Section 3

Friday, April 12, 2019 9:14 PM

section03

Section 03: Asymptotic Analysis

Review Problems

1. Code Analysis

For each of the following code blocks, what is the worst-case runtime? Give a big-Θ bound.

(a)
```java
public IList<String> repeat(DoubleLinkedList<String> list, int n) {
    IList<String> result = new DoubleLinkedList<String>();
    for(String str : list) {
        for(int i = 0; i < n; i++) {
            result.add(str);
        }
    }
    return result;
}
```

(b)
```java
public void foo(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 5; j < i; j++) {
            System.out.println("Hello!");
        }

        for (int j = i; j >= 0; j -= 2) {
            System.out.println("Hello!");
        }
    }
}
```

(c)
```java
public int num(int n){
    if (n < 10) {
        return n;
    } else if (n < 1000) {
        return num(n - 2);
    } else {
        return num(n / 2);
    }
}
```

(d)
```java
public int foo(int n) {
    if (n <= 0) {
        return 3;
    }
    int x = foo(n - 1);
    System.out.println("hello");
    x += foo(n - 1);
    return x;
}
```

1

2. Binary Search Trees

(a) Write a method `validate` to validate a BST. Although the basic algorithm can be converted to any data structure and work in any format, if it helps, you may write this method for the `IntTree` class:

```java
public class IntTree {
    private IntTreeNode overallRoot;

    // constructors and other methods omitted for clarity

    private class IntTreeNode {
        public int data;
        public IntTreeNode left;
        public IntTreeNode right;

        // constructors omitted for clarity
    }
}
```

Section Problems

3. Modeling recursive functions

(a) Consider the following method.

```java
public static int f(int n) {
    if (n == 0) {
        return 0;
    }

    int result = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++) {
            result++;
        }
    }
    return 5 * f(n / 2) + 3 * result + 2 * f(n / 2);
}
```

$T(n) = \begin{cases} c_1 & \text{if } n = 0 \\ \frac{n(n-1)}{2} + 2\cdot T(\frac{n}{2}) & \text{else} \end{cases}$

(i) Find a recurrence $T(n)$ modeling the worst-case runtime of `f(n)`. **Hint:** You may need to use Gauss's summation identity (see the last page).

(ii) Find a recurrence $W(n)$ modeling the *returned integer output* of `f(n)`.

$i=0: 0 + i=1: 1 + i=2: 2 + \ldots + i=n-1: n-1 = \frac{n(n-1)}{2}$ (Gauss)

2

(b) Consider the following method.

```java
public static int g(n) {
    if (n <= 1) {
        return 1000;
    }
    if (g(n / 3) > 5) {
        for (int i = 0; i < n; i++) {
            System.out.println("Hello");
        }
        return 5 * g(n / 3);
    } else {
        for (int i = 0; i < n * n; i++) {
            System.out.println("World");
        }
        return 4 * g(n / 3);
    }
}
```

(i) Find a recurrence $S(n)$ modeling the worst-case runtime of `g(n)`.

(ii) Find a recurrence $X(n)$ modeling the *returned integer output* of `g(n)`.

(iii) Find a recurrence $P(n)$ modeling the *printed output* of `g(n)`.

(c) Consider the following set of recursive methods.

```java
public int test(int n) {
    IDictionary<Integer, Integer> dict = new AvlDictionary<>();
    populate(n, dict);
    int counter = 0;
    for (int i = 0; i < n; i++) {
        counter += dict.get(i);
    }
    return counter;
}

private void populate(int k, IDictionary<Integer, Integer> dict) {
    if (k == 0) {
        dict.put(0, k);
    } else {
        for (int i = 0; i < k; i++) {
            dict.put(i, i);
        }
        populate(k / 2, dict);
    }
}
```

(i) Write a mathematical function representing the *worst-case runtime* of `test`.

You should write two functions, one for the runtime of `test` and one for the runtime of `populate`.

3

4. Master Theorem

For each of the recurrences below, use the Master Theorem to find the big-Θ of the closed form or explain why Master Theorem doesn't apply. (See the last page for the definition of Master Theorem.)

