ADTs, Asymptotics II, BSTs

Exam Level 07

Announcements

- Weekly Survey 6 due Monday 10/2
- HW2 due Wednesday 10/4
- Lab 7 due Friday 10/6
- Project 2A due Wednesday 10/11
- Seriously start Proj2 :’)

Content Review

Asymptotics Advice

- Asymptotic analysis is only valid on very large inputs, and comparisons between runtimes is only useful when comparing inputs of different orders of magnitude.
- Use Θ where you can, but won't always have tight bound (usually default to O)
- Reminder: total work done = sum of all work per iteration or recursive call
- While common themes are helpful, rules like "nested for loops are always N^2" can easily lead you astray (pay close attention to stopping conditions and how variables update)
- Drop lower-order terms (ie. $n^3 + 10000n^2 - 5000000 \rightarrow \Theta(n^3)$)

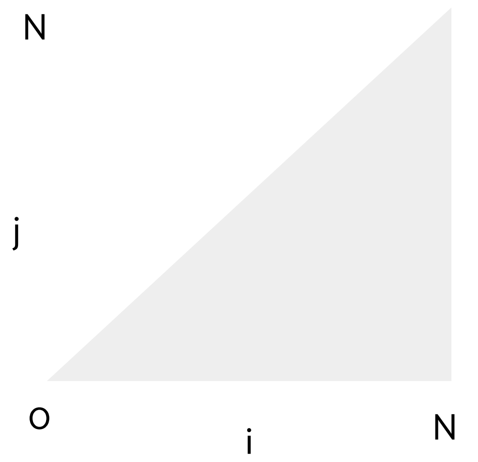
Asymptotics Advice

- For recursive problems, it's helpful to draw out the tree/structure of method calls
- Things to consider in your drawing and calculations of total work:
  - Height of tree: how many levels will it take for you to reach the base case?
  - Branching factor: how many times does the function call itself in the body of the function?
  - Work per node: how much actual work is done per function call?
- Life hack pattern matching when calculating total work where f(N) is some function of N
  - $1 + 2 + 3 + 4 + 5 + \ldots + f(N) = [f(N)]^2$
  - $1 + 2 + 4 + 8 + 16 + \ldots + f(N) = f(N)$
    - Rule applies with any geometric factor between terms, like $1 + 3 + 9 + \ldots + f(N)$

Asymptotics Advice

- Doing problems graphically can be helpful if you're a visual learner (plot variable values and calculate area formula):

```
for (int i = 0; i < N; i++) {
     for (int j = 0; j < i; j++) {
          /* Something constant */
     }
}
```



