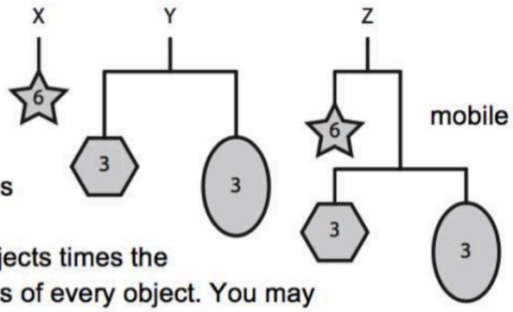
Discussion 8: Recursion II

A Little Town in Alabama...



You fondly remember the *mobiles* hanging above your crib, but you always wondered what force it took to hold them up. You wish to write `Force(mobile)` to answer that question. A mobile is either *simple* (has only a single object hanging from it), or *complex* (has a horizontal "inverted-T" rod that balances two mobiles on its left and right). Each object has a mass (the numbers in the examples on the right), and the *total* force is the *total* mass of all objects times the GRAVITY constant, also computed as the sum of the individual forces of every object. You may assume the vertical strings and horizontal "inverted-T" rods are weightless. From our example, `Force(X)` = `Force(Y)` = 6 * GRAVITY, and `Force(Z)` is double that.

Here are 4 helper blocks you'll need to use:

Block	Description
Simple? Mobile	Report if `Mobile` is *simple*, `true` for X above, `false` for Y and Z.
Left Complex Mobile	Reports the mobile on the *left* of the topmost "inverted-T" junction. Calling this function is an error if the mobile is simple. Example: `Left(Z)` would report a mobile identical to X.
Right Complex Mobile	Reports the mobile on the *right* of the topmost "inverted-T" junction. Calling this function is an error if the mobile is simple. Example: `Right(Z)` would report a mobile identical to Y.
Mass Simple Mobile	Reports the mass of the simple mobile. Calling this function is an error if the mobile is complex. Examples: `Mass(X)` would report 6, and `Mass(Left(Y))` would report 3.

a. Write a recursive function to find the force of a mobile.

```
Force(mobile):
```

b. As a function of the number of *objects* in a mobile, what is the runtime of `Force`?

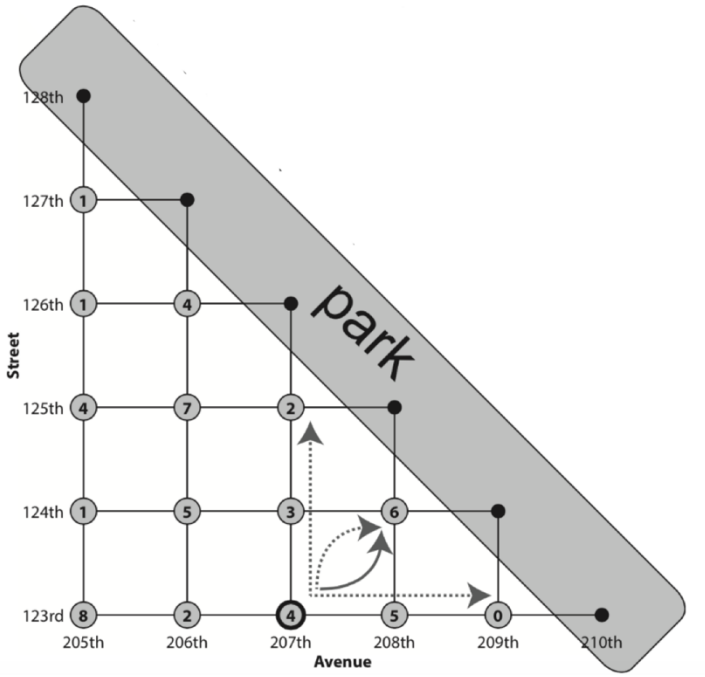
○ Constant ○ Logarithmic ○ Linear ○ Quadratic ○ Exponential

c. Your solution was recursive. Could you have written it iteratively?

○ yes ○ no

Candy Grabber

To the right is a map of your city: it is a perfect grid of streets (st) and avenues (ave), and there is a park at the northeast end of it. At each corner, there is candy, indicated by the number in each circle on the map.



At any given point, you can only walk towards the park (i.e. walk towards an increasing street or avenue number). For example, if you were at 207th Ave and 123rd St, you can only directly go to:

- 207th Ave and 124th St
- 208th Ave and 123rd St

a. Fill in the block below so that it reports the maximum candy you can grab by starting at a given intersection *if you always move towards the park*.

max x y	Takes in two numbers and reports the bigger of the two.
candy at corner ave st	Reports the amount of candy at a given ave, st corner
at the park? ave st	Reports whether a given ave, st corner is at the park

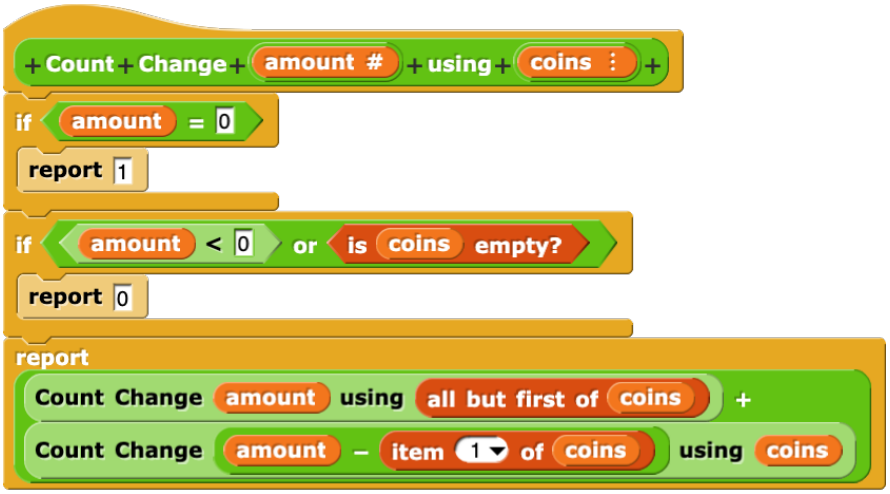
```
candy (ave) (st):

if _____________________________________________________________
     report ____________________________________________________
else
     report ____________________________________________________
            ____________________________________________________
```

b. What is the runtime of `candy` as a function of the distance to the park?

○ Constant ○ Logarithmic ○ Linear ○ Quadratic ○ Exponential

Counting Change

Recall `count change`, shown below:



a. If we swapped the order of the two `if` statements, when would it change our reported value?

_____

b. The number of ways to make change for 15 cents using coins [10, 5, 1] is 6, as our function reports. How would our function's output change if we changed the order of the coins to [1, 5, 10]?

_____

c. If we asked for the change for 2 cents using 1 penny (i.e. coins = [1]), it would report 1. What would it report if we actually had 2 kinds of pennies (i.e. coins = [1, 1])?

_____

d. If the recursive case, if we changed the second call to `Count Change` to use `all but first of (coins)`, what would our block be reporting?

○	○	○	○
The same thing as before	What would happen if we used each coin exactly once.	What would happen if we never used the first coin.	What would happen if we always used the first coin.

Extra Practice

1. Write a function called `boring multiply` which takes as input a number and a list, and recursively multiplies every item of the list by the number. It should output a new list containing the multiplied values (in order) without modifying the input list.

2. Write a function called exponent, which takes in two numbers x and y and finds x^y. You may not use the ^ block in your code.