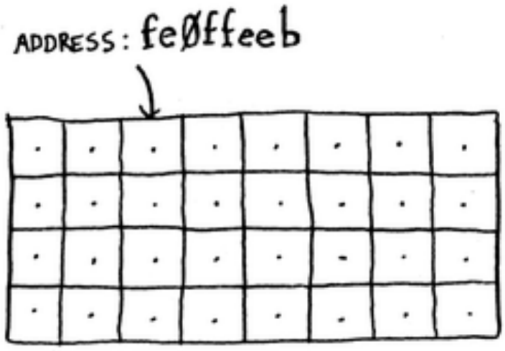
Selection Sort

How memory works

Imagine you go to a show and need to check your things. A chest of drawers is available. Each drawer can hold one element. You want to store two things, so you ask for two drawers

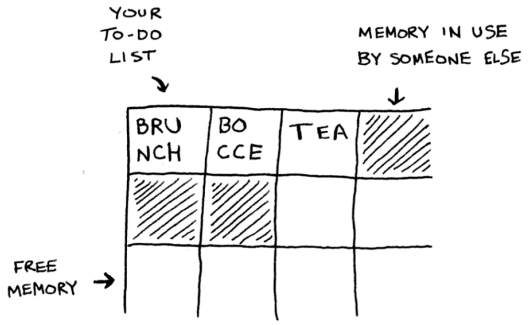
You store your two things here.
And you’re ready for the show! Tis is basically how your computer’s memory works. Your computer looks like a giant set of drawers, and each drawer has an address



Arrays and Linked lists

Arrays:

- **Imagine**: You and your friends are at the movies, and you each need a seat in a row. If you need more seats, and the row is full, you all have to move to a bigger row. Every time someone new joins, you might have to move again. 😅🍿
- **Problem**: Moving to a new row each time you need more seats is a hassle. You can try to grab extra seats just in case, but you might end up with a bunch of empty ones or still need to move if you add even more friends. 🎟️🚶🚶



You have a neat row of boxes, each with a numbered address. If you want to find the 5th item, you just need to know where the row starts and count over. No jumping through hoops! 🏠📦

Arrays are great if you want to read random elements, because you can look up any element in your array instantly

Let's run a thought experiment. Suppose Facebook keeps a list of usernames. When someone tries to log in to Facebook, a search is done for their username. If their name is in the list of usernames,
they can log in. People log in to Facebook pretty often, so there are a lot of searches through this list of usernames. Suppose Facebook uses binary search to search the list. Binary search needs random
access—you need to be able to get to the middle of the list of usernames instantly. Knowing this, would you implement the list as an array or a linked list?

Array: For Facebook's username list, use an array. Binary search requires random access to get to the middle of the list quickly, which is easiest with an array where you can jump straight to any index. It's like having a well-organized book where you can flip straight to any page! 📚🚀

Let's run a thought experiment. Suppose Facebook keeps a list of usernames. When someone tries to log in to Facebook, a search is done for their username. If their name is in the list of usernames,
they can log in. People log in to Facebook pretty often, so there are a lot of searches through this list of usernames. Suppose Facebook uses binary search to search the list. Binary search needs random
access—you need to be able to get to the middle of the list of usernames instantly. Knowing this, would you implement the list as an array or a linked list?

Downsides:

- **Resizing**: If you use an array and need to add new users, you might run into issues if the array is full. You'll need to create a larger array, move all existing users to the new array, and then add the new users. It's like having to move to a bigger house every time you get more friends! 🏠📦
- **Inefficiency**: The resizing process can be slow and inefficient, especially if you have a lot of users.
-

People sign up for Facebook pretty ofen, too. Suppose you decided to use an array to store the list of users. What are the downsides of an array for inserts? In particular, suppose you're using binary
search to search for logins. What happens when you add new users to an array?

