Mini Lecture on Recursion

CS7646 – Georgia Tech
James Chan

9/15/17

What is Recursion?

- A function that repeatedly calls itself.
- Perform repetitive tasks similar to loops.
- Applicable when the main problem is made up of smaller versions of itself.

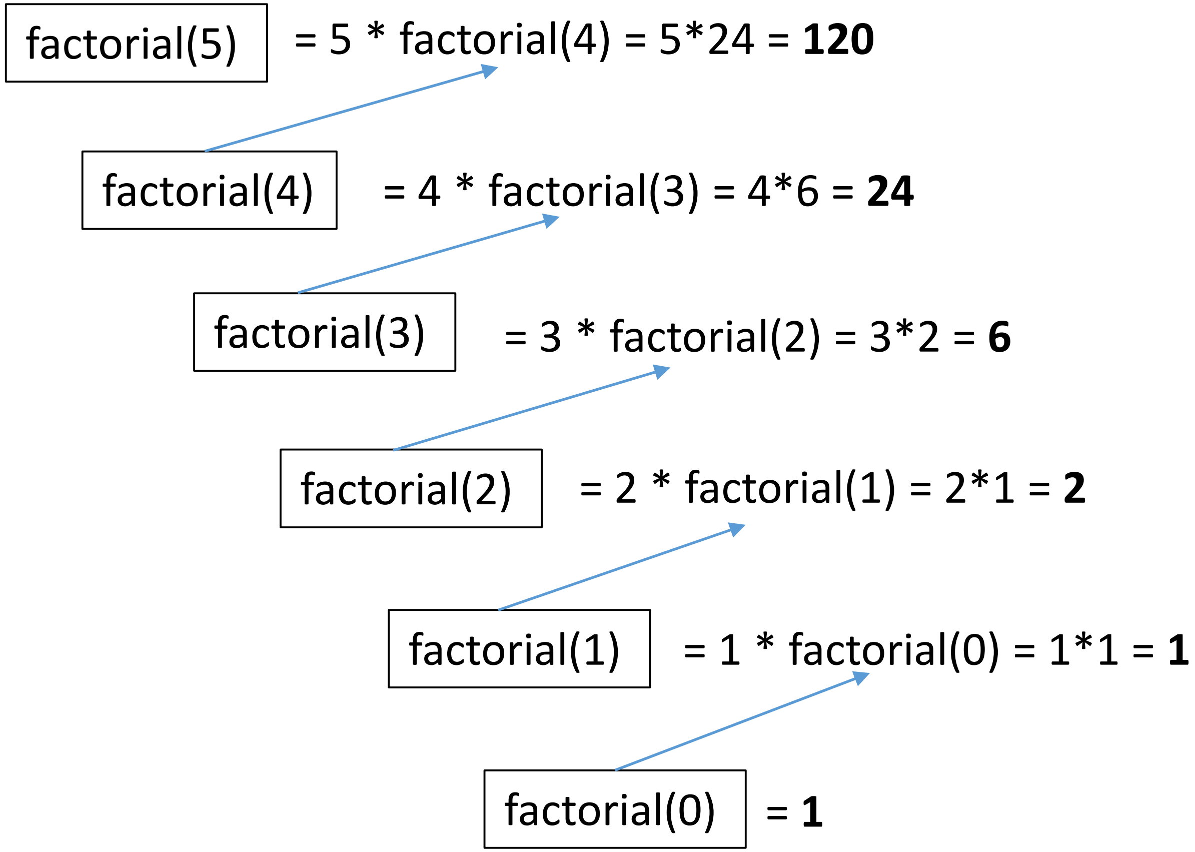
A Cliché Example: Factorial

- !N = 1 x 2 x 3 x 4 … N
- Whereby each element is multiplied by the product of all the preceding elements.
- !N **=** N * !(N-1)
- Examples:
  - !2 = 1 x 2 = **2**
  - !5 = 1 x 2 x 3 x 4 x 5 = **120**

