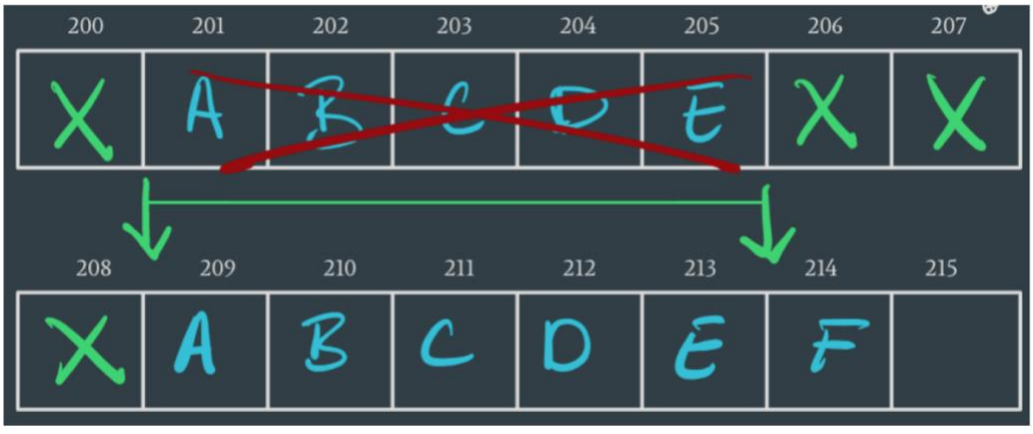
Arrays

An array is an ordered list of data that we access with a numerical index.

Generally speaking, an array is allocated upfront as a single block of memory based on the number of elements and type of data we want the array to hold.

Elements must be stored **contiguously** in memory.

But what happens when we run out of contiguous memory space?



Removing (or “splicing”) requires shifting all elements over by one to fill the gap; inserting a new element requires shifting or allocating a larger array to hold the elements, and

finding values requires iterating over the entire array in the worst case.

Arrays are most efficient when accessing the elements in random order (**by index**) or modifying at the **end of the array**.

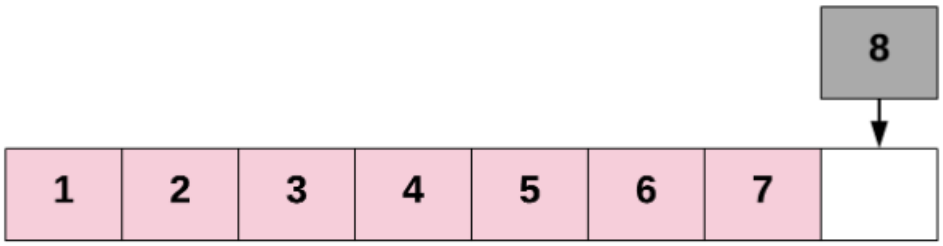
Removing, inserting, or finding arbitrary values in an array can be a **linear-time** operation (O(n)).

Common array methods:

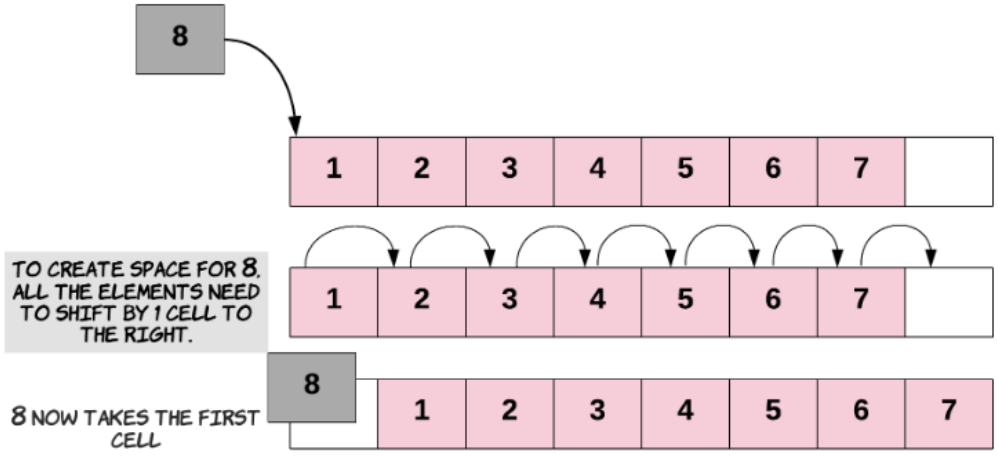
push: O(1) – last element (no need for shifting)
pop: O(1) – last element (no need for shifting)
shift: O(n)
unshift: O(n)
slice: O(n)
sort: O(n * log n)
forEach/map/filter/etc. - O(n)

