Section 02: Asymptotic Analysis

General Concepts

1. Simple Definitions

(a) What are $\mathcal{O}$ Θ Ω? What are the differences between these three?

(b) What is the meaning of tight bounds? Give two lower bounds, two upper bounds and theta bound for BSTs. If part of the question is not applicable, why?

(c) In the worst case, is finding an element in a sorted array using binary search $\mathcal{O}(n)$? In the worst case, is finding an element in a sorted array using binary search $\Omega(n)$?

2. Comparing growth rates

(a) Order each of the following functions from fastest to slowest in terms of asymptotic growth. (By "fastest", we mean which function increases the most rapidly as n increases.)

- $\log_4(n) + \log_2(n)$
- $\frac{n}{2} + 4$
- $2^n + 3$
- $750,000,000$
- $8n + 4n^2$

(b) For each of the above expressions, state the simplified tight $\mathcal{O}$ bound in terms of n.

(c) Order each of these more esoteric functions from fastest to slowest in terms of asymptotic growth. (By "fastest", we mean which function increases the most rapidly as n increases.) Also state a simplified tight $\mathcal{O}$ bound for each.

- $2^{n/2}$
- 3^n
- 2^n

3. True or false?

(a) If a function is in $\Omega(n)$, then it could also be in $\mathcal{O}(n^2)$.

(b) If a function is in $\Theta(n)$, then it could also be in $\mathcal{O}(n^2)$.

(c) If a function is in $\Omega(n)$, then it is always in $\mathcal{O}(n)$.

Algorithm Analysis without Recurrence

4. Modeling code

For each of the following code blocks, give a summation that represents the worst-case runtime in terms of n, then evaluate the summation to give a tight big-O bound of it.

(a)
```
int x = 0;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        x++;
    }
}
```

(b)
```
int x = 0;
for (int i = n; i >= 1; i /= 2) {
    x += i;
}
```

(c)
```
int x = 0;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n * n / 3; j++) {
        x += j;
    }
}
```

(d)
```
int x = 0;
for (int i = n; i >= 0; i -= 1) {
    if (i % 3 == 0) {
        break;
    } else {
        x += n;
    }
}
```

(e)
```
int x = 0;
for (int i = 0; i < n; i++) {
    if (i % 5 == 0) {
        for (int j = 0; j < n; j++) {
            if (i == j) {
                x += i * j;
            }
        }
    }
}
```

(f)
```
int x = 0;
for (int i = 0; i < n; i++) {
    if (n < 100000) {
        for (int j = 0; j < n; j++) {
            x += 1;
        }
    } else {
        x += 1;
    }
}
```

(g)
```
int x = 0;
if (n % 2 == 0) {
    for (int i = 0; i < n * n * n * n; i++) {
        x++;
    }
} else {
    for (int i = 0; i < n * n * n; i++) {
        x++;
    }
}
```

Algorithm Analysis with Recurrence

5. Practicing Recurrence Equations

With recursive or divide-and-conquer algorithms, it is often helpful to express the runtime recursively with a recurrence equation.

Identify recurrence equations for the following programs, then unroll your recurrence equations using the expansion or tree method to find a big-O bound.

(a) Consider the function f

```
f(n) {
    if (n <= 0) {
        return 1;
    }
    return 2 * f(n - 1) + 1;
```

(b) Consider the function g

```
g(n) {
    if (n <= 1) {
        return 1000
    }
    if (g(n/3) > 5) {
        for (int i = 0; i < n; i++) {
            println("Yay!")
        }
        return 5 * g(n/3)
    }
    else {
        for (int i = 0; i < n * n; i++) {
            println("Yay!")
        }
        return 4 * g(n/3)
    }
}
```

6. Unrolling Recurrences

Unroll the following recurrence equations. For these problems, you should show work and find a closed form, not a big-O bound.

(a) Unroll the following recurrence

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

(b) Unroll the following recurrence

$$T(n) = \begin{cases} c_0 & \text{if } n \leq 1 \\ 2T\left(\frac{n}{2}\right) + c_1 & \text{otherwise} \end{cases}$$

(c) Unroll the following recurrence

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

7. Applying definitions

For each of the following, choose a c and n_0 which show $f(n) \in \mathcal{O}(g(n))$. Explain why your values of c and n_0 work.

(a) $f(n) = 3n + 4, g(n) = 5n^2$

(b) $f(n) = 33n^3 + \sqrt{n} - 6, g(n) = 17n^4$

(c) $f(n) = 17\log(n), g(n) = 32n + 2n\log(n)$

Binary Search Tree

8. Binary Search Tree (BST) Practice Problems

Consider the definition for a Binary Tree Node as below:

```
public class TreeNode {
    int value;
    TreeNode left;
    TreeNode right;

    TreeNode(int n) {
        value = n;
    }
}
```

(a) Search in BST: Given the root node of a BST and a value, write an efficient method $searchBST$ to find the node in the BST that the node's value is the same as the given value. Return true if such node exist, else false. Lastly, provide the complexity analysis of your solution (Time complexity, Space complexity).

```
public boolean searchBST(TreeNode root, int val) {
    // begin your code
}
```

(b) Validate BST: Given a binary tree, write an efficient method $isValidBST$ to determine if it is a valid binary search tree (BST). Lastly, provide the complexity analysis of your solution (Time complexity, Space complexity).

```
public boolean isValidBST(TreeNode root) {
    // begin your code
}
```

(c) Inorder Traversal in BST: Given a binary tree, write an efficient method $inorderTraversalBST$ to return the inorder traversal of its nodes' values. Lastly, provide the complexity analysis of your solution (Time complexity, Space complexity).

```
public List<Integer> inorderTraversalBST(TreeNode root) {
    // begin your code
}
```