UNIVERSITY OF CALIFORNIA, BERKELEY
Department of Electrical Engineering and Computer Sciences
Computer Science Division

**CS10 Fall 2024** **TA: Dorottya Urmossy**


**Discussion-7: Tree Recursion & Fractals**

# SOLUTIONS

**Instructions**

- If you're attending this section in-person, at the end of section, you'll be given a secret phrase. Please enter it here: https://tinyurl.com/fa24-attend
- To get attendance credit please submit your worksheet on Gradescope.
- Please open up https://snap.berkeley.edu/snap/snap.html on your computer.
- Ask the person to your right / left about their opinion on cheese.

## Q2. Tree-Recursion: Lingua Franca

**A.** What is *tree recursion*? What does it mean for a procedure to be defined *tree recursively*? What is the other form of recursion we've studied?

Tree-recursion is the process that occurs when a procedure makes more than one recursive call to itself in a recursive case. Tree-recursive procedures utilize tree-recursion. The other form of recursion, wherein a process calls itself at most once in all recursive cases, is called linear-recursion.

**B.** Given the choice between a tree-recursive implementation and a linear-recursive implementation for a given problem, which one should you choose? Why?

Tree-recursive procedures, in general, make many more recursive calls than linear-recursive procedures, for inputs of similar size. This means that your computer uses a lot more memory for tree-recursive procedures (it needs to track data across many more function calls.) Try calling fibonacci with N = 100, and see what happens. Then call factorial with N = 100 (it should compute instantly!)

**D.** Classify the following problems based on the recursive approach used to solve them in lecture: Factorial, Fibonacci, Count Change, Fractal.

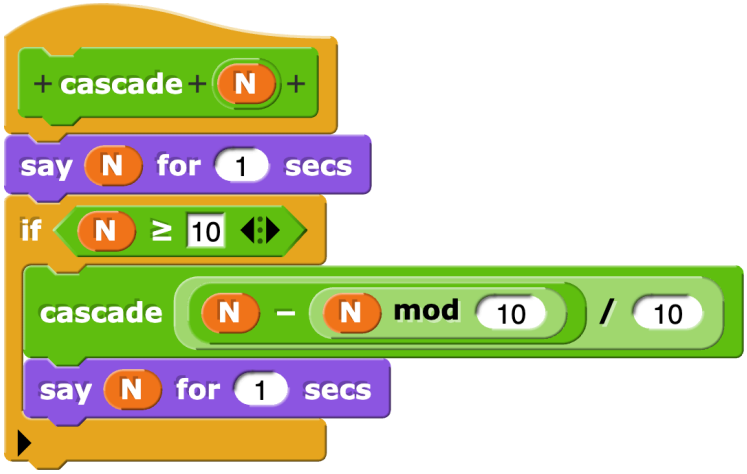
Tree-recursive: Fibonacci, Count-Change, Fractal.
Linear-recursive: Factorial

**E.** Suppose I had a non-negative integer N and a list of digits D. I'd like to find out how many distinct summations involving >= 1 digits in D sum up to N. For example, if N = 3 and digits = [1, 2, 3] then the result is 3, with the summations: 1 + 1 + 1, 2 + 1, and 3. Can we write a linearly-recursive procedure to solve this problem?

This is a thinly-veiled version of the count-change problem, which *cannot* be solved using just one recursive call (do you see why?) Think of the digits as the denominations of coins, and N as the amount you're making change for.

Generally, count-change and this "distinct summation" problem are part of a larger class of "partitioning" problems — wherein you're exploring the partitions of a whole unit into various subunits under different constraints.

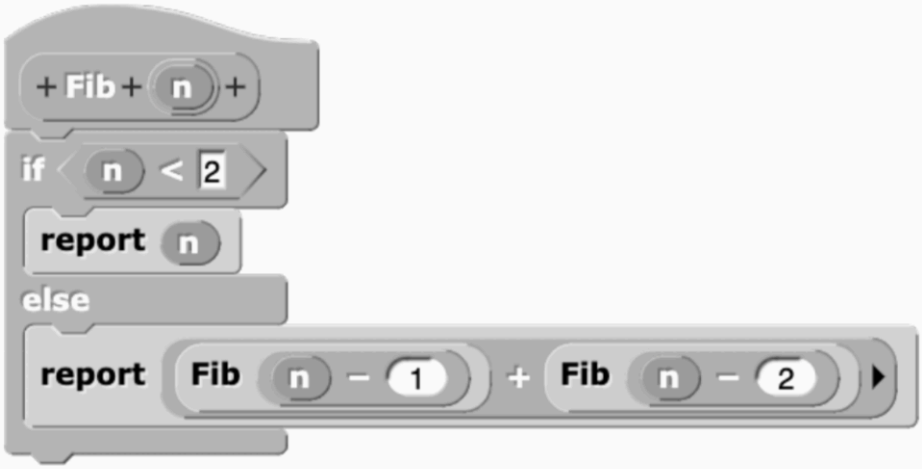
## Q3. Analyzing Recursive Procedures




What's *said* when we call cascade with N = 123456?