Basic operations

At any point in time, we know the index of the last element of the Array, as we've kept track of it in our length variable. All we need to do for **inserting an element at the end** is to assign the new element to one index past the current last element. This would always be **O(n)** time.



To insert an element at the **start of an array**, we'll need to shift all other elements in the array to the right by one index to create space for the new element. This is a very costly operation, since each of the existing elements has to be shifted one step to the right. In terms of time complexity analysis, this is a linear time complexity: **O(n).**



2D Arrays

At a higher level, arrays can be used to store references to other types - even other arrays. 2D arrays are a common array-of-arrays pattern that can be used to store tabulated data, like a location on a grid, or the price per unit of goods at different volumes. These 2D arrays are indexed by two subscripts, one for the row and another for the column.

When calculating the size of a 2D array, simply multiply the number of elements by the size of each element.

2 pointer traversals

Removing, inserting, or finding values in arrays can be time-consuming. 2 pointer traversal is a useful technique for efficiently working with **sorted arrays**.

```
1 ▼ const reverseArray = (array) => {
2     let  i = 0;
3     let j = array.length-1;
4 ▼   while(i < j) {
5         let temp = array[i];
6         array[i] = array[j];
7         array[j] = temp;
8         i++;
9         j--;
10    }
11    return array;
12  }
13
14  console.log(reverseArray([1,2,3,4,5]))
```

When to use an array in an interview

Use an array when you need data in an ordered list with fast indexing or compact memory footprint. Don’t use an array if you need to search for unsorted items efficiently or insert and remove items frequently.

An array organizes items sequentially, one after another in memory.

Each position in the array has an **index**, starting at 0.

Strengths:

- **Fast lookups**. Retrieving the element at a given index takes $O(1)$ time, regardless of the length of the array.
- **Fast appends**. Adding a new element at the end of the array takes $O(1)$ time, if the array has space.

	Worst Case
space	$O(n)$
lookup	$O(1)$
append	$O(1)$
insert	$O(n)$
delete	$O(n)$

Weaknesses:

- **Fixed size**. You need to specify how many elements you're going to store in your array ahead of time. (Unless you're using a fancy dynamic array.)
- **Costly inserts and deletes.** You have to "scoot over" the other elements to fill in or close gaps, which takes worst-case $O(n)$ time.

Some common array problems

Find the Difference of Two Arrays

a = [1, 2, 3, 4]
b = [3, 4, 5, 6]

diff(a, b) # => [1, 2, 5, 6]

https://leetcode.com/problems/find-the-difference-of-two-arrays/

Concatenation of Array

a = [1, 2, 1]

concatenation(a) #=> [1, 2, 1, 1, 2, 1]

https://leetcode.com/problems/concatenation-of-array/

Shuffle the Array

a = [1, 2, 3]
b = [7, 8, 9]

shuffle(a, b) #=> [1, 7, 2, 8, 3, 9]

https://leetcode.com/problems/shuffle-the-array/

Flipping an image

image = [[1, 1, 0], [1, 0, 1], [0, 0, 0]]

flipAndInvert(image) #=> [[1, 0, 0], [0, 1, 0], [1, 1, 1]]

https://leetcode.com/problems/flipping-an-image/

Maximum Subarray

a = [-2, 1, -3, 4, -1, 2, 1, -5, 4]

maxSubArray(a) #=> 6 [4, -1, 2, 1]

https://leetcode.com/problems/maximum-subarray/

Contains Duplicates

a = [1, 2, 3, 1]
b = [1,2,4,5]

containsDuplicated(a) #=> true
containsDuplicated(b) #=> false

https://leetcode.com/problems/contains-duplicate/