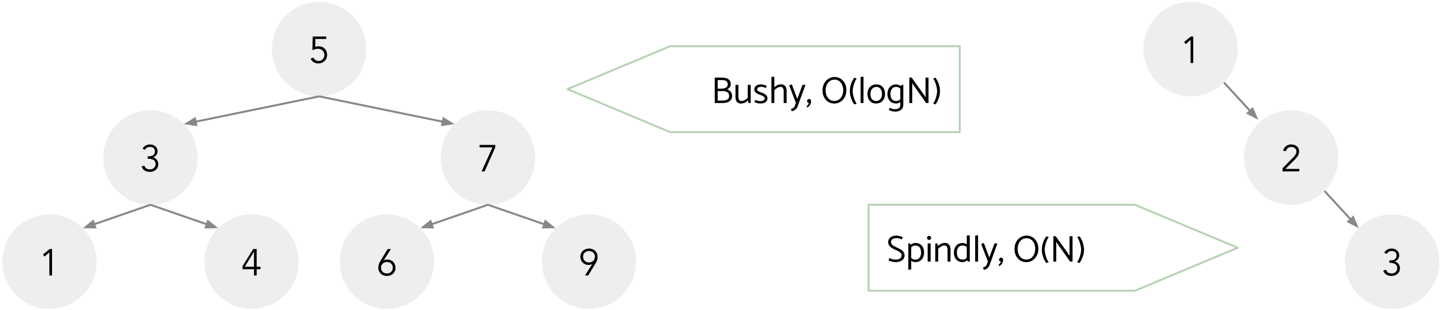
$\frac{1}{2} N^2 = N^2$

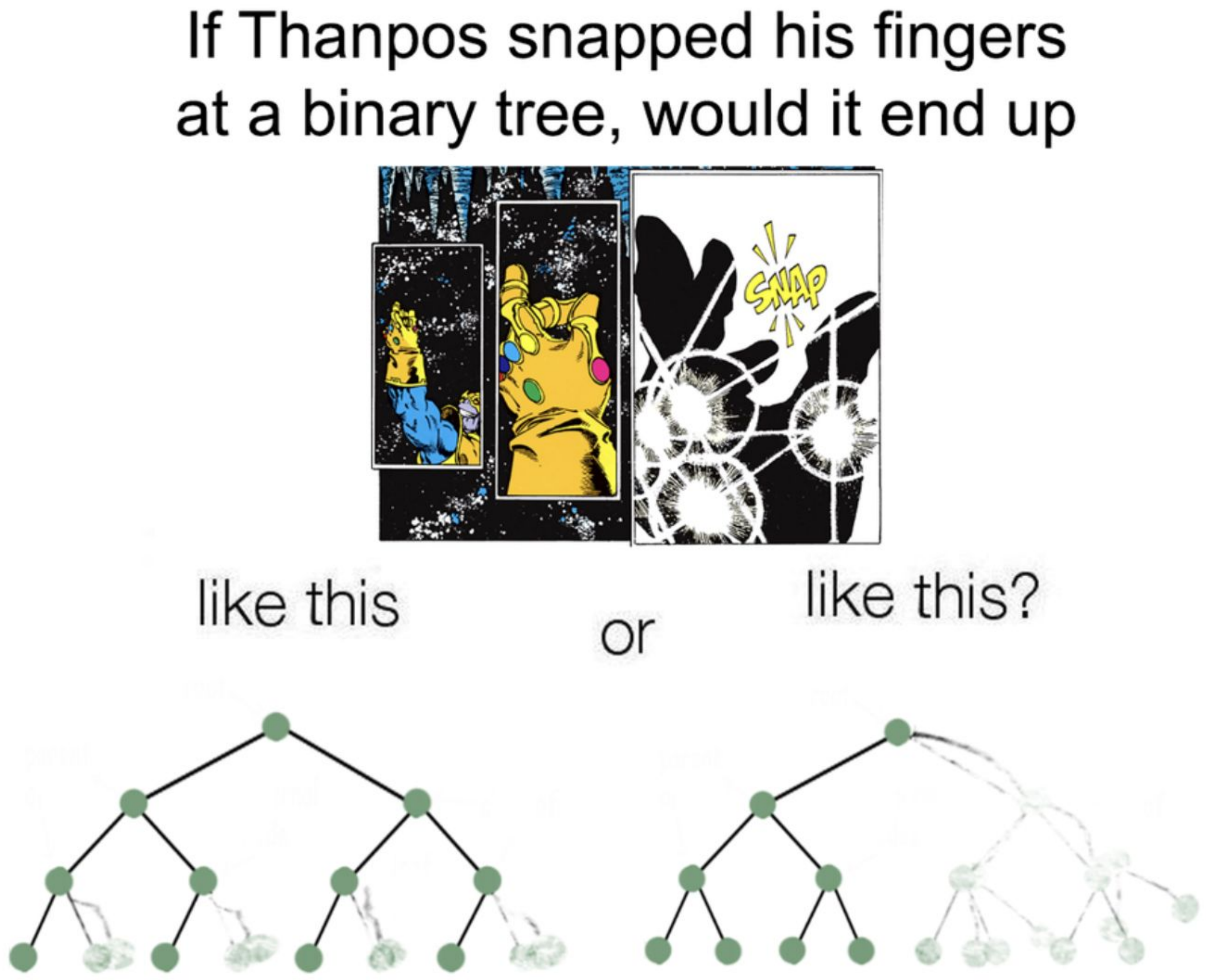
Binary Search Trees

Binary Search Trees are data structures that allow us to quickly access elements in sorted order. They have several important properties:

