CS 61BL Lab 7

Ryan Purpura

Agenda for today: Asymptotics and Order of Growth

- What we'll be doing:
  - Formalize how we measure program performance
  - Be able to *predict* how programs will scale with bigger and bigger input

Finding Index of Item

- Consider a simple function that finds the maximum value of an integer array

```
public static int findItemIndex(int[] array, int item) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == item) {
                return i;
            }
        }
        return -1;
}
```

Runtime

```
public static boolean hasDuplicates(int[] array) {
    for (int i = 0; i < array.length; i++) {
        for (int j = i + 1; j < array.length; j++) {
            if (array[i] == array[j]) {
                return true;
            }
        }
    }
    return false;
}
```

- Assume array is length ***N***.
- In the **worst case**, how many comparisons (calls to `array[i] == array[j]`) do we need find the answer? What array gives us the worst case?
- What about the **best case**? (# of comparisons and which arrays)

```java
public static boolean hasDuplicates(int[] array) {
    for (int i = 0; i < array.length; i++) {
        for (int j = i + 1; j < array.length; j++) {
            if (array[i] == array[j]) {
                return true;
            }
        }
    }
    return false;
}
```

value of ***i***	range of ***j***	# Comparisons
i = 0	[1, ***N***]	(***N*** - 1)
i = 1	[2, ***N***]	(***N*** - 2)
i = 2	[3, ***N***]	(***N*** - 3)
i = *N* - 1	[***N*** - 1, ***N***]	1
i = *N*	N/A	0

$$0 + 1 + 2 + 3 + \ldots + (N - 1) = \sum_{i=0}^{N-1} i = \frac{N(N-1)}{2}$$

Simplifying

- Drop multiplicative constants and lower-order terms
- So $\frac{N(N-1)}{2} \rightarrow N^2$

- Justification: as ***N*** gets bigger and bigger, those two functions have an order of growth that differs only by a constant factor:

$$\lim_{N \to \infty} \frac{N^2}{N(N-1)/2} = 2$$

Announcements

- Project 1 due tonight at 11:59 pm
- Project Party at 3:30-7:30pm today in the Wozniak Lounge
- Midterm 1 on Monday! Monday's lab will be an open office hours/review session

Big Theta

- $\Theta\left(f(N)\right)$ is the *family* of functions that grows as fast as $f(N)$.

- For example, we can say that $\frac{N(N-1)}{2}$ is in the family of functions $\Theta\left(N^2\right)$

- Or more succinctly, $\frac{N(N-1)}{2} \in \Theta\left(N^2\right)$

Big O

- $O\left(f(N)\right)$ is the *family* of functions that grows as fast as ***or slower than*** $f(N)$.

- $N \notin \Theta\left(N^2\right)$ but $N \in O\left(N^2\right)$ since N grows slower than N^2

Big Omega

- $\boldsymbol{\Omega}\left(f(N)\right)$ is the *family* of functions that grows as fast as ***or faster than*** $f(N)$.

- $N^2 \notin \boldsymbol{\Theta}\left(N\right)$ but $N^2 \in \boldsymbol{\Omega}\left(N\right)$ since N grows slower than N^2

Bringing it all together: Figuring out order of growth

- Pick a cost model: typically how many times some line(s) of code is run

- Use a counting technique to come up with a function in terms of N (the input size)

- Simplify by dropping lower-order terms and multiplicative constants.

- Determine the most appropriate bound (Theta, O, or Omega)

Find the Asymptotic Runtime.

```
public static int findItemIndex(int[] array, int item) {
    for (int i = 0; i < array.length; i++) {
        if (array[i] == item) {
            return i;
        }
    }
    return -1;
}
```

N is the length of array.

In the best case, we need only one comparison. In the worst case, we need ***N*** comparisons.

Runtime is $O(N)$

Find the Worst-Case Runtime.

```
public static int findItemIndex(int[] array, int item) {
    for (int i = 0; i < array.length; i++) {
        if (array[i] == item) {
            return i;
        }
    }
    return -1;
}
```

N is the length of array.

In the worst case, we need N comparisons.

Worst-Case Runtime is $\Theta(N)$

Why is $O(N)$ not the best answer?

Find the Asymptotic Runtime.

```
public static void printer(int N) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < Math.log(N); j++) {
            System.out.println("Hello.");
        }
    }
}
```

The inner loop takes $\log N$ amount of work

The inner loop gets run in its entirety N times.

