CMPT 225: Data Structures & Programming – Unit 06 – Recursion

Dr. Jack Thomas

Simon Fraser University

Spring 2021

Today’s Topics

- What is Recursion?
- Linear Recursion
- Recursion Tracing
- Binary & Multiple Recursion

What if a Function Calls Itself?

- Functions can call themselves, which causes the current **instance** to pause until the new instance finishes running.
- If you’re not careful, this could cause an **infinite loop**.
- With a little planning, this can be an efficient and effective programming tool.
- In algorithmic terms, this is called **Recursion**.

Recursion

- Recursion is on the algorithmic/programming side of this course's content.
- Along with more explicit loops like for and while, it's a **tool for repetition**.

```
class ArraySizeCounterClass {
    public static int ArraySizeCounter(String[] a)
    {
        if (a.length == 1)
        {
            return 1;
        }
        else
        {
            String[] b = java.util.Arrays.copyOf(a, newLength: a.length - 1);
            int counter = ArraySizeCounter(b) + 1;
            return counter;
        }
    }
}
```

```
String[] countThese = {"Count", "these", "four", "Strings."};
int result = ArraySizeCounterClass.ArraySizeCounter(countThese);
System.out.println("Array count: " + result);
```

```
Array count: 4
```

More Useful Recursive Example: The Fibonacci Sequence

- 0, 1, 1, 2, 3, 5, 8…
- F[i] = F[i-1] + F[i-2]
- Each value in the Fibonacci sequence is the sum of the two previous values.
- If I asked you what the 23rd Fibonacci value… wait

Image credit: https://en.wikipedia.org/wiki/Fibonacci#/media/File:Leonardo_da_Pisa.jpg

Hold Up, Peep That Chill Dude

- Is he wearing a skull mask?
- Sick

Image credit: https://en.wikipedia.org/wiki/Fibonacci#/media/File:Leonardo_da_Pisa.jpg

Okay, Back On Topic

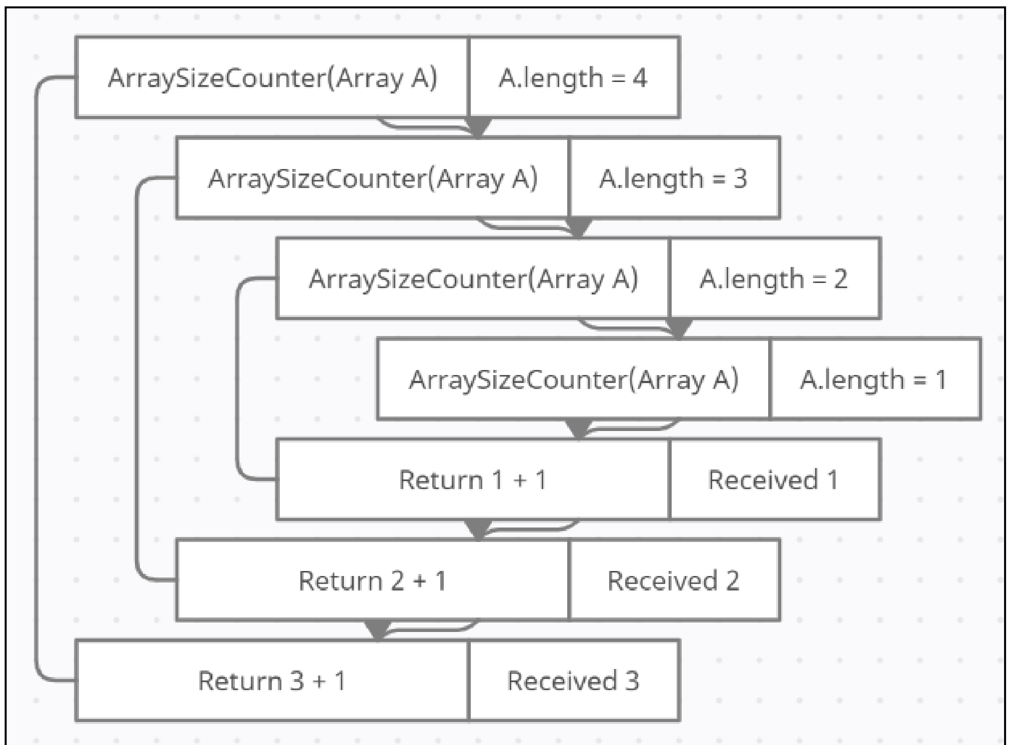
- If I asked you what the 23rd Fibonacci Sequence value is, you'd need to find the 22nd and 21st. The 22nd would require the 21st and 20th, while the 21st would require the 20th and 19th…
- Each of these steps is a repetition of the question "What is the Fibonacci Sequence value for x?".
- We could structure our solution to take advantage of this pattern.

Anatomy of a Recursive Function

- Each run of a recursive function goes one of two ways:
  - **A Recursive Call**: The condition where the function will call itself. Typically has to change something between calls, like maybe calling itself on a subset (Ex: Asking for the values of the two previous Fibonacci Sequence terms, instead of asking for itself again).
  - **The Base Case:** The condition where a recursive function does *not* call itself again, typically when it starts working its way back up to the solution (Ex: The first two Fibonacci Sequence values are known to be 0 and 1).