123456
12345
1234
123
12
1
12
123
1234
12345
123456

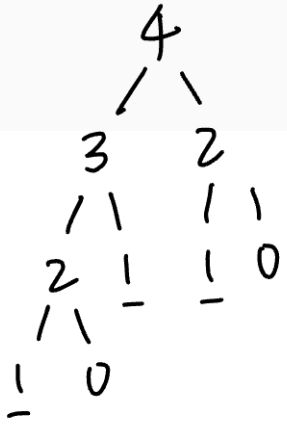
*So pretty!*




After running the script above, we run the following block: Fib 4

**A.** In the execution of the Fib 4 block, how many times will Snap! make a recursive call to Fib 1 ? State your answer in the following box:

3




Thanks to (anonymous student) for sharing this tree-diagram with us!

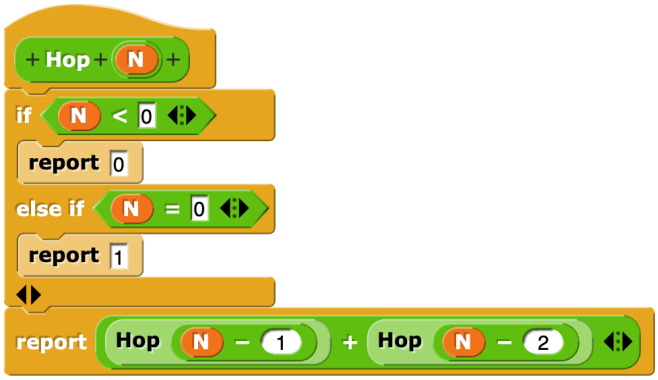
## Q4. Hop-Hop

1. We're climbing a staircase with a total of N steps. We can take either 1 or 2 steps at a time (a mix of both is possible) to reach the top. Write a procedure that takes in N and returns the number of ways that we can climb the stairs.

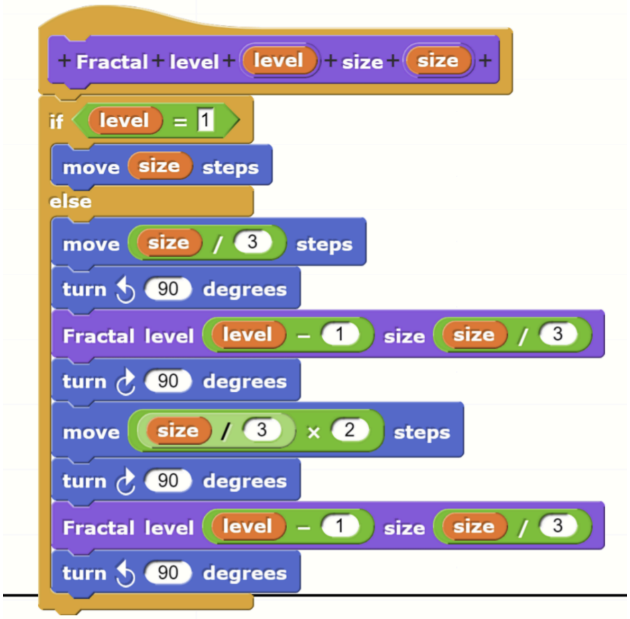
For example, if N = 3, then you can climb the stairs in the following ways:

Take 1 step, 1 step, 1 step
Take 1 step, then 2 steps
Take 2 steps, then 1 step

So, the result is = 3.