A Cliché Example: Factorial

- Python code:

```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n-1)
```



factorial(5) = 5 * factorial(4) = 5*24 = **120**

factorial(4) = 4 * factorial(3) = 4*6 = **24**

factorial(3) = 3 * factorial(2) = 3*2 = **6**

factorial(2) = 2 * factorial(1) = 2*1 = **2**

factorial(1) = 1 * factorial(0) = 1*1 = **1**

factorial(0) = **1**

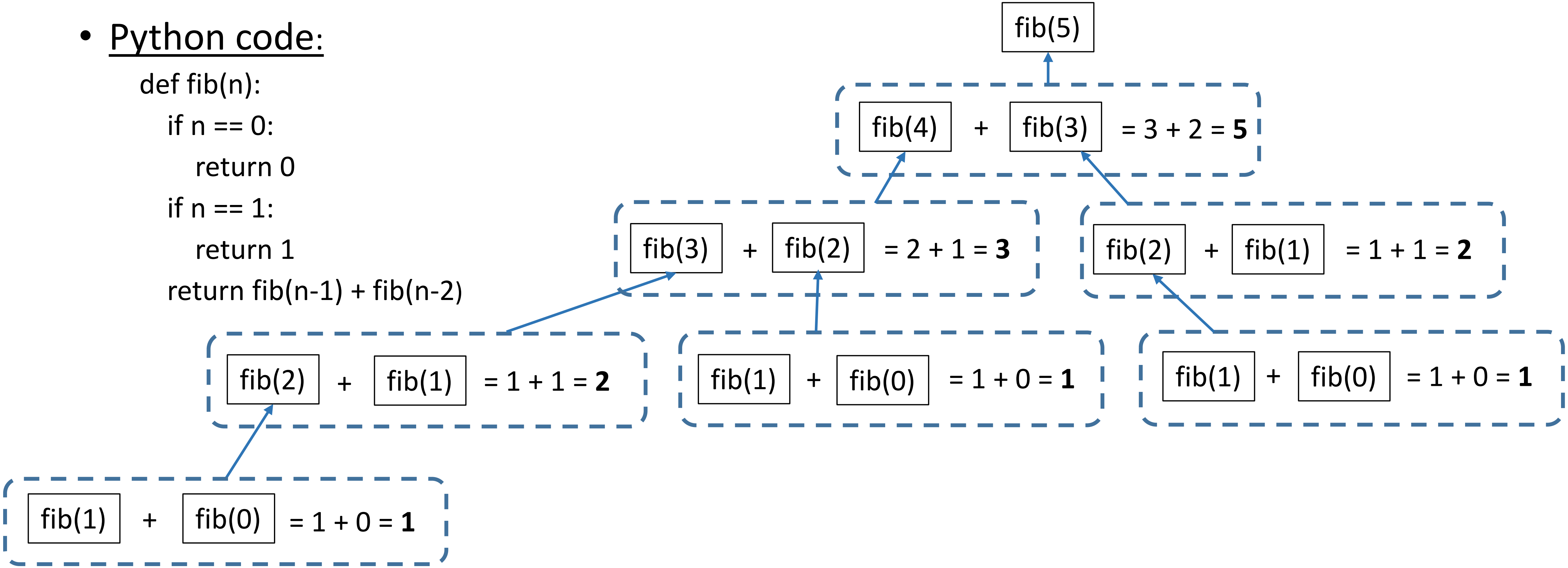
Another Cliché Example: Fibonacci Sequence

- 0,1,1,2,3,5,7,13,20…etc
- Whereby each element is the sum of the previous two elements.
- Fib(N) = Fib(N-1) + Fib(N-2)
- Poor efficiency, demo only. O(2^N) running time!?

Another Cliché Example: Fibonacci Sequence

- Python code:

```
def fib(n):
   if n == 0:
      return 0
   if n == 1:
      return 1
   return fib(n-1) + fib(n-2)
```



fib(5)