Linear Recursion

- The simplest form – a method only makes one recursive call each time it's called, down to the base case.



Linear Example: Summing an Array

- Say we wanted to add up an array A of n integers recursively:

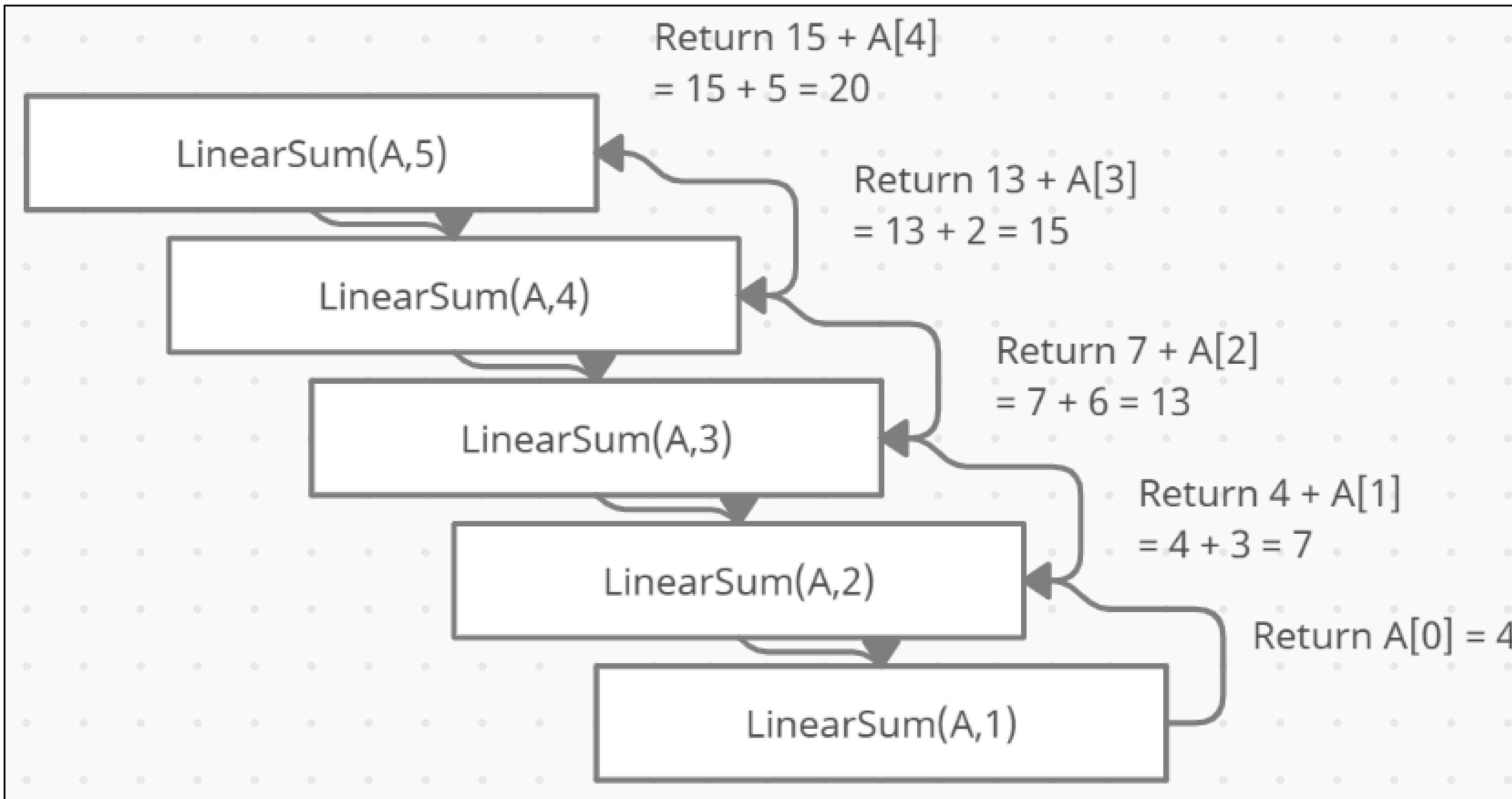
```
Algorithm LinearSum(A,n):
    Input: An integer array A and an integer
    n >= 1 such that A has at least n elements
    Output: The sum of the first n integers in A
    if n = 1 then
        return A[0]
    else
        return LinearSum(A, n – 1) + A[n-1]
```

Recursion Tracing

- How we **visualize** what a recursive function will actually do across its multiple instances.
- Draw a box for each instance, label it with the parameters of the method.
- Then draw an arrow from each calling method to their called method.
- Once you reach the base case, start tracing the results back "up" to the first call.

Recursion Trace for LinearSum

- Let’s give LinearSum(A,n) an array A of {4,3,6,2,5} and n of 5.



