ADTs, Asymptotics II, BSTs

Exam-Level 06: October 7, 2024

1 Finish the Runtimes

Below we see the standard nested for loop, but with missing pieces!

```
for (int i = 1; i < ______; i = ______) {
    for (int j = 1; j < ______; j = ______) {
        System.out.println("Circle is the best TA");
    }
}
```

For each part below, **some** of the blanks will be filled in, and a desired runtime will be given. Fill in the remaining blanks to achieve the desired runtime! There may be more than one correct answer.

Hint: You may find `Math.pow` helpful.

(a) Desired runtime: $\Theta(N^2)$

```
for (int i = 1; i < N; i = i + 1) {              → N
    for (int j = 1; j < i; j = __+1__) {         → N
        System.out.println("This is one is low key hard");
    }
}
```

(b) Desired runtime: $\Theta(\log(N))$

```
for (int i = 1; i < N; i = i * 2) {              → Θ(log n)
                                                   any constant
    for (int j = 1; j < __C__; j = j * 2) {      → O(1)
        System.out.println("This is one is mid key hard");
    }
}
```

(c) Desired runtime: $\Theta(2^N)$. $\frac{2^N}{N}$ is a valid answer, could you think of another?

```
for (int i = 1; i < N; i = i + 1) {
    for (int j = 1; j < pow(2,i); j = j + 1) {
        System.out.println("This is one is high key hard");
    }
}
```

$1 + 2 + 4 + \ldots 2^{n-1} + 2^n = \Theta(2^n)$

$i = 1, 2, \ldots N-1, N$

$2^0 + 2^1 + 2^2 + \ldots 2^{n-1} + 2^n$

(d) Desired runtime: $\Theta(N^3)$

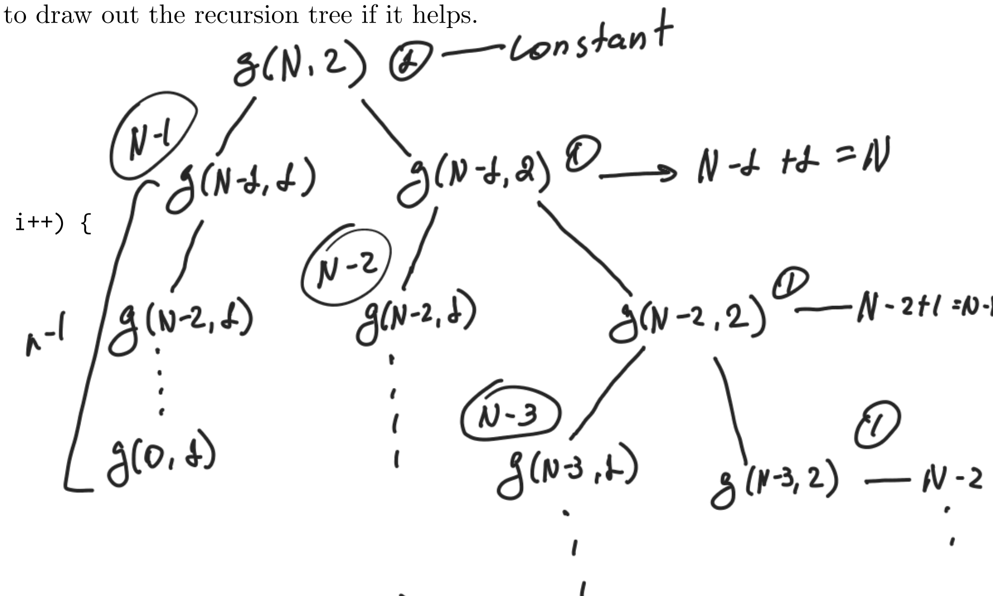
```
for (int i = 1; i < Math.pow(2,N); i = i * 2) {
    for (int j = 1; j < N * N; j = __+1__) {     → N^2, and so we need outer loop to be N
        System.out.println("yikes");
    }
}
```