(a) $T(n) = \begin{cases} 18 & \text{if } n \le 5 \\ 3T(n/4) + n^2 & \text{otherwise} \end{cases}$

(b) $T(n) = \begin{cases} 1 & \text{if } n \le 1 \\ 9T(n/3) + n^2 & \text{otherwise} \end{cases}$

(c) $T(n) = \begin{cases} 1 & \text{if } n \le 1 \\ \log(n)T(n/2) + n & \text{otherwise} \end{cases}$

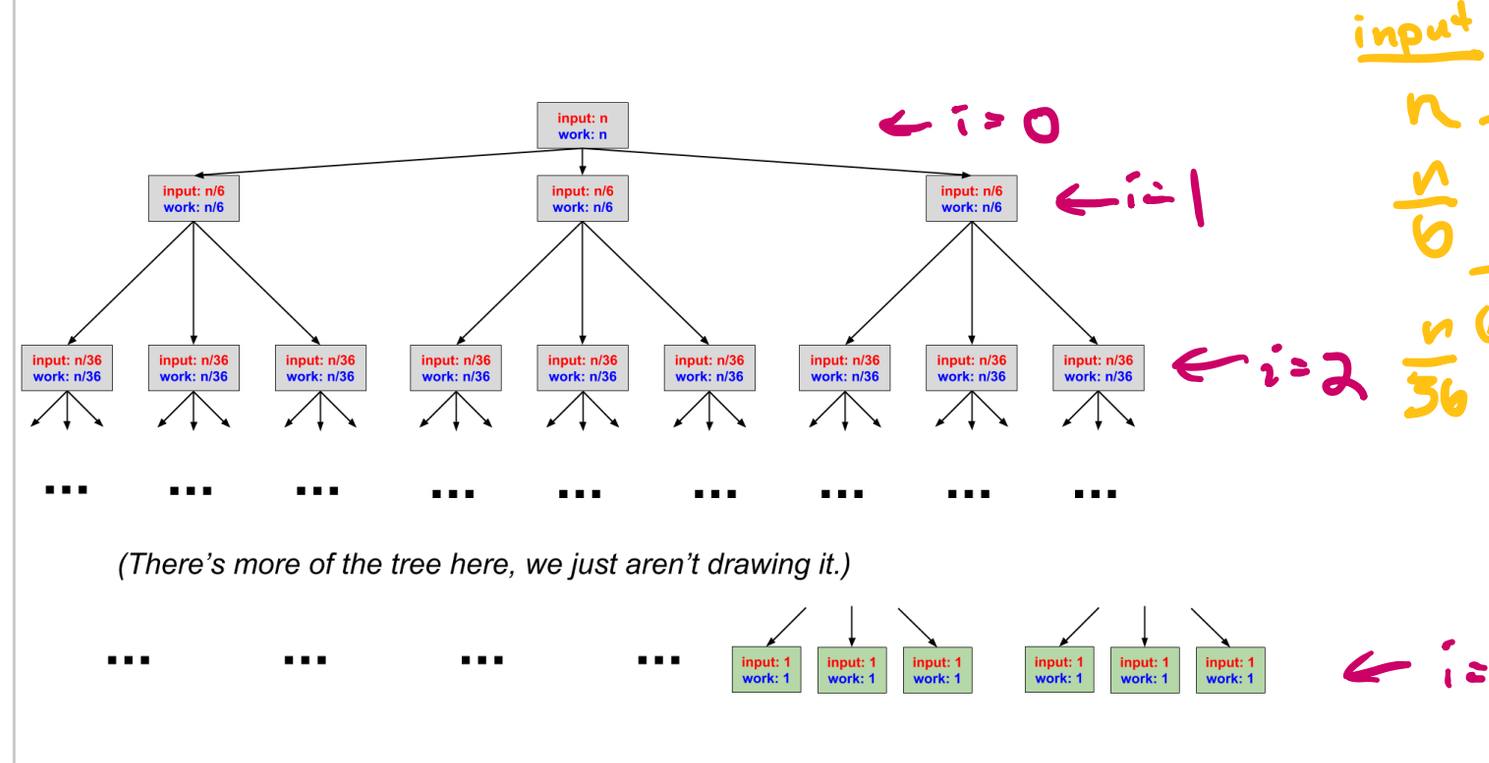
(d) $T(n) = \begin{cases} 1 & \text{if } n \le 19 \\ 4T(n/3) + n & \text{otherwise} \end{cases}$

