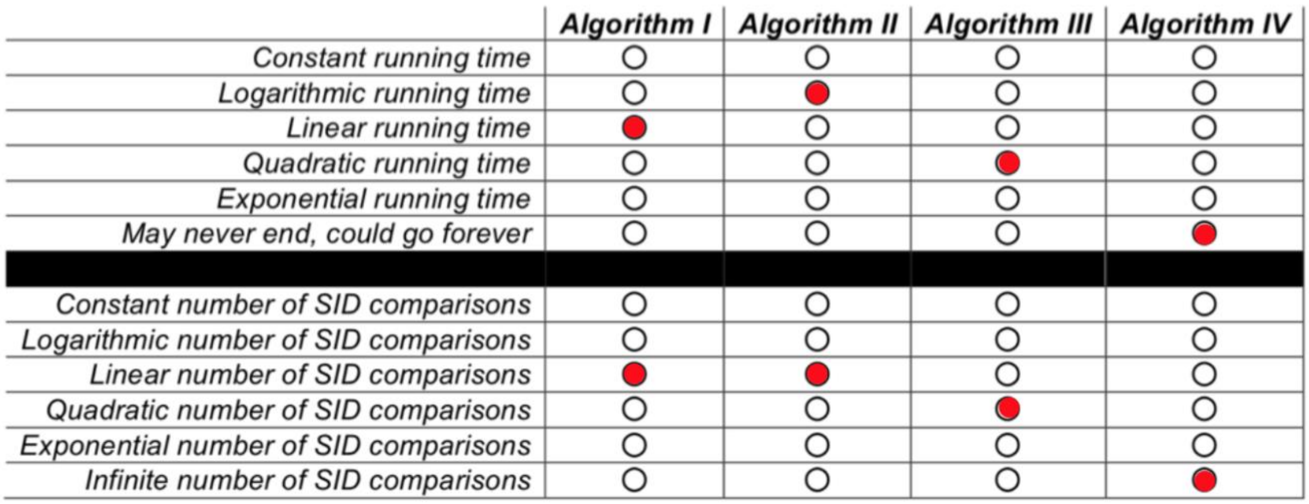
Discussion 9A: Midterm Review

Algorithmic Complexity

1. Consider the problem of trying to find out which student has the largest student ID number (SID) in a class of N students, with N a power of two (2, 4, 8, 16, ...). Note that all SIDs are unique. There are four algorithms proposed:

   - **Algorithm 1:** The students line up, sitting down. The first two students stand and compare their SIDs. The student with the smaller SID sits back down, the other remains standing. The next student in line stands up and this process repeats until there is only one student (with the largest SID) left standing.
   - **Algorithm 2:** All students stand up, pair up, and simultaneously compare SIDs, and the smaller of each pair sits down. Those still standing repeat the process, pairing up with another standing person, until there is only one left standing; that student has the largest SID.
   - **Algorithm 3:** All students are seated in a circle, facing inward. They write their SID on a sticky note and put it on the back of their neck, number facing out. A random student stands up and walks around to each person and compares their number with the number on the neck. If theirs is larger than all others, they declare themselves as the largest. If they are not, they sit down and someone who has not stood up before stands up, and the process repeats until one person declares they are largest.
   - **Algorithm 4:** Same as Algorithm III but after a person goes around and isn't the largest, rather than the next person being someone who hasn't stood up before, a random person (of the N total students) is chosen again (and it could be someone who has gotten up before).

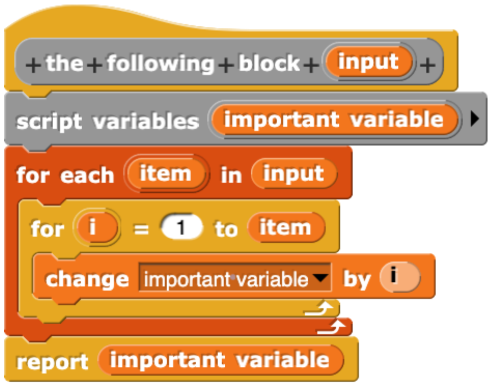
   Fill in the table for the *worst-case* running time and the *worst-case* number of SID comparisons. (Select ONE for each algorithm from the top group, and one for each from the bottom).



