Big O

But that doesn't mean startups don't care about big O analysis. A great engineer (startup or otherwise) knows how to strike the right balance between runtime, space, implementation time, maintainability, and readability.

**Arrays** have O(1)-time lookups. But you need enough uninterrupted space in RAM to store the whole array. And the array items need to be the same size.

Pointer Array

That's the tradeoff. This pointer-based array requires less uninterrupted memory and can accommodate elements that aren't all the same size, but it's slower because it's not cache-friendly.

Appending an item to an array is usually an O(1) time operation, but **a single doubling append is an O(n) time operation since we have to copy all n items from our array.**

**The advantage of dynamic arrays over arrays is that you don't have to specify the size ahead of time, but the disadvantage is that some appends can be expensive**. That's the tradeoff.

**Linked lists have worst-case O(1) time appends, which is better than the worst-case O(n) time of dynamic arrays.**

So if linked lists are so great, why do we usually store strings in an array? **Because** [**arrays have O(1) -time lookups**](https://www.interviewcake.com/article/python3/data-structures-coding-interview?course=fc1&section=algorithmic-thinking#constant-time-array-lookups)**.** And those constant-time lookups come from the fact that all the array elements are lined up next to each other in memory.

The main thing we use logarithms for is **solving for x when x is in an exponent**.

