logN)$ ○ $\Theta(N)$
○ $\Theta(NlogN)$ ○ $\Theta(N^2)$ ○ $\Theta(N^2logN)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3 \text{ log N})$
○ $\Theta(N^4)$ ○ $\Theta(N^4 \text{ log N})$ ○ Worse than $\Theta(N^4logN)$ ○ Never terminates

Worst Case:

○ $\Theta(1)$ ○ $\Theta(log(logN))$ ○ $\Theta((logN)^2)$ ○ $\Theta(logN)$ ○ $\Theta(N)$
○ $\Theta(NlogN)$ ○ $\Theta(N^2)$ ○ $\Theta(N^2logN)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3 \text{ log N})$
○ $\Theta(N^4)$ ○ $\Theta(N^4 \text{ log N})$ ○ Worse than $\Theta(N^4logN)$ ○ Never terminates

e) Suppose that we call `addToBST` with an empty tree, what is the runtime of the `addToBST`? Assume `contains` and `add` are implemented similarly to `put` and `get` from lab.

```java
public void addToBST(int N, BinarySearchTree<Integer> tree) {
    if (tree.contains(N) || N == 0) {
        return;
    }
    tree.add(N);
    addToBST(N, tree);
    addToBST(N - 1, tree);
}
```

Best Case:

○ $\Theta(1)$ ○ $\Theta(log(logN))$ ○ $\Theta((logN)^2)$ ○ $\Theta(logN)$ ○ $\Theta(N)$
○ $\Theta(NlogN)$ ○ $\Theta(N^2)$ ○ $\Theta(N^2logN)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3$ log N)
○ $\Theta(N^4)$ ○ $\Theta(N^4$ log N) ○ Worse than $\Theta(N^4logN)$ ○ Never terminates

Worst Case:

○ $\Theta(1)$ ○ $\Theta(log(logN))$ ○ $\Theta((logN)^2)$ ○ $\Theta(logN)$ ○ $\Theta(N)$
○ $\Theta(NlogN)$ ○ $\Theta(N^2)$ ○ $\Theta(N^2logN)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3$ log N)
○ $\Theta(N^4)$ ○ $\Theta(N^4$ log N) ○ Worse than $\Theta(N^4logN)$ ○ Never terminates

f) What is the runtime of `recurse`?

```java
public static void recurse(int N) {
    if (N < 11) {
        return;
    }
    cubic(N);
    recurse(N / 2);
    recurse(N / 2);
    recurse(N / 2);
    recurse(N / 2);
}

private static void cubic(int N) {
    for (int i = 0; i < N * N * N; i += 1) {
        System.out.println("schedel");
    }
}
```

○ $\Theta(1)$ ○ $\Theta(log(logN))$ ○ $\Theta((logN)^2)$ ○ $\Theta(logN)$ ○ $\Theta(N)$
○ $\Theta(NlogN)$ ○ $\Theta(N^2)$ ○ $\Theta(N^2logN)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3$ log N)
○ $\Theta(N^4)$ ○ $\Theta(N^4$ log N) ○ Worse than $\Theta(N^4logN)$ ○ Never terminates

2 Hashing

a) Suppose we have the `Dog` class below.

```java
public class Dog {
    private static int dogsCreated = 0;
    private int age;
    private final int microchipID;
    public String name;
    public String owner;

    public Dog(int age, String name, String owner) {
        Dog.dogsCreated += 1;
        this.age = age;
        this.name = name;
        this.owner = owner;
        this.microchipID = dogsCreated;
    }

    public void birthday() {
        this.age += 1;
    }

    @Override
    public boolean equals(Object o) {
        if (!(o instanceof Dog)) {
            return false;
        }
        Dog other = (Dog) o;
        return this.name.equals(other.name) && this.microchipID == other.microchipID;
    }

    @Override
    public int hashCode() {
        // part a
    }
    ...
}
```

For each of the following implementations of the Dog's `hashCode` method, determine if it is good and valid, bad but valid, or invalid.

