

# 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- $\Theta$  bound.

(a) 

```
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) 

```
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) 

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

(d) 

```
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;
}
```

## 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:

```
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. Recurrences

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

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

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

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

$$(d) T(n) = \begin{cases} 1 & \text{if } n = 0 \\ T(n/3) + 4 & \text{otherwise} \end{cases}$$

$$(e) T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n-1) + 1 & \text{otherwise} \end{cases}$$

$$(f) T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n/2) + 100 & \text{otherwise} \end{cases}$$

## 4. Modeling recursive functions

(a) Consider the following method.

```
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 += j;
        }
    }
    return 5 * f(n / 2) + 3 * result + 2 * f(n / 2);
}
```

(i) Find a recurrence  $T(n)$  modeling the worst-case runtime of  $f(n)$ .

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

(b) Consider the following method.

```
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.

```

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.

(ii) Write a mathematical function  $Y(n)$  representing the *returned integer output* of test.

## 5. AVL Trees

(a) Draw an AVL Tree as each of the following keys are added in the order given. Show intermediate steps.

(i)

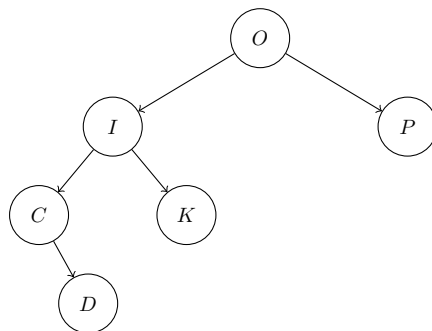
{13, 17, 14, 19, 22, 18, 11, 10, 21}

(ii)

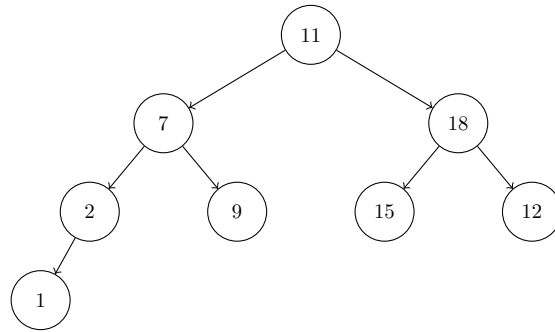
{1, 2, 3, 4, 5, 6}

(b) Identify if the following trees are AVL trees. Explain your answer.

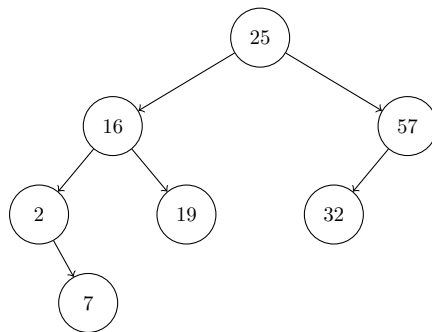
(i) Tree 1



(ii) Tree 2



(iii) Tree 3



## Food for thought

### 6. AVL trees

What is the minimum number of nodes in an AVL tree, given the following heights? Draw a picture of such a tree. (Reminder: an AVL tree's height is 0 for a tree with only 1 node in it!)

(a) 1

(b) 2

(c) 4

### 7. Algorithm Design

(a) Given a binary search tree, describe how you could convert it into an AVL tree with worst-case time  $\mathcal{O}(n \log(n))$ . What is the best case runtime of your algorithm?

(b) Given an AVL tree, describe how would you do a level order tree traversal. What is the worst-case runtime of your algorithm?

## Challenge Problems

### 8. Recurrences

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

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

### 9. Modeling recursive functions

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

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

## 10. AVL Trees

- (a) Is there a relationship between an AVL tree's height, and its minimum or maximum number of nodes? If so, what is it?
- (b) Write a method `isAVLTree` to check if a given tree (which is guaranteed to be a valid BST) is a valid AVL tree. If it helps, you may write this method for this tree class, `HeightTree`, which keeps track of the height of a tree at each node:

```
public class HeightTree {
    private IntHeightNode overallRoot;

    // constructors and other methods omitted for clarity

    private class IntHeightNode {
        public int data;
        public int height;
        public IntHeightNode left;
        public IntHeightNode right;

        // constructors omitted for clarity
    }
}
```

- (c) Now write `isAVLTree` *without* assuming that the tree is a valid BST.
- (d) Now write `isAVLTree` for the `IntTree` class (you may assume again that the tree is guaranteed to be a valid BST).