Arrays

Interview Essentials

Arrays are one of the most basic data structures, but they make for some very hard interview questions! It's imperative that you know the array methods in your chosen interviewing language very well, because interviewers will definitely be judging you on this.

An array is a data structure that contains a group of elements. The most basic implementation of an array is a static array, in which the array has a fixed size that must be declared when it is initialized. In technical interviews, you're also going to come across questions about dynamic arrays (similar to static arrays, but with space reserved for additional elements) and multidimensional arrays (arrays in which the individual elements are also arrays).

Static Array

What Is a Static Array?

A random-access data structure that is of fixed size. "Adding" or "deleting" an element from an array requires creating an appropriately sized contiguous chunk of memory, and copying the elements that you want to keep into the new array.

Crucial Terms

- **Size:** The number of elements in the array.

Strengths and Weaknesses

Strengths

- The item lookup by index for arrays is very quick - `O(1)`.
- Arrays are a simple data structure with a very low memory footprint.

Weaknesses

- Arrays have to be implemented in contiguous memory.
- Adding or deleting elements to the array is `O(n)`, where `n` is the number of elements in the array.
- Arrays do not allow for the quick rearrangement of elements.
- Searching in the array for an entry with particular attributes is `O(n)`.

Common Operations Cheat Sheet

Operation	Description	Time complexity	Mutates structure
`append(item)`	Adds `item` to the end of the list	`O(n)`	Yes
`delete(index)`	Removes `item` from list	`O(n)`	Yes
`[index]`	Retrieves element at `index`	`O(1)`	No

Dynamic Array

What Is a Dynamic Array?

A random-access, variable sized list data structure that allows elements to be added or removed. Like a regular array, but with reserved space for additional elements.

Crucial Terms

- **Logical size:** The number of elements being used by the array's contents.
- **Physical size:** The size of the underlying array, counted in either the number of elements it has room for, or the number of bytes used.

Reserved
Reserved
Reserved
A dynamic array with a physical size of 7 elements, logical size of 4 elements.

Strengths and Weaknesses

Strengths

- On average, inserting elements to the end of the dynamic array is quick.
- Item lookup by index is `O(1)`.

Weaknesses

- Dynamic arrays must be implemented in contiguous memory.
- Dynamic arrays have inconsistent run times for adding to the array, so they're not good for real-time systems.
- Dynamic arrays do not allow for the quick rearrangement of elements.

In Interviews, Use Dynamic Arrays When...

- You'll need to add or delete information from the array.
- An (occasional) long operation won't significantly affect your program's performance.

Common Operations Cheat Sheet

Operation	Description	Time complexity	Mutates structure
`append(item)`	Adds `item` to the end of the list	`O(1)` (amoritized)	Yes
`delete(index)`	Removes `item` from list	`O(n)` in general	Yes
`delete(index)`	Removes last `item` from list	`O(1)` from end of list	Yes
`[index]`	Retrieves element at `index`	`O(1)`	No

Multidimensional Arrays

What Is a Multidimensional Array?

A multidimensional array is a random-access data structure in which the data is accessed by more than one index. Some languages, such as C#, have built-in support for multidimensional arrays, while others support multiple indices by making an "array of arrays". A multidimensional array that needs `N` numbers to reference a piece of data is called a `N`-dimensional array, or `N`D array.

While multidimensional arrays of any size are possible, it's most important for you to be familiar with 2D arrays. You will often see 2D arrays used to implement:

- Matrices `M`, where `M[i][j]` represents the number in row `i` and column `j`

- Matrices `M`, where `M[i][j]` represents the number in row `i` and column `j`
- Game boards `board` (such as chess or checkers boards), where `board[row][col]` represents the state of the board at location `(row, col)`
- Maps, where the map is divided into cells.

Crucial Terms

Different languages have different internal representations of multidimensional arrays. When implementing a 2D array as an "array of arrays" (i.e. `array[i]` is itself an array), there is no technical reason why `len(array[i]) == len(array[j])`. If the arrays have different lengths, we call them *jagged arrays*. When discussing multidimensional arrays, we will mean *regular* (or *rectangular*) arrays, in which each dimension has a fixed length.

- **Dimension:** the number of indicies needed to locate a single element in the array.
- **Size:** A tuple (d_1, d_2, ..., d_N) where d_i is the number of distinct indices for index `i`.
- **Array of arrays:** The multidimensional array `A` is represented as an array, where the entries of the array `A[i]` are arrays. Element `A[i][j]` is the `j` th element in the array `A[i]`. The arrays `A[i]` may be dynamic or static.
- **Native multidimensional array:** The multidimensional array `A` is allocated as a single block of memory. Generally, the arrays are constrained to be *regular* or *rectangular*, and you cannot alter single columns or single rows of `A`.

Strengths and Weaknesses

Strengths

- The item lookup by index for multidimensional arrays is `O(1)`.
- You can use multiple indices that make sense in the problem domain (for example, using rows and columns for a chess board), making code easier to read and maintain.
- It's easy to iterate over every element stored in a multidimensional array.

Weaknesses

- Multidimensional arrays do not allow for the quick rearrangement of elements.
- It requires a long time to change the size of a multidimensional array.

Common Operations Cheat Sheet

Operation	Description	Time complexity	Mutates structure
`append(item)`	Adds `item` to the end of the list	implem. dependent	Yes
`delete(index)`	Removes `item` from list	implem. dependent	Yes
`[index1, index2,..,indexN]`	Retrieves element at `(index1, index2,..,indexN)`	`O(1)`	No
`size()`	Returns the size of the array `(d1, d2,..,dN)`	`O(1)`	No
`dim()`	Returns the dimension of the array	`O(1)`	No