i)

```java
@Override
public int hashCode() {
    return this.age;
}
```

◯ good and valid ◯ bad but valid ◯ invalid

ii)

```java
@Override
public int hashCode() {
    return this.microchipID;
}
```

◯ good and valid ◯ bad but valid ◯ invalid

iii)

```java
@Override
public int hashCode() {
    return (int) this.name.charAt(0);
}
```

◯ good and valid ◯ bad but valid ◯ invalid

iv)

```java
@Override
public int hashCode() {
    return this.owner.hashCode();
}
```

◯ good and valid ◯ bad but valid ◯ invalid

b) If a `HashMap` contains two or more items with equal `keys`, how might have this happened? Give one reason. Assume that we are referring to the built in java `HashMap`. Note that this part is independent from the previous, i.e. the `keys` aren't necessarily `Dogs`.

`Reason:`

c) Implement the method `isUnique` in the `HashMap` class that returns `true` if and only if the `HashMap` has **unique** keys, i.e. it does *not* contain two or more equal `keys`. Assume all implementations of the `List` and `Set` interface have been imported.

Hint: Consider objects with *invalid* hashCodes.

```
public class 61BLHashMap<K, V> implements Map61BL<K, V> {
    LinkedList<Entry<K, V>>[] map;
    int size;

    public boolean isUnique() {

        _________________________________________________________;

        for (_______________________________________________________) {

            for (_____________________________________________________) {

                if (__________________________________________________) {

                    return _______________________________________________;
                }
                ______________________________________________________;
            }
        }
        return ________________________________________________________;
    }

    ...

    private static class Entry<K, V> {
        private K key;
        private V value;

        Entry(K key, V value) {
            this.key = key;
            this.value = value;
        }
    }
}
```

3 Disjoint Sets

For all parts of this question, assume sets implementation **do not** use path compression, and the `WeightedQuickUnion` also implements `DisjointSets`.

Consider the following `WeightedQuickUnion` array state that uses negative values to denote set size {`-2, 2, 5, 0, 5, -4`}.

a) How many connected components are there?

◯ 1 ◯ 2 ◯ 3 ◯ 4 ◯ 5 ◯ 6

b) Does `isConnected(2, 3)` evaluate to `true` or `false`?

◯ `true` ◯ `false`

c) Suppose we call `connect(0, 1)`. What is the resultant array? If we connect two sets of the same size, put the smaller number as a child of the other. Format your answer like so {`-2, 2, 5, 0, 5, -4`}.

Array: ____________________________

4 Fall 2022 MT2 Q8

Suppose we want to add a new operation to the `LinkedListDeque` from Proj1A, specifically a `removeEvery(int x)` operation.

For simplicity, assume our `LinkedListDeque` is hard coded to use integer values. This method should remove all instances of x from the list. Describe an implementation of the modified `LinkedListDeque` class below. Your methods must complete in $O(k + \log N)$ time on average (i.e. it's OK to ignore any occasional resize operation), where `k` is the number of `x`'s present in the `Deque`, and `N` is the number of items in the `Deque`.

In other words, the methods below must be no worse than linear with respect to k, and no worse than logarithmic with respect to N. It's OK if your methods are faster than the requirement. No credit will be given for naïve solutions (e.g. $\Theta(N)$ runtime, since this is linear with respect to N).

As a reminder, the `LinkedListDeque` class is partially defined below:

```
public class LinkedListDeque {
    private Node sentinel;
    private int size;
    public void addFirst(int item) { ... }
    public int removeFirst() { ... }
    ...
}
```

(a) What additional instance variable do you need? List a single instance variable, **including the type and name**. You may use only one! You may use any data structure from the reference sheet or any primitive type.

(b) Describe below how the addFirst(int x) operation is different (if at all) from the addFirst operation in a normal LinkedListDeque. If you use your instance variable from part a, use the name you gave to that instance variable. If there are no differences, just write "no difference".

(c) Describe below your `removeEvery(int x)` operation. If you use your instance variable from part a, use the name you gave to that instance variable.