	Algorithm I	*Algorithm II*	*Algorithm III*	*Algorithm IV*
Constant running time	○	○	○	○
Logarithmic running time	○	●	○	○
Linear running time	●	○	○	○
Quadratic running time	○	○	●	○
Exponential running time	○	○	○	○
May never end, could go forever	○	○	○	●
Constant number of SID comparisons	○	○	○	○
Logarithmic number of SID comparisons	○	○	○	○
Linear number of SID comparisons	●	●	○	○
Quadratic number of SID comparisons	○	○	●	○
Exponential number of SID comparisons	○	○	○	○
Infinite number of SID comparisons	○	○	○	●

R1time is constant; Runtime is linear; Runntime is Quadratic; Run^{n}time is exponential.

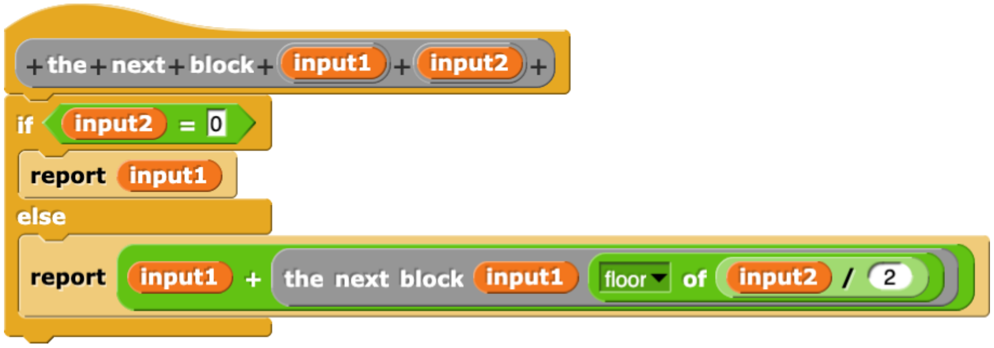
What is the runtime of the following block?



```
+the+following+block+ (input) +
script variables (important variable)
for each (item) in (input)
    for (i) = 1 to (item)
        change [important variable] by (i)
report (important variable)
```

Runtime: So this question is actually missing important information that should've been included initially. Assume that the items of the list are numbers bounded by some constant value, meaning they can be no larger than this given value. With this information, the runtime of the block should be **linear**.

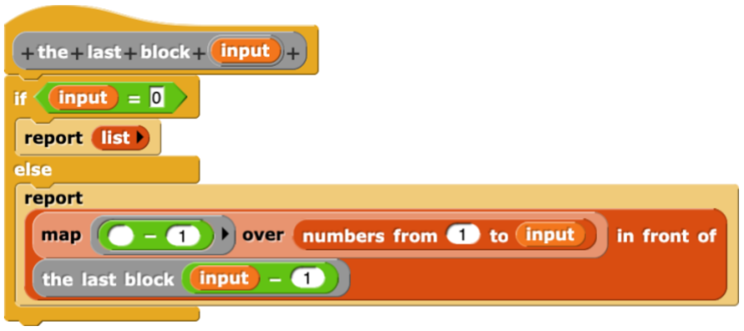
As a function of *input2*, what is the runtime of the next block?



```
+the+next+block+ (input1) + (input2) +
if <(input2) = 0>
    report (input1)
else
    report ((input1) + (the next block (input1) ([floor] of ((input2) / 2))))
```

Runtime: Logarithmic

And finally, what is the runtime of the last block?

