CS 61B
Fall 2025

Asymptotics II and BSTs

Exam-Level Discussion 06: October 06, 2025

1 Re-cursed with Asymptotics

(a) What is the runtime of the code below in terms of `n`?

```
public static int curse(int n) {
    if (n <= 0) {
        return 0;
    } else {
        return n + curse(n - 1);
    }
}
```

(b) Can you find a runtime bound for the code below? We can assume the `System.arraycopy` method takes $\Theta(N)$ time, where N is the number of elements copied. The official signature is `System.arrayCopy(Object sourceArr, int srcPos, Object dest, int destPos, int length)`. Here, `srcPos` and `destPos` are the starting points in the source and destination arrays to start copying and pasting in, respectively, and `length` is the number of elements copied.

```
public static void silly(int[] arr) {
    if (arr.length <= 1) {
        System.out.println("You won!");
        return;
    }

    int newLen = arr.length / 2;
    int[] firstHalf = new int[newLen];
    int[] secondHalf = new int[newLen];

    System.arraycopy(arr, 0, firstHalf, 0, newLen);
    System.arraycopy(arr, newLen, secondHalf, 0, newLen);

    silly(firstHalf);
    silly(secondHalf);
}
```

(c) Given that `exponentialWork` runs in $\Theta(3^N)$ time with respect to input N, what is the runtime of `amy`?

```
public void ronnie(int N) {
    if (N <= 1) {
        return;
    }
    amy(N - 2);
    amy(N - 2);
    amy(N - 2);
    exponentialWork(N); // Runs in $Theta(3^N)$ time
}
```

2 Asymptotics is Fun!

(a) Using the function `g` defined below, what is the runtime of the following function calls? Write each answer in terms of `N`. Feel free to draw out the recursion tree if it helps.

```
public static void g(int N, int x) {
    if (N == 0) {
        return;
    }
    for (int i = 1; i <= x; i++) {
        g(N - 1, i);
    }
}
```

`g(N, 1)`: Θ(____)

`g(N, 2)`: Θ(____)

(b) Suppose we change line 6 to `g(N - 1, x)` and change the stopping condition in the for loop to `i <= f(x)` where `f` returns a random number between 1 and `x`, inclusive. For the following function calls, find the tightest Ω and big O bounds. Feel free to draw out the recursion tree if it helps.

```
public static void g(int N, int x) {
    if (N == 0) {
        return;
    }
    for (int i = 1; i <= f(x); i++) {
        g(N - 1, x);
    }
}
```

`g(N, 2)`: Ω(____), O(____)

`g(N, N)`: Ω(____), O(____)

3 Is this a BST?

In this setup, assume a **BST** (Binary Search Tree) has a **key** (the value of the tree root represented as an **int**) and pointers to two other child BSTs, **left** and **right**. Additionally, assume that **key** is between **Integer.MIN_VALUE** and **Integer.MAX_VALUE** non-inclusive.

(a) The following code should check if a given binary tree is a BST. However, for some trees, it returns the wrong answer. Give an example of a binary tree for which **brokenIsBST** fails.

```
public static boolean brokenIsBST(BST tree) {
    if (tree == null) {
        return true;
    } else if (tree.left != null && tree.left.key >= tree.key) {
        return false;
    } else if (tree.right != null && tree.right.key <= tree.key) {
        return false;
    } else {
        return brokenIsBST(tree.left) && brokenIsBST(tree.right);
    }
}
```

(b) Now, write **isBST** that fixes the error encountered in part (a).

Hint: You will find **Integer.MIN_VALUE** and **Integer.MAX_VALUE** helpful.

Hint 2: You want to somehow store information about the keys from previous layers, not just the direct parent and children. How do you use the parameters given to do this?

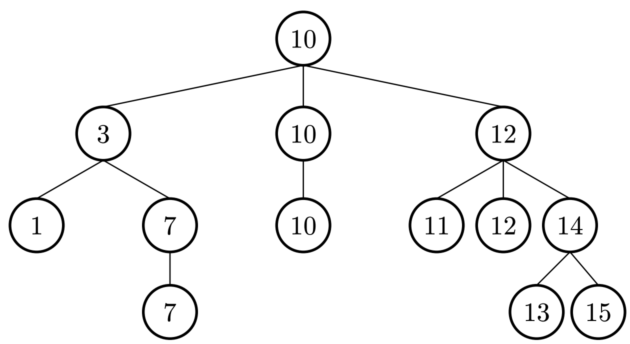
```
public static boolean isBST(BST T) {
    return isBSTHelper(____________________________________________);
}

public static boolean isBSTHelper(BST T, int min, int max) {

    if (________________________________________) {

        __________________________________________________

    } else if (___________________________________________________) {

        __________________________________________________

    } else {

        __________________________________________________

    }
}
```

4 Extra: Tri-nary Search Tree

We'd like a data structure that acts like a BST (Binary Search Tree) in terms of operation runtimes but allows duplicate values. Therefore, we decide to create a new data structure called a TST (Trinary Search Tree), which can have up to three children, which we'll refer to as **`left`**, **`middle`**, and **`right`**. In this setup, we have the following invariants, which are very similar to the BST invariants:

1. Each node in a TST is a root of a smaller TST
2. Every node to the **`left`** of a root has a value "lesser than" that of the root
3. Every node to the **`right`** of a root has a value "greater than" that of the root
4. **Every node to the `middle` of a root has a value equal to that of the root**

Below is an example TST to help with visualization.



Describe an algorithm that will print the elements in a TST in **descending** order. *(Hint: recall that an in-order traversal for a BST gives elements in increasing order.)*