(e) $T(n) = \begin{cases} 5 & \text{if } n \le 24 \\ 2T(n-2) + 5n^3 & \text{otherwise} \end{cases}$

5. Tree method walk-through

Consider the following recurrence: $A(n) = \begin{cases} 1 & \text{if } n \le 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$

We want to find an *exact* closed form of this equation by using the tree method. Suppose we draw out the total work done by this method as a tree, as discussed in lecture. Let n be the initial input to A.



(There's more of the tree here, we just aren't drawing it.)

(a) What is the size of the input at level i (as in class, call the root level 0)? $\frac{n}{6^i}$

(b) What is the number of nodes at level i?

Note: let $i = 0$ indicate the level corresponding to the root node. So, when $i = 0$, your expression should be equal to 1. 3^i

(c) What is the total work at the i^{th} **recursive** level? $3^i \times \left(\frac{n}{6^i}\right)$ (# nodes × work per node)

(d) What is the last level of the tree? $i = \log_6 n$

Part d: $\frac{n}{6^i} = 1$ (input @ level i) → $n = 6^i$ → $\log_6 n = i$

(e) What is the work done in the base case? $3^{\log_6 n} \times 1$ (# nodes × work per node)

(f) Combine your answers from previous parts to get an expression for the total work.

$\left(\sum_{i=0}^{\log_6 n - 1} 3^i \times \frac{n}{6^i}\right) + 3^{\log_6 n}$ (Recursive + Base Case)

4

(g) Simplify to a closed form.

Note: you do not need to simplify your answer, once you found the closed form. Hint: You should use the finite geometric series identity somewhere while finding a closed form.

(h) Use the master theorem to find a big-Θ bound of $A(n)$. (See the last page for the definition of Master Theorem.)

6. More tree method recurrences

For each of the following recurrences, find their closed form using the tree method. Then, check your answer using the master method (if applicable). It may be a useful guide to use the steps from section 4 of this handout to help you with all the parts of solving a recurrence problem fully.

(a) $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 3 & \text{otherwise} \end{cases}$

(b) $S(q) = \begin{cases} 1 & \text{if } q = 1 \\ 2S(q-1) + 1 & \text{otherwise} \end{cases}$

(c) $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 8T(n/2) + 4n^2 & \text{otherwise} \end{cases}$

Food for Thought

7. TreeMap implemented as a Binary Search Tree

Consider the following method, which is a part of a Binary Search Tree implementation of a TreeMap class.

```java
public V find(K key) {
    return find(this.root, key);
}

private V find(Node<K, V> current, K key) {
    if (current == null) {
        return null;
    }
    if (current.key.equals(key)) {
        return current.value;
    }
    if (current.key.compareTo(key) > 0) {
        return find(current.left, key);
    } else {
        return find(current.right, key);
    }
}
```

(a) We want to analyze the runtime of our `find(x)` method in the best possible case and the worst possible case. What does our tree look like in the best possible case? In the worst possible case?

5

(b) Write a recurrence to represent the worst-case runtime for `find(x)` in terms of n, the number of elements contained within our tree. Then, provide a Θ bound.

(c) Assuming we have an optimally structured tree, write a recurrence for the runtime of `find(x)` (again in terms of n). Then, provide a Θ bound.

Challenge Problems

8. Modeling recursive functions

Consider the following recursive function. You may assume that the input will be a multiple of 3.

```java
public int test(int n) {
    if (n <= 6) {
        return 2;
    } else {
        int curr = 0;
        for (int i = 0; i < n * n; i++) {
            curr += 1;
        }
        return curr + test(n - 3);
    }
}
```

(a) Write a recurrence modeling the *worst-case runtime* of `test`.

(b) Unfold the recurrence into a summation (for $n \ge 6$).

(c) Simplify the summation into a closed form (for $n \ge 6$).

