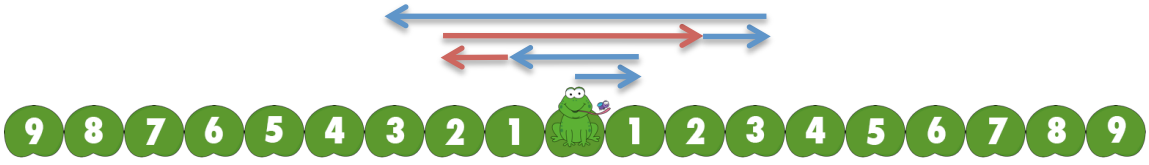
Data Structures II

Micah Stairs and William Fiset
COMP 2631 – Winter 2015
Dr. Keliher

A long time ago, in a pond far, far away, there was a row of lily pads arranged in a perfectly straight line, stretching forever in both directions. On one of these lily pads sat a frog. This frog knew that one of the lily pads was a magic lily pad, and that hopping onto this magic lily pad would transform the frog into a powerful rock star named Prince. The problem was that all the lily pads looked exactly the same, so there was no way to tell which one was the magic lily pad except to hop onto it. The only action the frog can (repeatedly) perform is to hop from its current lily pad to one of the two adjacent lily pads. Suppose the magic lily pad is exactly n hops away from the frog's current position, but the frog does not know the value of n nor the direction in which the magic lily pad lies. How can the frog perform O(n) hops and always be guaranteed to find the magic lily pad?

This is a clever question. Since the lily pads are arranged in a line stretching *forever* in both directions (and the frog does not know how far away this magic lily pad is), he cannot simply hop in one direction then turn around once he has reached the end and check the lily pads in the other direction (which would take 3n hops, or O(n) hops).

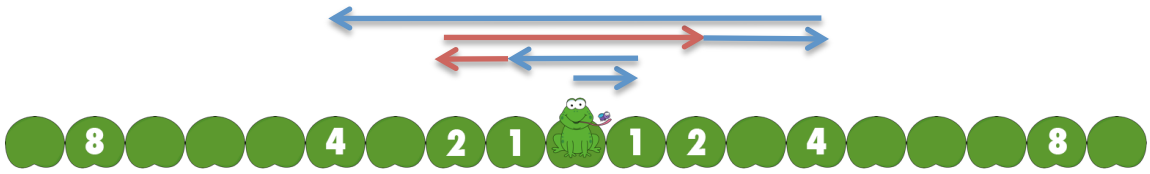
It is obvious then, that the frog must repeatedly change directions. For example, he could try hopping 1 lily pad deep in one direction, then turning around and hopping 1 lily pad deep on the other side. He could repeat this procedure, going one more lily pad deep each time.



Noting that each repetition of this procedure takes 2r+1 steps, this technique would yield the results on the right. It turns out that the relationship between the total hops and n is $\boldsymbol{n^2+2n}$*, therefore this algorithm has a complexity of* $\boldsymbol{O(n^2)}$*.*

Repetition # (r)	**# Lily pads deep (n)**	**# Total Hops**
1	1	0 + (2(1) + 1) = **3**
2	2	3 + (2(2) + 1) = **8**
3	3	8 + (2(3) + 1) = **15**
4	4	15 + (2(4) + 1) = **24**
5	5	24 + (2(5) + 1) = **35**

To reduce this algorithm down to a complexity of O(n), we need to make one small modification. Instead of simply incrementing the depth by 1 each time, we should double it.



*With the exception of the first repetition, each repetition of this procedure takes 2^r+2^{r-2} steps. It turns out that the relationship between the total hops and n is **5n-2**, therefore this algorithm has a complexity of **O(n)**.*

Repetition # (r)	**# Lily pads deep (n)**	**# Total Hops**
1	1	**3**
2	2	$3 + (2^2 + 2^0) =$ **8**
3	4	$8 + (2^3 + 2^1) =$ **18**
4	8	$18 + (2^4 + 2^2) =$ **38**
5	16	$38 + (2^5 + 2^3) =$ **78**