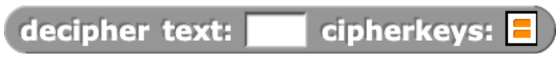


```
+the+last+block+ (input) +
if <(input) = 0>
    report (list)
else
    report
        (map (( ) - 1) over (numbers from 1 to (input))) in front of
        (the last block ((input) - 1))
```

Runtime: Quadratic

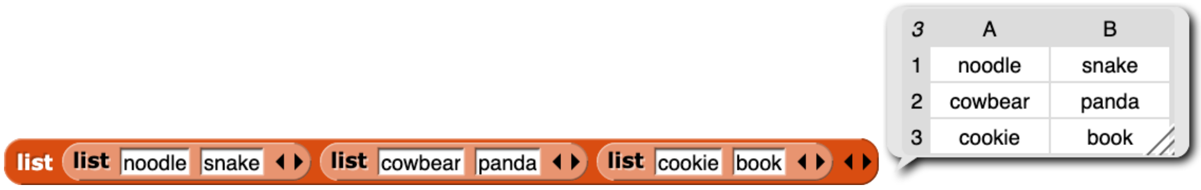
Cipher Deciphering Cipher Decipherer

Your job is to write the block *decipher text*, which takes in two inputs: a line of text *text*, and a list of lists *cipherkeys*.



```
decipher text: [ ] cipherkeys: [list]
```

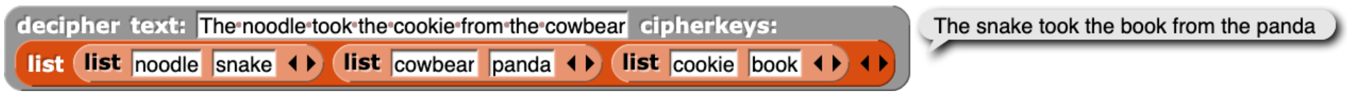
The text is what needs to be converted, and the list of lists contains pairs of codeword, real word pairs, like so:



```
list (list noodle snake) (list cowbear panda) (list cookie book)
```

3	A	B
1	noodle	snake
2	cowbear	panda
3	cookie	book

Here, the list is stating that the word noodle should be converted to snake, cowbear to panda, etc. For example:

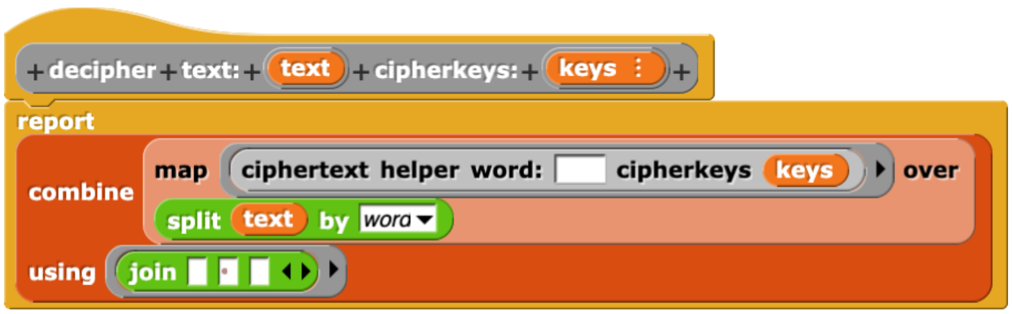


```
decipher text: The noodle took the cookie from the cowbear cipherkeys:
list (list noodle snake) (list cowbear panda) (list cookie book)
```

```
The snake took the book from the panda
```

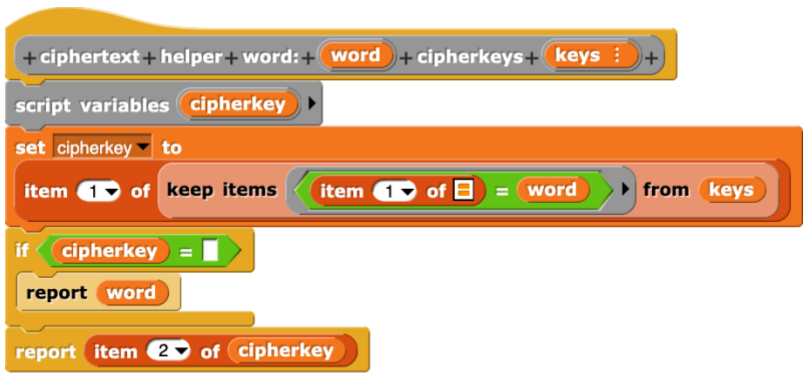
Using only higher-order functions, and optionally, a helper function that also utilizes HoFs, write out the code necessary for us to convert the text.