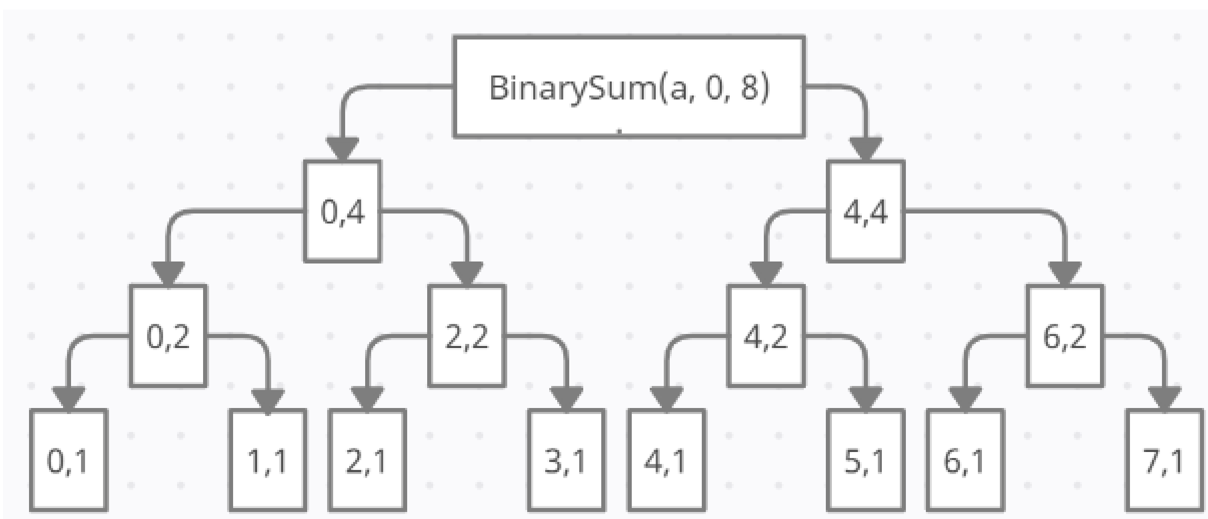
Binary Recursion: Call Twice

- Exactly what it sounds like – where Linear had one recursive call, Binary has two.

```
Algorithm BinarySum (A, i, n):
        Input: an array A and integers i and n
        Output: the sum of the n integers in A starting at index I
        if n = 1 then
                return A[i]
        return BinarSum(A,I,(n/2)) + BinarySum(A,i+(n/2),(n/2))
```

Tracing BinarySum

- Say we gave BinarySum(a, i, n) an eight-number array a, with i = 0 (start summing from the first index) and n = 8 (the length of the array).



“Aha,” you say. “So this is how we do the Fibonacci Sequence!”

- Surprisingly, no.
- It’s not hard to imagine a binary recursive algorithm that can do it – basically just “BinaryFib(n-1) + BinaryFib(n-2)” with a base case where if n <= 1, return n.
- The problem is, this leads to an **exponential number of function calls** – every Fibonacci Sequence value gets calculated multiple times over the run of the program.

Linear Fibonacci

- Let's tweak our approach to calculate both the value we want and the value before it at the same time.

```
Algorithm LinearFibonacci(k):
        Input: A nonnegative integer k
        Output: Pair of Fibonacci values (Fk, Fk-1)
        if k <= 1 then
                return (k,0)
        else
                (i, j) <- LinearFibonacci(k - 1)
                return (i+j, i)
```

Image credit: https://www.sciencesource.com/archive/Leonardo-Fibonacci--Italian-Mathematician-SS2192062.html

By The Way Someone Painted The Face Of That Fibonacci Statue From Earlier And Yeeesh

- Kinda looks like Mark Zuckerberg?
  - Maybe it’s the hoodie…

Image credit: https://www.sciencesource.com/archive/Leonardo-Fibonacci--Italian-Mathematician-SS2192062.html

How About More Than Twice?

- **Multiple Recursion** can be generalized from Binary Recursion pretty directly.
- Useful for solving **complicated combination or permutation puzzles**, where you want to test many configurations.
- We should probably make sure we can walk before we try running, though – no need to start trying this one out just yet.

Tips for Designing Recursive Functions

- Think of ways you can **subdivide** your problem into smaller problems with the same general structure – halving an array, for example.
- You may need to **redefine the question** you're asking to achieve this, like asking for two different Fibonacci values instead of just the one you want.
- It might help to **work up from the base cases** – if the problem is easily solved when n = 0 or 1 (for whatever that means in your problem), try starting with them!

Recursion, Loops, Arrays, Lists…

- It’s starting to look like we have **a number of different options** for data structures and algorithms to design our systems with.
- Some of these have **different properties** that can make them **more or less efficient**, like recursion leading to exponential function calls or lists being slow to traverse.
- How can we analyze these differences to make good design decisions? **Tune in next week…**

Recap – Now Go Read These Slides in Reverse

- **Recursion** is a technique for repetition whereby a function calls itself.
- Recursive functions typically include conditions that trigger a **recursive call** and conditions that begin the process of resolving the function, called the **base case(s)**.
- **Recursion Tracing** lets us visualize what's going on in a recursive function.
- In **Linear** recursion, each instance of the recursive function will only call itself one time, while **Binary** recursion calls itself twice, and **Multiple** recursion can call itself many times.