Discussion 5: Algorithmic Complexity

Algorithmic Complexity: Definitions

1. What is runtime? How do we measure it?

   Runtime is a measure of the amount of time a procedure takes to execute. But timing computer programs that execute in sub-seconds is impractical; instead, we measure runtime as the number of steps a procedure takes to execute, as a function of the input size.

2. If a function runs in O(n) time, that means it runs…

   [✓ ○ in linear time at worst]   ○ in linear time on average   ○ in linear time at best

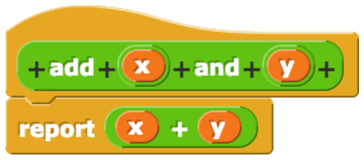
Understanding Runtimes

1. Fill in the following chart:

Runtime	**Notation**	**As input size increases by…**	**The number of steps change by…**
Constant	$O(1)$	+1	+0 (no change)
Logarithmic (base B)	$O(\log_B n)$	xB	+1
Linear	$O(n)$	+1	+1
Quadratic	$O(n^2)$	x2	x4
Exponential (base B)	$O(B^n)$	+1	xB

Runtime Practice

1. Find the runtime of each of the following blocks or processes:

a. 

```
+add+(x)+and+(y)+
report (x) + (y)
```

Runtime: Constant

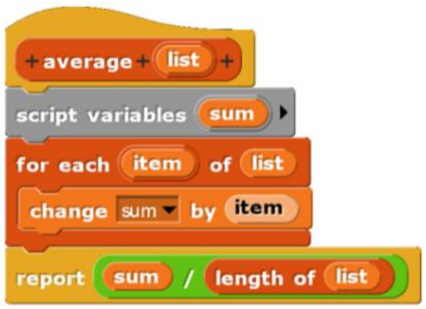
Mathematical operations take constant time to run.

b. This block takes in a value and a list and searches through every item in the list one by one to see if it can find that value.

Runtime: Linear

Here, the input size would be the length of the input list. As the length of the input list increases by 1 (i.e. an item is added to the list), the number of items the algorithm needs to look through increases by 1 too, which follows linear growth.

c.



```
+average+ (list) +
script variables (sum)
for each (item) of (list)
    change [sum] by (item)
report ((sum) / (length of (list)))
```

Runtime: Linear

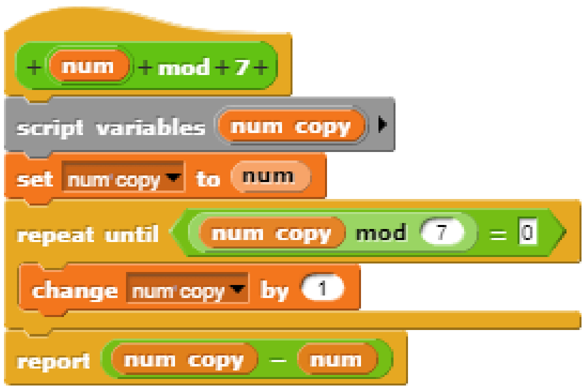
Here, the input size is the length of the input list. As the length of the input list increases by 1 (i.e. an item is added), the algorithm needs to go through one more item to add to the sum, increasing the total number of steps by 1.

d. This block takes in a value and a sorted list and searches for the value in the sorted list. Every iteration of the algorithm, it figures out which half of the list the value would be in, and then only searches in that half of the list.

Runtime: Logarithmic

Every iteration of the algorithm, it eliminates half of the input size by only searching half the list.

e.



```
+ (num) + mod + 7 +
script variables (num copy)
set [num copy] to (num)
repeat until <((num copy) mod (7)) = (0)>
    change [num copy] by (1)
report ((num copy) − (num))
```

