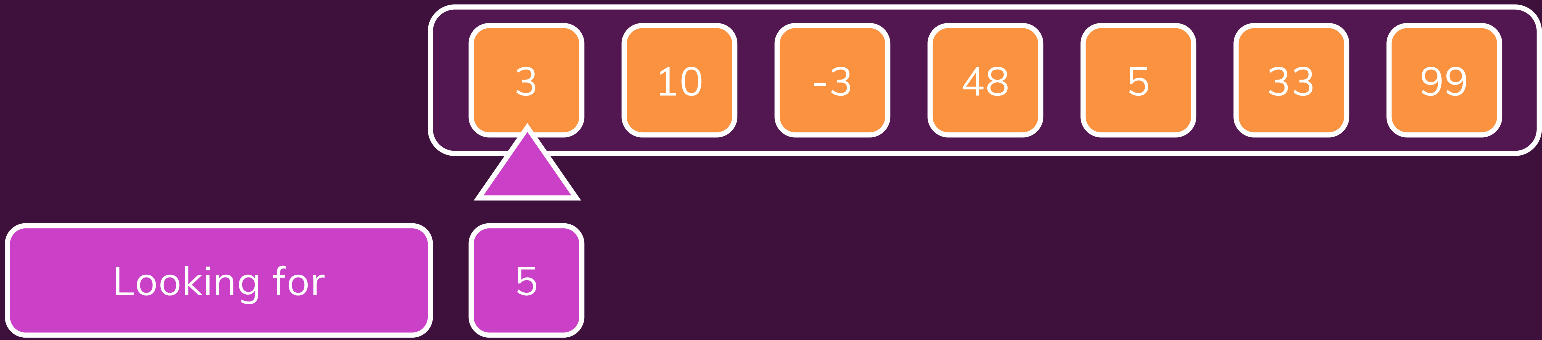
Search Algorithms?

3	10	-3	48	5	33	99

Looking for 5

You can use different algorithms for different kinds of lists – with different time complexities

Linear Search



Linear search works on **ordered and unordered** lists and **checks all items** until it finds the element you're searching.

Linear Search – Time Complexity

Best Case	Average Case	Worst Case
The item we're looking for is the very **first** item in the list	Random order, we don't know where the item is	The item we're looking for is the very **last** item in the list
O(1)	Tends to be O(n)	O(n)

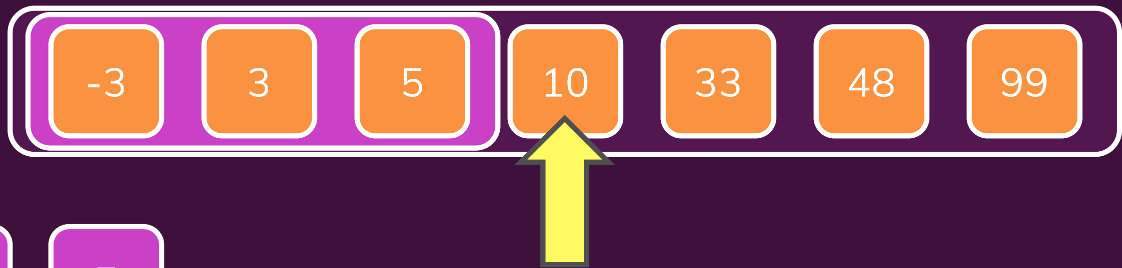
Binary Search

3	10	-3	48	5	33	99

Looking for 5

Doesn't work on unordered lists! You need to sort the list first!

Binary Search



Looking for 5

Find median and compare it to the element you're trying to find

Is it the element you're looking for?

If element wasn't found, take the half in which must be inside

Repeat!

Binary Search – Time Complexity

Best Case	Average Case	Worst Case
The item we're looking for is **right in the middle**	We don't know where the item will be	The item we're looking for is **at the beginning or end**
O(1)	Tends to be O(log n)	O(log n)

Because we split the array in half in every iteration

Recursion & Big O (The Master Theorem)

How do you derive Big O for more complex recursive algorithms?

Master Theorem

Runtime of recursion: $O(n^{log_b a})$

<u>Overall algorithm runtime (time complexity) - three cases:</u>

Recursion does more work

$O(n^{log_b a})$

Same work inside and outside of recursion

$O(n^{log_b a} \log n)$

Non-recursive part does more work

$O(fn(n))$

where

a equals the number of subproblems (number of recursion splits)

b equals the relative subproblem size (input reduction per split)

f(n) equals the runtime outside of the recursion