1. Each node in a BST is a root of a smaller BST
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

BSTs can be bushy or spindly:

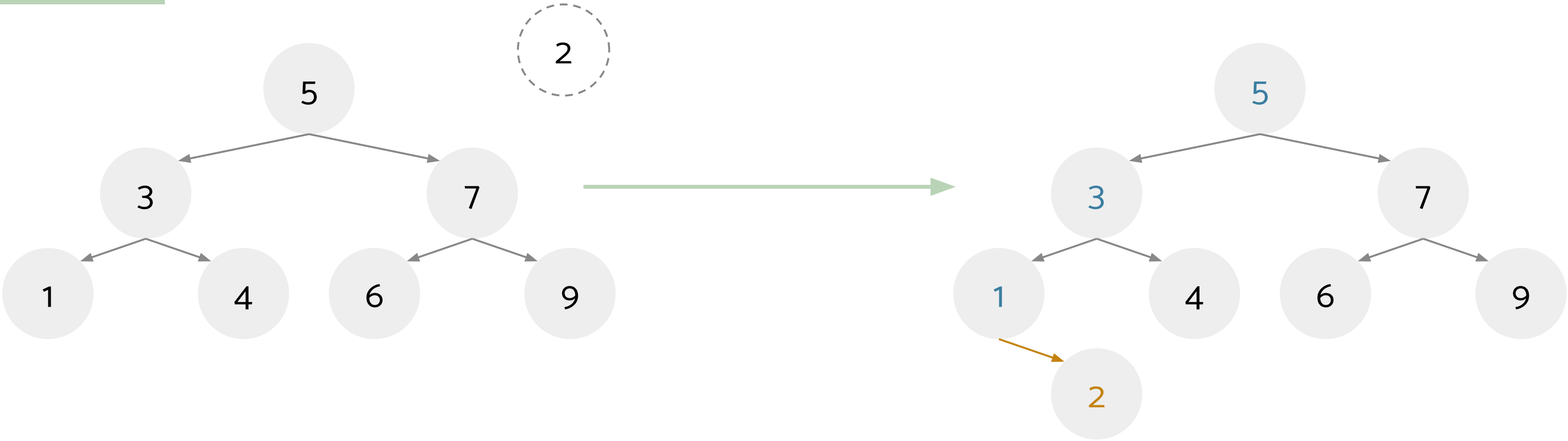



If Thanpos snapped his fingers
at a binary tree, would it end up
SNAP
like this
or
like this?

BST Insertion

Items in a BST are always inserted as leaves.