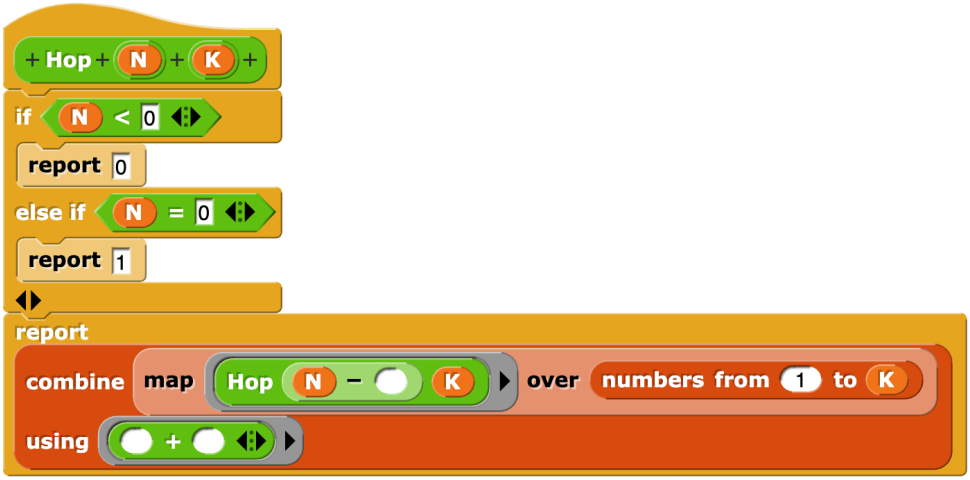



## Q5. Fractal



## Q6: Tree Recursion: Combinations

2. Now, we're getting ambitious and decide that we can climb up to K steps at a time. (In the previous problem, K was = 2.) Write a procedure that takes N and K as inputs and returns the number of ways that we can climb the stairs.




## Q7: Path of Least Resistance

| | | | | | | |
|---|---|---|---|---|---|---|
| 5 | 25 | 26 | 29 | 34 | 41 | 50 |
| 4 | 16 | 17 | 20 | 25 | 32 | 41 |
| 3 | 9 | 10 | 13 | 18 | 25 | 34 |
| 2 | 4 | 5 | 8 | 13 | 20 | 29 |
| 1 | 1 | 2 | 5 | 10 | 17 | 26 |
| 0 | 0 | 1 | 4 | 9 | 16 | 25 |
| | 0 | 1 | 2 | 3 | 4 | 5 |

In the maze above, notice that the position of each tile (box) in the maze is determined by a particular row (horizontal) and column (vertical.) The number inside a tile is called the *resistance* of that tile and is guaranteed to be non-negative. For example, the resistance of the tile at (1, 2): i.e. row 1 column 2 is = 5, and the tile is shaded gray.

You start at a particular position in the maze (R, C) and must find the *path of least*

*resistance* to reach (0, 0). A path is a sequence of steps wherein each step is either downward or leftward. The *total resistance* of a path is the sum of the resistances of each tile that is encountered on the path. For example, the total resistance of the shaded path is 50 + 41 + 32 + … 2 + 1 + 0 = 195.

Given the 5x5 board B and starting position: R, C, your function should return the total resistance that is encountered on the path from (R, C) to (0, 0) with least total resistance. In the example above, it would return 195.

Assume you have access to a block: **`Get_Resistance(Row, Col, Maze)`** which returns the resistance of the tile at the given row and column in the maze.

**`Get_Resistance`** returns $\infty$ if the provided position is out of bounds for the maze.

The next page has space for you to write your answer:

```
if row == 0 and col == 0:
    return get_resistance(0, 0, Maze) # Assume this is finite.
else:
    return min(resistance(R-1, C, M), resistance(R, C-1, M))
```

## Q8: Parking

When parking vehicles in a row, a motorcycle occupies 1 parking spot and is represented by %, while a car occupies 2 adjacent parking spots and is represented by < for the left half and > for the right half. An empty spot is denoted by a dot: .

For instance, the string ".%%.<><>" depicts an empty spot, followed by two spots with motorcycles, another empty spot, and then four spots containing two cars.

Implement count_park(n), which returns the number of ways that vehicles can be parked in n adjacent parking spots for a positive integer n. Some or all spots can be empty.

For example, count_park(1) should return 2 since the possibilities are '.' or '%'
count_park(2) should return 5 for possibilities '..', '%.', '.%', '%%', or '<>'.
For count_park(4), one of the possible configurations is '%.<>', and the total number of ways is 29.

# Q9: Subsets

Given a list of positive numbers L and a target positive number N, write a reporter that generates all the possible subsets of L that sum up to N. It should output a list of lists, wherein each inner list contains a subset of numbers from L that sum to N.

Can you write this procedure using both tree and linear recursion? Which version is more efficient? Can you write it in a manner such that its runtime is better than exponential?

Here's a somewhat naive (but perfectly valid) approach: let's first write a reporter that generates all possible subsets, then filter out those subsets that don't sum up to N.