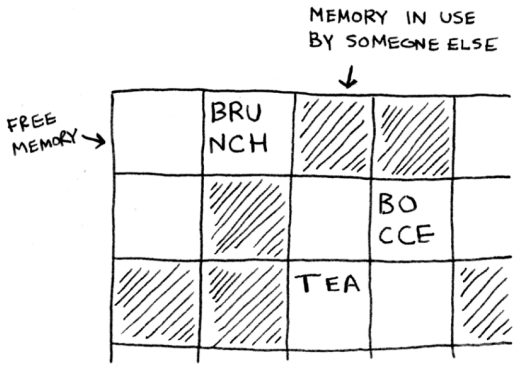
Downsides of Arrays for Inserts:

1. **Resizing Overhead**:
    - **Problem**: When the array becomes full and you need to add more users, you have to create a larger array and copy all existing usernames to it. This resizing process can be slow and time-consuming, especially if there are a lot of users. It's like having to move all your stuff to a bigger house every time you get new friends! 🏠🔄

2. **Inefficient Insertions**:
    - **Problem**: Even if the array has extra space, inserting new users might require shifting elements around to maintain the correct order. For example, if you insert a new username in the middle, you need to move all the subsequent usernames to make space. This can be slow and inefficient. It's like rearranging furniture every time you get a new guest! 🪑🔄
3. **Binary Search Impact**:
    - **Problem**: Binary search requires the array to be sorted. Every time you add a new user, you need to insert them in the correct position to keep the array sorted. If you don't maintain order, binary search won't work efficiently. Keeping the array sorted while frequently inserting new users can be cumbersome and impact performance. It's like making sure all your friends are always in the right order for a group photo! 📸🔍

Linked Lists:

- **Imagine**: You and your friends sit on a magical couch that can stretch or shrink. If a new friend shows up, the couch just grows to fit everyone without moving! 🛋️✨
- **Benefit**: Adding or removing friends is super easy. No moving around; the couch just adjusts itself. 🪄👍



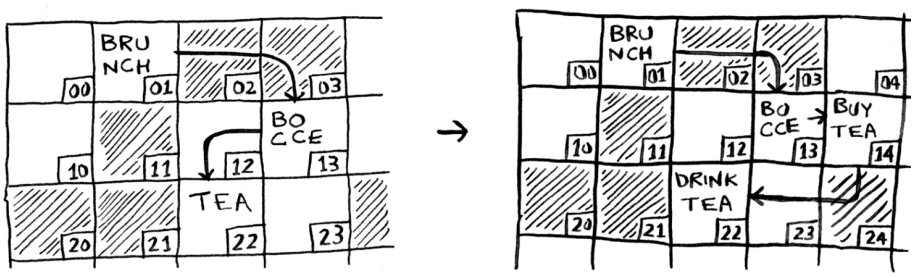
The Top-10 List Website Trick

Imagine a website showing you the Top 10 Best TV Villains. Instead of showing you the whole list at once, it makes you click "Next" for each villain. So you start with Newman and click "Next" 9 times to finally see Gustavo Fring at #1. This way, the site gets to show you 10 pages of ads. It's a bit like being stuck in a never-ending loop of boring waiting! 😩➡️📄

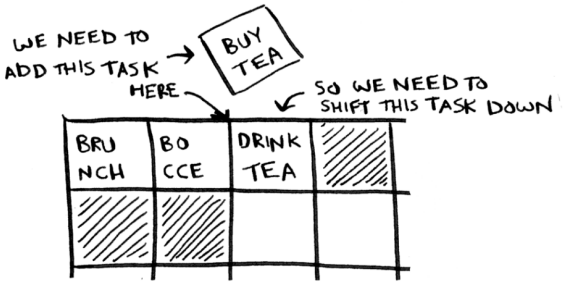
It would be much better if the whole list was on one page and you could click each person's name for more info.

You're on a scavenger hunt where each clue leads you to the next one. To find the final clue, you have to follow each clue in order from the beginning to the end. It's great if you're moving through one clue at a time but a pain if you need to jump to a specific clue! 🕵️🔍

Inserting into the middle of a list



But for arrays, you have to shift all the rest of the elements down.



And if there's no space, you might have to copy everything to a new location! Lists are better if you want to insert elements into the middle.