Decipher text(text, ciphertkeys):



```
+ decipher + text: + (text) + cipherkeys: + (keys ⋮) +
report
  combine
    map (ciphertext helper word: [ ] cipherkeys (keys)) over
    split (text) by word
  using (join [ ] [·] [ ])
```

Decipher text helper (any inputs you want):

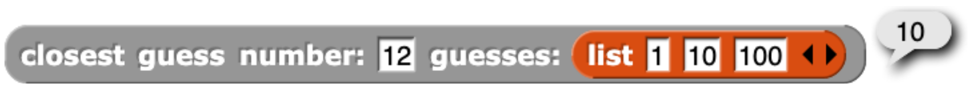


```
+ ciphertext + helper + word: + (word) + cipherkeys + (keys ⋮) +
script variables (cipherkey)
set cipherkey to
  item (1) of keep items ((item (1) of [list]) = (word)) from (keys)
if <(cipherkey) = [ ]>
  report (word)
report item (2) of (cipherkey)
```

Closest Guess!

For this question, you have to fill out the *closest guess* block which takes in a number *secret number*, and a list of numbers *guesses*. The idea is that you want to report the number in the list of guesses that's the closest to the secret number.

For example:

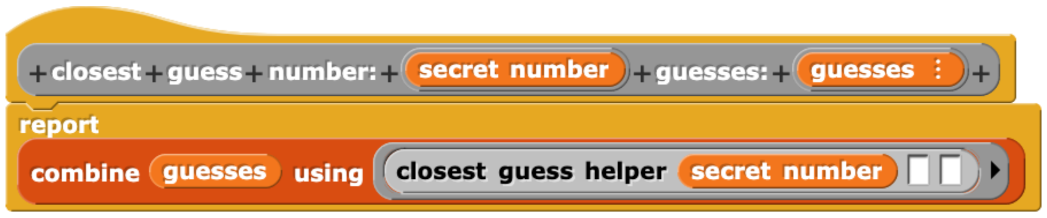


```
closest guess number: 12 guesses: list 1 10 100
```

```
10
```

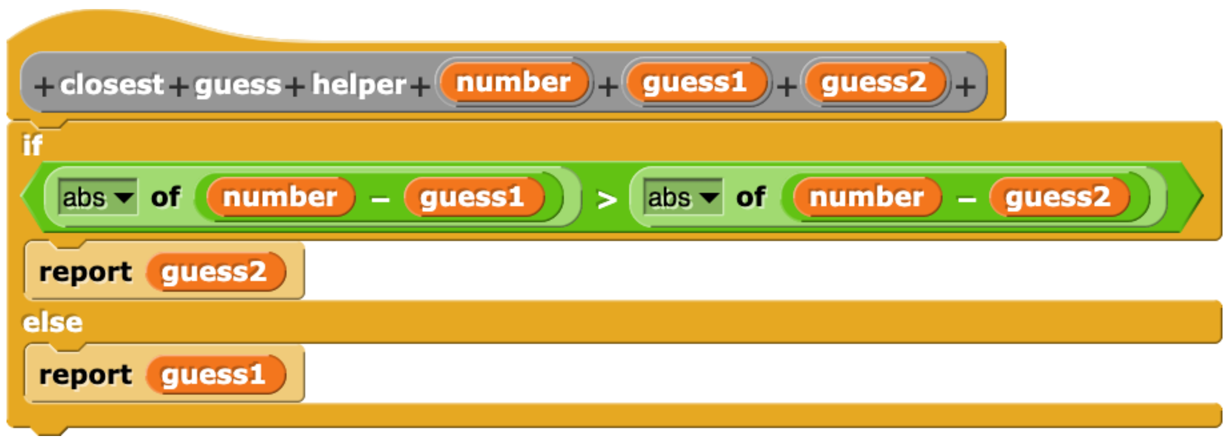
Similar to the previous question, you may only use higher-order functions, but you're free to build a helper block however you want.