In total, we have $\Theta(N \log N)$

Recursion and You

- The usual strategy when finding the runtime of recursive functions is to draw a tree diagram that represents the recursive calls

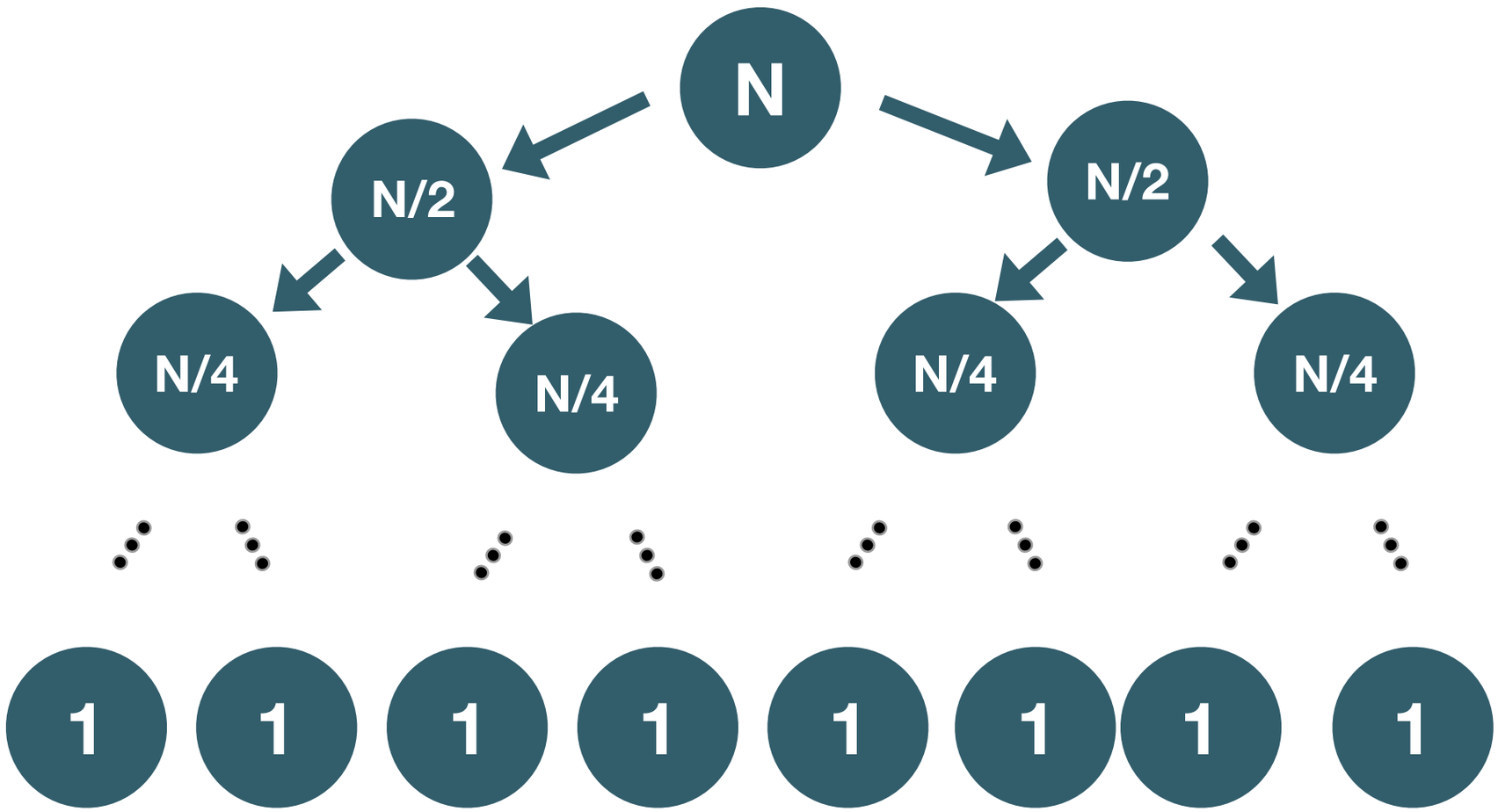
- Each node in the tree represents a function call to the recursive function.