$\underbrace{1, 2, 4, 8 \ldots 2^n}_{n \text{ iterations}}$

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.

```
1  void g(int N, int x) {
2      if (N == 0) {
3          return;
4      }
5      for (int i = 1; i <= x; i++) {
6          g(N - 1, i);
7      }
8  }
```



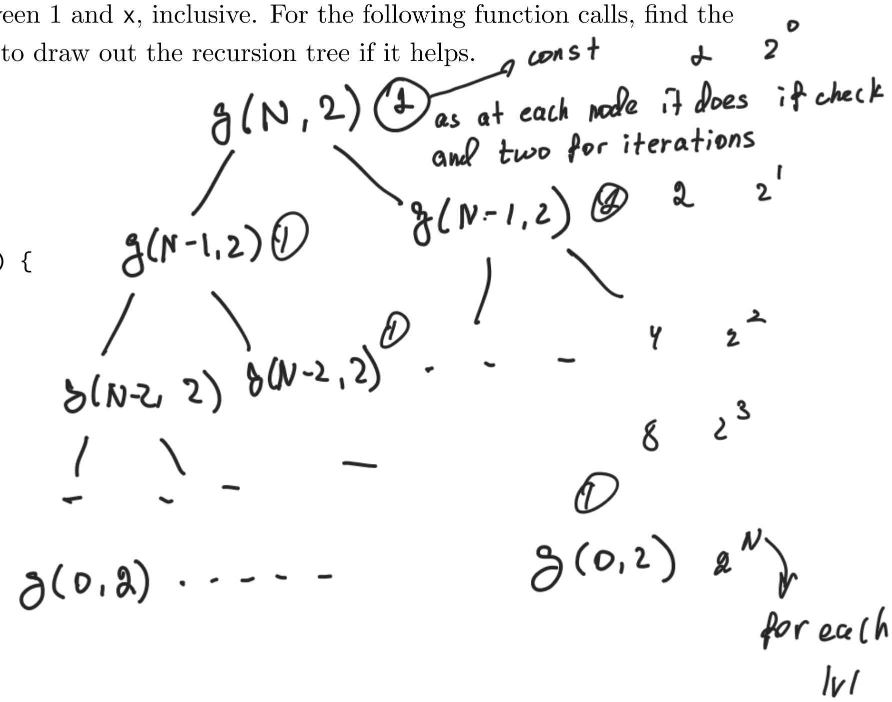
`g(N, 1)`: $\Theta(N)$

N times
g(N, 1)
↓
g(N-1, 1)
⋮
g(0, 1)

`g(N, 2)`: $\Theta(N^2)$ $N + (N-1) + (N-2) + \ldots = \Theta(N^2)$

(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.

```
1  void g(int N, int x) {
2      if (N == 0) {
3          return;
4      }
5      for (int i = 1; i <= f(x); i++) {
6          g(N - 1, x);
7      }
8  }
```



`g(N, 2)`: $\Omega(N)$, $O(2^N)$

↑
from the previous part
↓

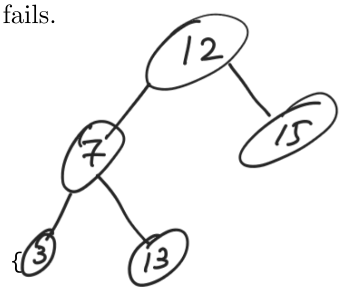
`g(N, N)`: $\Omega(N)$, $O(N^N)$

same as with before, its just we've N instead of 2
and at each lvl, with nodes work done is N and not 1
and sum is $N^N + N^{N-1} + N^{N-2} \ldots + N = N^N$

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`.

(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);
    }
}
```

in this case we recursively check and we get no error, but this clearly isnt BST if we check for node 7, it still returns true, but its an error

(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(T, Intger.Min_Val, Int.Max_Val);
}

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

    if (T = null) {

        return true

    } else if (T.key < min || T.key > max) {

        return false

    } else {

        return isBSTHelper(T.left, min, T.key)
            && isBSTHelper(T.right, T.key, max)
    }
}
```

if on the left its lesser