Closest guess(number, guesses):



```
+closest+guess+number:+ (secret number) +guesses:+ (guesses ⋮) +
report
  combine (guesses) using (closest guess helper (secret number) [ ] [ ])
```

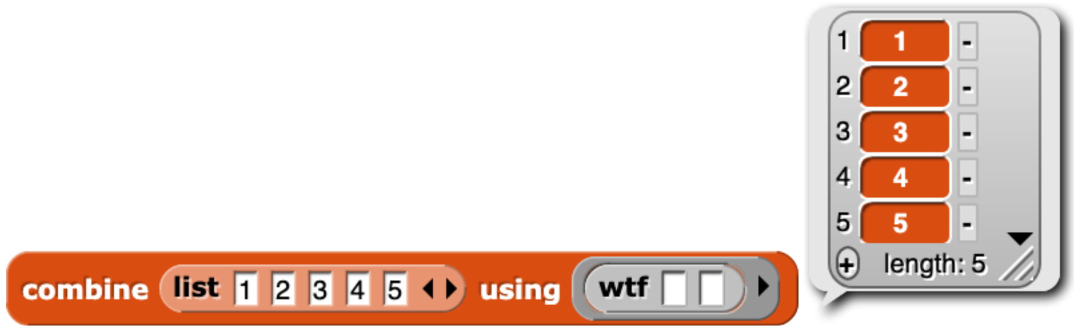
Helper(any input you want):



```
+closest+guess+helper+ (number) + (guess1) + (guess2) +
if < (abs of ((number) − (guess1))) > (abs of ((number) − (guess2))) >
  report (guess2)
else
  report (guess1)
```

??? (challenge question)

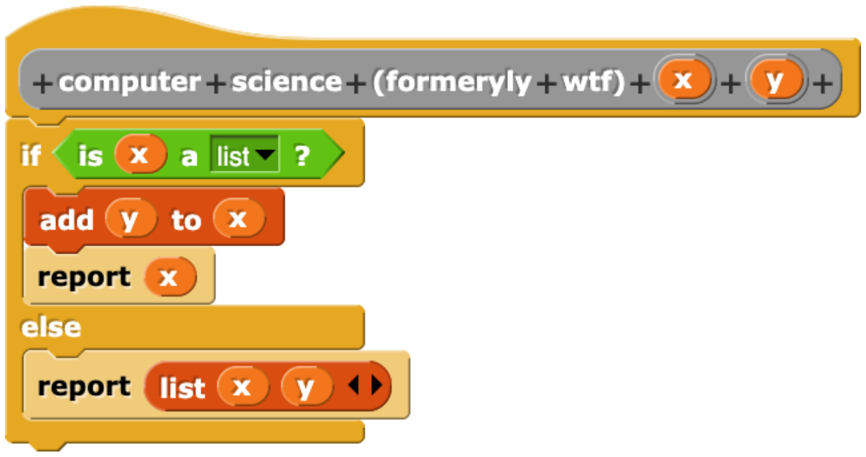
Using **combine** and a **helper block**, return a copy of a given list.



```
combine (list 1 2 3 4 5 ◂▸) using (wtf [] [] ▸)
```

```
1  1
2  2
3  3
4  4
5  5
length: 5
```

Solution:



```
+computer+science+(formeryly+wtf)+(x)+(y)+
if <is (x) a [list] ?>
    add (y) to (x)
    report (x)
else
    report (list (x) (y) ◂▸)
```