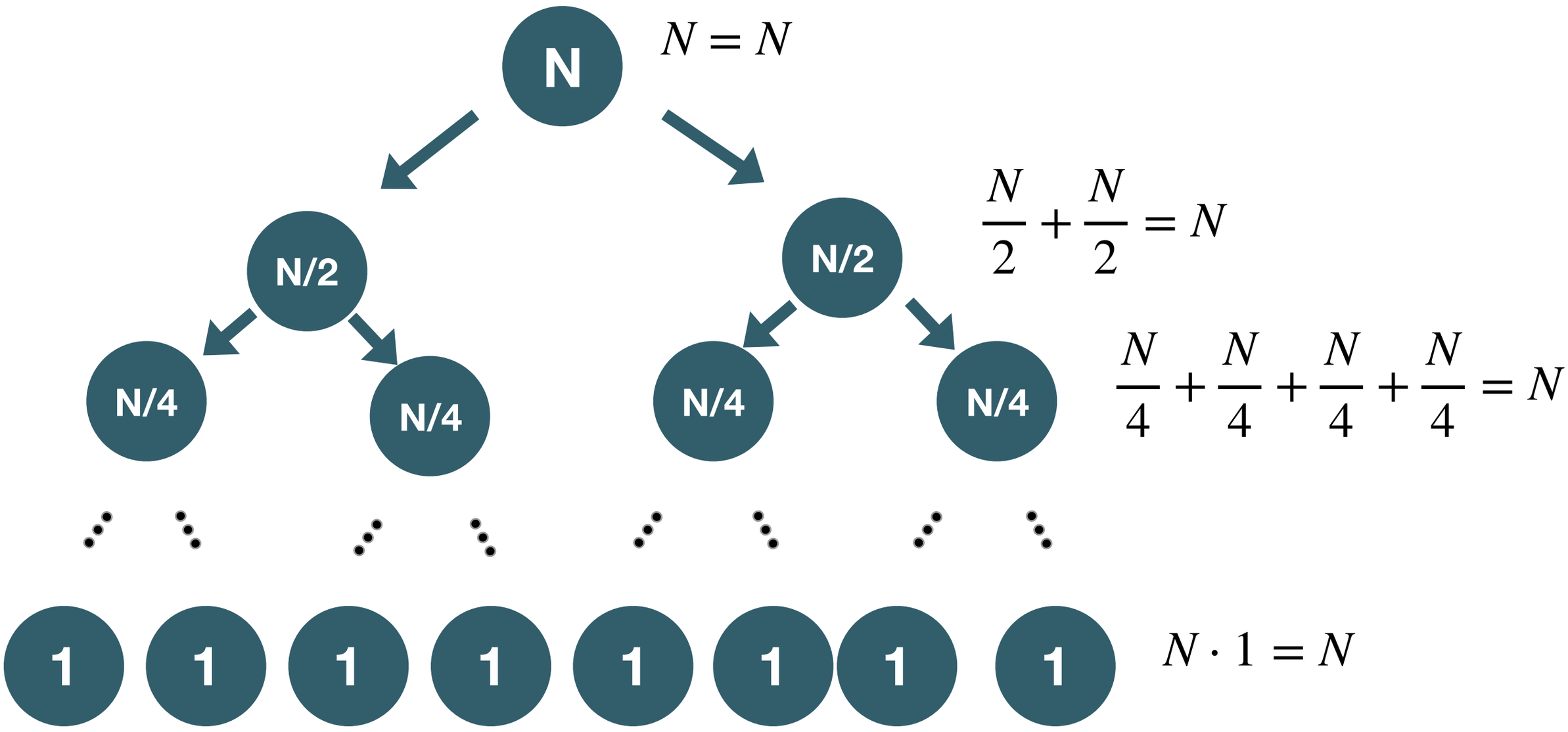
- Figure out how much work is done at each node in terms of *N*, add them up

Runtime

```
public void andSlam(int N) {
    if (N > 1) {
        for (int i = 0; i < N; i += 1) {
            System.out.println("datboi.jpg");
        }
        andSlam(N / 2);
        andSlam(N / 2);
    }
}
```

```java
public void andSlam(int N) {
    if (N > 1) {
        for (int i = 0; i < N; i += 1) {
            System.out.println("datboi.jpg");
        }
        andSlam(N / 2);
        andSlam(N / 2);
    }
}
```





$N = N$

$\frac{N}{2} + \frac{N}{2} = N$

$\frac{N}{4} + \frac{N}{4} + \frac{N}{4} + \frac{N}{4} = N$

$N \cdot 1 = N$

Total runtime: $\Theta\left(N \log(N)\right)$