9. Recurrences

For the following recurrence, use the unfolding method to first convert the recurrence into a summation. Then, find a big-Θ bound on the function in terms of n. Assume all division operations are integer division.

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2T(n/3) + n & \text{otherwise} \end{cases}$$

6

Master Theorem

For recurrences in this form, where a, b, c, e are constants:

$$T(n) = \begin{cases} d & \text{if } n \le \text{some constant} \\ aT(n/b) + e \cdot n^c & \text{otherwise} \end{cases} \qquad T(n) \text{ is } \begin{cases} \Theta(n^c) & \text{if } \log_b(a) < c \\ \Theta(n^c \log n) & \text{if } \log_b(a) = c \\ \Theta(n^{\log_b(a)}) & \text{if } \log_b(a) > c \end{cases}$$

Useful summation identities

Splitting a sum

$$\sum_{i=a}^{b}(x+y) = \sum_{i=a}^{b} x + \sum_{i=a}^{b} y$$

Adjusting summation bounds

$$\sum_{i=a}^{b} f(x) = \sum_{i=0}^{b} f(x) - \sum_{i=0}^{a-1} f(x)$$

Factoring out a constant

$$\sum_{i=a}^{b} cf(i) = c\sum_{i=a}^{b} f(i)$$

Summation of a constant

$$\sum_{i=0}^{n-1} c = \underbrace{c + c + \ldots + c}_{n \text{ times}} = cn$$

Note: this rule is a special case of the rule on the left

Gauss's identity

$$\sum_{i=0}^{n-1} i = 0 + 1 + \ldots + n - 1 = \frac{n(n-1)}{2}$$

Sum of squares

$$\sum_{i=0}^{n-1} i^2 = \frac{n(n-1)(2n-1)}{6}$$

Finite geometric series

$$\sum_{i=0}^{n-1} x^i = \frac{x^n - 1}{x - 1}$$

Infinite geometric series

$$\sum_{i=0}^{\infty} x_i = \frac{1}{1-x}$$

Note: applicable only when $-1 < x < 1$

7

quickcheck 03

Quickcheck 03: Analysis

Your friend used summations to make a model for the running time of one of their functions. Simplify their summations into a final (exact) closed form. Then determine what the big-$\mathcal{O}$ of the function is.

$$\sum_{i=0}^{n-1}(6i^2 - 3) + \sum_{i=0}^{n-1} \frac{2^i}{n} + \sum_{i=0}^{n/2-1}\sum_{j=0}^{i-1} 1$$

(A, B, C)

Here is a list of identities that may be useful:

Manipulating Sums:

$$\sum_{i=a}^{b}(x+y) = \sum_{i=a}^{b} x + \sum_{i=a}^{b} y \qquad \sum_{i=a}^{b} f(i) = \sum_{i=0}^{b} f(i) - \sum_{i=0}^{a-1} f(i) \qquad \sum_{i=a}^{b} c \cdot f(i) = c\sum_{i=a}^{b} f(i)$$

Geometric Series Identities:

$$\sum_{i=0}^{n-1} x^i = \frac{x^n - 1}{x - 1} \qquad \sum_{i=0}^{\infty} x^i = \frac{1}{1-x} \text{ if } -1 < x < 1$$

Other Common Summations:

$$\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2} \qquad \sum_{i=0}^{n-1} i^2 = \frac{n(n-1)(2n-1)}{6} \qquad \sum_{i=0}^{n-1} c = cn$$

A) $\sum_{i=0}^{n-1}(6i^2 - 3)$

$\sum_{i=0}^{n-1} 6i^2 - \sum_{i=0}^{n-1} 3$

$6\sum_{i=0}^{n-1} i^2 - 3 \cdot n$

$6 \cdot \left[\frac{n(n-1)(2n-1)}{6}\right] - 3n$

Another question

Do you have any questions about this course? It could be about policy, content, instructors, TAs, etc.

B) $\sum_{i=0}^{n-1} \frac{2^i}{n}$

2

C) $\sum_{i=0}^{\frac{n}{2}-1} \sum_{j=0}^{i-1} 1$