fib(4) + fib(3) = 3 + 2 = **5**

fib(3) + fib(2) = 2 + 1 = **3**

fib(2) + fib(1) = 1 + 1 = **2**

fib(2) + fib(1) = 1 + 1 = **2**

fib(1) + fib(0) = 1 + 0 = **1**

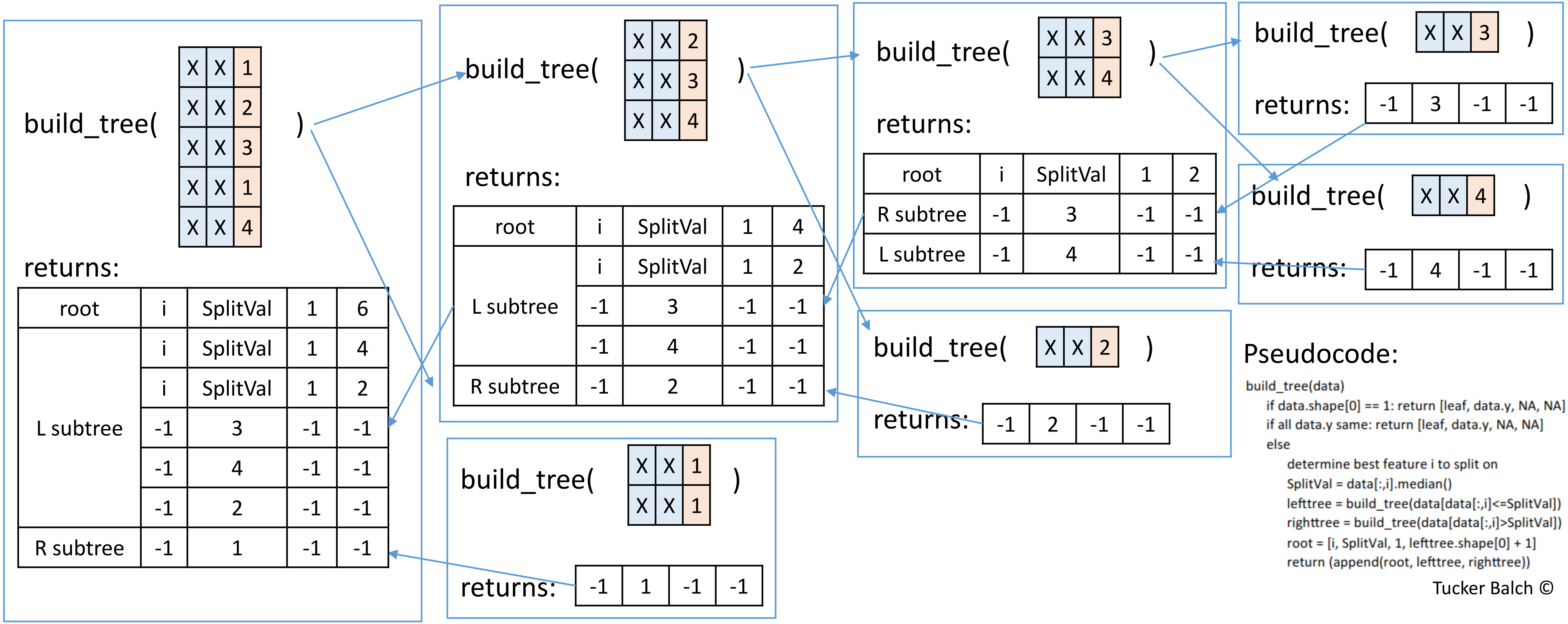
fib(1) + fib(0) = 1 + 0 = **1**

fib(1) + fib(0) = 1 + 0 = **1**

Some Famous Recursive Algorithms

- MergeSort, QuickSort
- Tree Traversals
- Binary Tree Construction!

Decision Tree Construction



build_tree(

X	X	1
X	X	2
X	X	3
X	X	1
X	X	4

)

returns:

root	i	SplitVal	1	6
L subtree	i	SplitVal	1	4
	i	SplitVal	1	2
	-1	3	-1	-1
	-1	4	-1	-1
	-1	2	-1	-1
R subtree	-1	1	-1	-1

build_tree(

X	X	2
X	X	3
X	X	4

)

returns:

root	i	SplitVal	1	4
L subtree	i	SplitVal	1	2
	-1	3	-1	-1
	-1	4	-1	-1
R subtree	-1	2	-1	-1

build_tree(

X	X	1
X	X	1

)

returns:

-1	1	-1	-1

build_tree(

X	X	3
X	X	4

)

returns:

root	i	SplitVal	1	2
R subtree	-1	3	-1	-1
L subtree	-1	4	-1	-1

build_tree(

X	X	3

)

returns:

-1	3	-1	-1

build_tree(

X	X	4

)

returns:

-1	4	-1	-1

build_tree(

X	X	2

)

returns:

-1	2	-1	-1

Pseudocode:

```
build_tree(data)
    if data.shape[0] == 1: return [leaf, data.y, NA, NA]
    if all data.y same: return [leaf, data.y, NA, NA]
    else
        determine best feature i to split on
        SplitVal = data[:,i].median()
        lefttree = build_tree(data[data[:,i]<=SplitVal])
        righttree = build_tree(data[data[:,i]>SplitVal])
        root = [i, SplitVal, 1, lefttree.shape[0] + 1]
        return (append(root, lefttree, righttree))
```