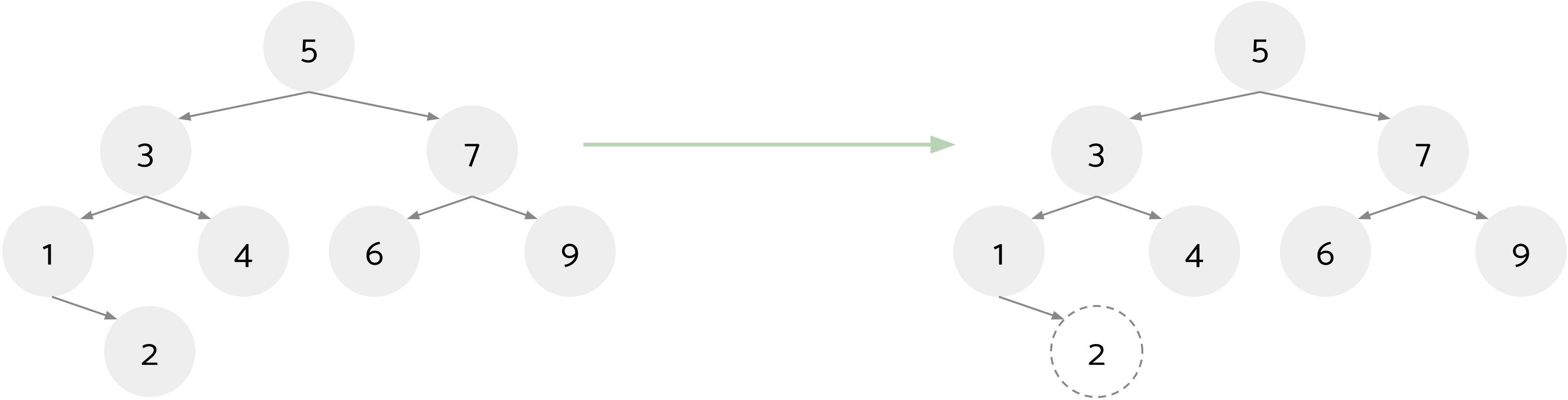
`insert(2)`



BST Deletion

Items in a BST are always deleted via a method called Hibbard Deletion. There are several cases to consider:

`delete(2)`

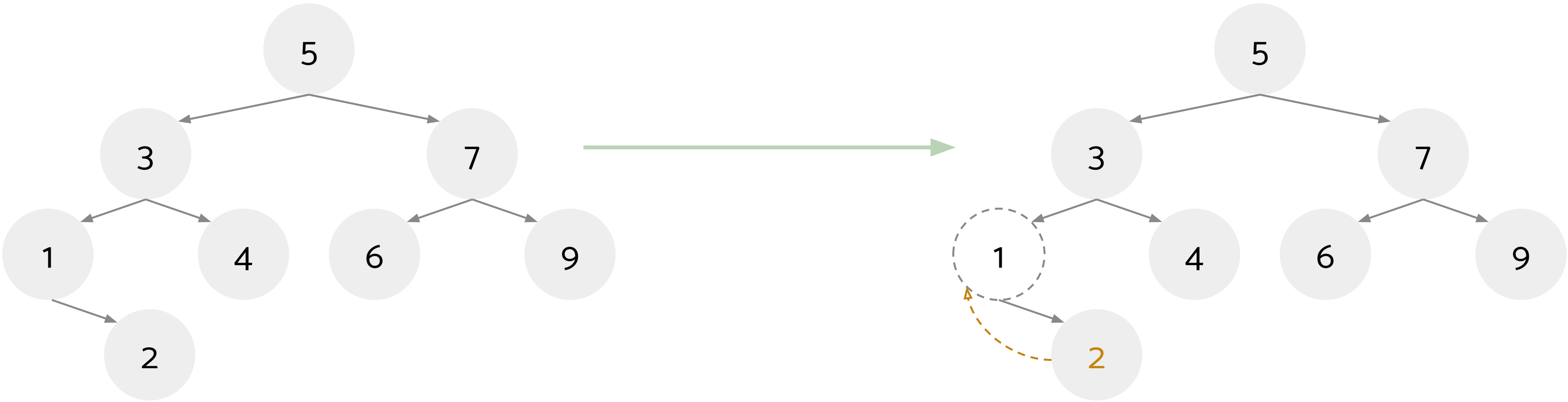


In this case, the node has no children so deletion is an easy process.

BST Deletion

Items in a BST are always deleted via a method called Hibbard Deletion. There are several cases to consider:

`delete(1)`

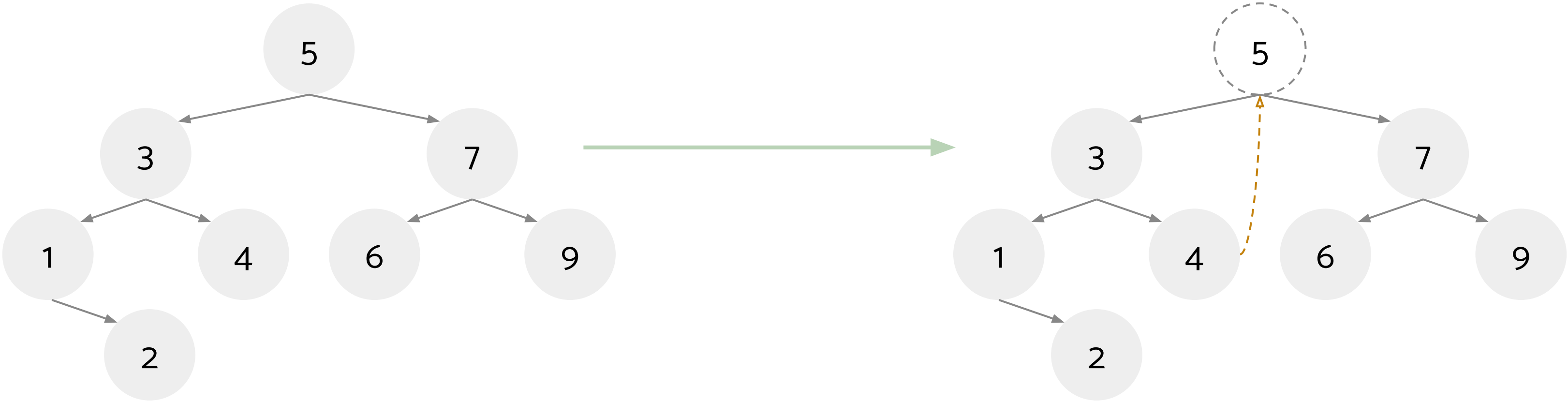


In this case, the node has one child, so it simply replaces the deleted node, and then we act as if the child was deleted in a recursive pattern until we hit a leaf.

BST Deletion

Items in a BST are always deleted via a method called Hibbard Deletion. There are several cases to consider:

`delete(5)`



In this case, the node has two children, so we pick either the leftmost node on in the right subtree or the rightmost node in the left subtree.

Worksheet

1 Finish the Runtimes

For each part, a desired runtime is 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.

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

1 Finish the Runtimes

For each part, a desired runtime is 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.

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

1 Finish the Runtimes

For each part, a desired runtime is 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.

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

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

1 Finish the Runtimes

For each part, a desired runtime is 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.

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

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

$1 + 2 + 3 + \ldots + N = \Theta(N^2)$

1 Finish the Runtimes

For each part, a desired runtime is 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.

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

```
for (int i = 1; i < N; i = i * 2) {
    for (int j = 1; j < ______; j = j * 2) {
        System.out.println("This is one is mid key hard");
    }
}
```

1 Finish the Runtimes

For each part, a desired runtime is 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.

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

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

i = 1, 2, 4, … N → log N iterations

1 Finish the Runtimes

For each part, a desired runtime is 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.

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

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

1 Finish the Runtimes

For each part, a desired runtime is 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.

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

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

$1 + 2 + 4 + \ldots 2^N = \Theta(2^N)$

1 Finish the Runtimes

For each part, a desired runtime is 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.

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

```
for (int i = 1; i < ______; i = i * 2) {
    for (int j = 1; j < N * N; j = ______) {
        System.out.println("yikes");
    }
}
```

1 Finish the Runtimes

For each part, a desired runtime is 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.

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 = j + 1) {
        System.out.println("yikes");
    }
}
```

Outer loop: $i = 1, 2, 4, \ldots, 2^N \rightarrow N$ iterations

Inner loop: N^2

2A Asymptotics is Fun!

```
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( )

2A Asymptotics is Fun!

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

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

g(N, 2): $\Theta(N^2)$

2B Asymptotics is Fun!

```
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( )

2B Asymptotics is Fun!

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

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

3A Is This a BST?

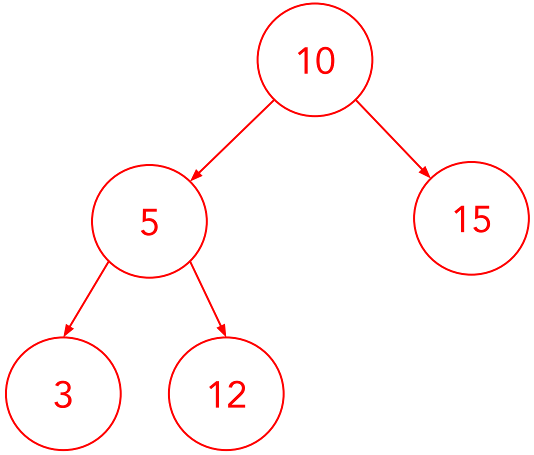
The following code should check if a given binary tree is a BST. However, for some trees, it returns the wrong answer. Give an example of a binary tree for which `brokenIsBST` fails.