Runtime: Constant

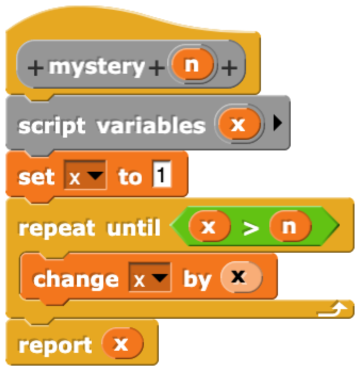
When measuring runtime, we care about how many steps the algorithm will take as the input size gets very large. This algorithm will only repeat its loop at most 6 times, no matter how big the input size gets. Thus, because the number of steps does not continue to grow as the input size grows, this algorithm takes constant time.

f. You know a secret, and you want to share it with the world. In *state 0*, you are the only person who knows the secret. Then in *state 1*, you share the secret with two friends, so three total people know the secret. Then in *state 2*, both of your friends tell two of their friends, so seven total people know the secret. This pattern continues indefinitely. As a function of the *state*, what is the order of growth of the number of people who know the secret?

Runtime: Exponential

As the state increases by 1, double the number of people will know the secret.

f.



```
+mystery+(n)+
script variables (x)
set x to 1
repeat until <(x) > (n)>
    change x by (x)
report (x)
```

Runtime: Logarithmic

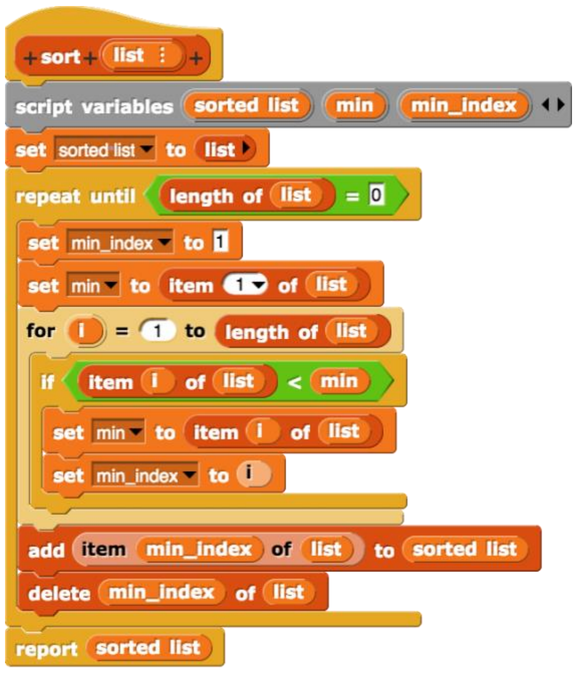
Every time the loop runs, x is doubled, thus halving the number of iterations the loop will need to take for x to be greater than n.

g. This block finds the number of pairs, sets of two values that are equal, in a list. It first starts out by finding pairs for the first element: it searches through everything after the first element and counts the number of values equal to the first element. Then, it finds pairs for the second element by searching through everything after the second element for pairs. It continues with this process until it reaches the end of the list.

Runtime: Quadratic

Every time this algorithm wants to find a pair for an item, it searches every other item in the list. Thus, if the input list had n elements, for every element the algorithm tries to find a pair for, it does O(n) work. The algorithm tries to find pairs for all n elements, which means it does O(n) work n times, so the runtime is $O(n^2)$, or quadratic runtime.

h.



```
+sort+(list :)+
script variables (sorted list) (min) (min_index)
set sorted list to (list)
repeat until <(length of (list)) = 0>
    set min_index to 1
    set min to (item 1 of (list))
    for (i) = 1 to (length of (list))
        if <(item (i) of (list)) < (min)>
            set min to (item (i) of (list))
            set min_index to (i)
    add (item (min_index) of (list)) to (sorted list)
    delete (min_index) of (list)
report (sorted list)
```

Runtime: Quadratic

There are two loops in this function: one that repeats until the length of the list is zero, and one inside of it which repeats once for every item of the list. Every time the inner loop runs, it takes O(n) time, where n is the length of the list. After it finishes running completely, the algorithm deletes an element of the input list. This process will need to repeat n times for all of the elements to be deleted and for the outer loop to stop running, so this takes $O(n^2)$, or quadratic, time.