```
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);
     }
}
```

3A Is This a BST?

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);
     }
}
```



3B Is This a BST?

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

```
public static boolean isBST(BST T) {

}

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





}
```

3B Is This a BST?

```
public static boolean isBST(BST T) {
    return isBSTHelper(T, Integer.MIN_VALUE, Integer.MAX_VALUE);
}

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);
    }
}
```

CS 61B — Fall 2023

Asymptotics, Disjoint Sets

Exam-Level 7: September 25, 2023

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

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

this is log already, pick a constant

```
for (int i = 1; i < N; i = i * 2) {
    for (int j = 1; j < __2___; j = j * 2) {
        System.out.println("This is one is mid key hard");
    }
}
```

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

Math.pow(2, i)

```
for (int i = 1; i < N; i = i + 1) {
    for (int j = 1; j < __2^i___; j = j + 1) {
        System.out.println("This is one is high key hard");
    }
}
```

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

Math.pow(2, N)

N

```
for (int i = 1; i < __2^N___; i = i * 2) {
    for (int j = 1; j < N * N; j = _j+1___) {
        System.out.println("yikes");
    }
}
```

N²

Independent from outer loop

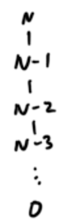
2 Asymptotics is Fun!

(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 }
```

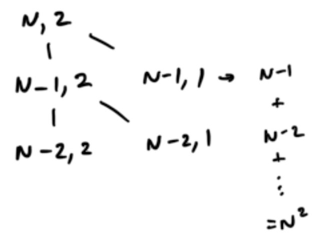
`g(N, 1)`: $\Theta($ N $)$

1 iteration of inner loop



`g(N, 2)`: $\Theta($ N^2 $)$

2 iterations of inner loop



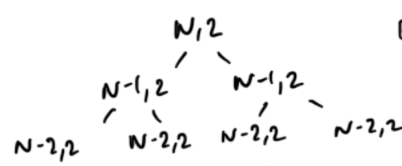
(b) Suppose we change line 6 to `g(N - 1, x)` and change the stopping condition in the for loop to `i <= f(x)` where `f` returns a random number between 1 and `x`, inclusive. For the following function calls, find the tightest Ω and big O bounds. Feel free to draw out the recursion tree if it helps.

f(x) determines Ω and O
↓ ↓
1 x

```
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 $)$

N,2
N-1, 2
N-2, 2
⋮
N times



Branching Factor: x (2 here)

O(1) work/node

\# of nodes: $1+2+4+\ldots+2^N = \frac{1(1-2^{N+1})}{1-2} = 2^{N+1}-1 = O(2^N)$

Starts at 2^0
Ends at 2^N
N+1 terms

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

↳ Branching Factor: N

\# of nodes: $1+N+N^2+\ldots+N^N = \frac{1(1-N^{N+1})}{1-N} = N^N + \ldots = O(N^N)$

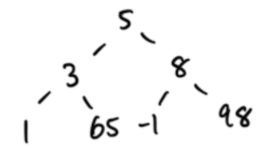
N,N
N-1,N
N-2,N
⋮
N times

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.

```java
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);
    }
}
```



Not recursively checking all values

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

Hint: You will find `Integer.MIN_VALUE` and `Integer.MAX_VALUE` helpful. ← Start here

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?

```java
public static boolean isBST(BST T) {
    return isBSTHelper(isBSTHelper(T, Integer.MIN_VALUE, Integer.MAX_VALUE);
}

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);

    }
}
```

Same base case idea
↳ else if checks validity, else does recursion

update max

update min