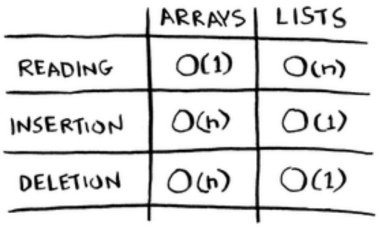
Suppose you're building an app for restaurants to take customer orders. Your app needs to store a list of orders. Servers keep adding orders to this list, and chefs

take orders of the list and make them.
It's an order queue: servers add orders to the back of the queue, and the chef takes the first order of the queue and cooks it. Would you use an array or a linked list to implement this queue?
(Hint: Linked lists are good for inserts/deletes, and arrays are good for random access. Which one are you going to be doing here?)

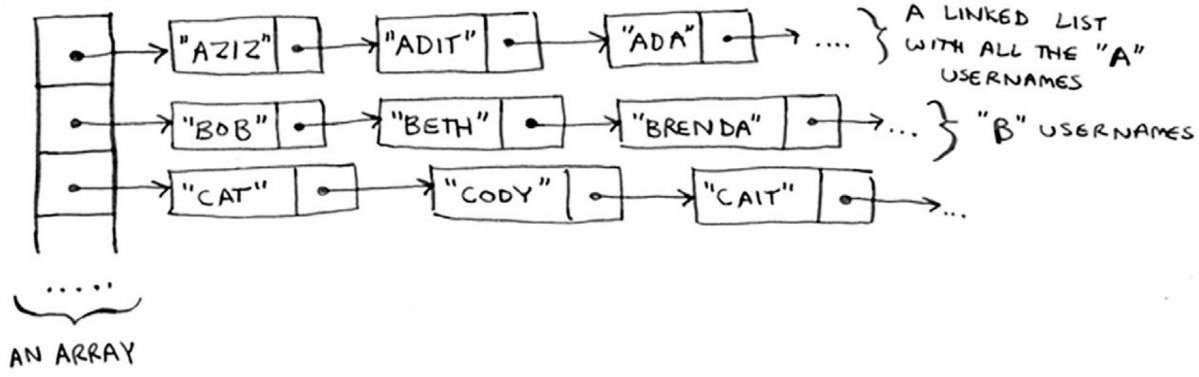
Linked List: Use a linked list for your restaurant's order queue. Servers can easily add new orders to the back, and chefs can quickly take orders from the front. It's like a never-ending line at a food truck: orders get added to the end, and the first one in line gets served first. 🍔🔄

In a Nutshell:

- **Arrays**: Great for when you know the exact number of seats you need, but moving can be a pain if you need more.
- **Linked Lists**: Perfect for when you want a flexible, magical couch that grows and shrinks as needed without any hassle! 🌟🛋️



In reality, Facebook uses neither an array nor a linked list to store user information. Let's consider a hybrid data structure: an array of linked lists. You have an array with 26 slots. Each slot points to a
linked list. For example, the first slot in the array points to a linked list containing all the usernames starting with a. The second slot points to a linked list containing all the usernames starting with b,
and so on



Inserting New Users

- **Faster Than Arrays**:

  In an array, adding a new user often involves resizing and shifting elements, which can be slow. In contrast, in the hybrid structure, you simply add the user to the end of the appropriate linked list. No resizing is required.

- **Comparable to Linked Lists**:

  Adding a user to a linked list (inside the array) is also quick, as you just append the user to the end of the list. The hybrid structure handles this as efficiently as a standalone linked list.

Searching for Users

- **Faster Than Linked Lists**:

  Linked lists require a linear search to find a username. The hybrid structure narrows down the search by first using the array to quickly locate the correct linked list (based on the first letter of the username). Then, you search within that smaller linked list. This two-step process can be faster than searching through an entire linked list.

- **Slower Than Arrays**:

  In an array, if the list is sorted, binary search provides quick access. The hybrid structure involves a two-step process: locating the right linked list and then searching within it. Although it's generally faster than searching a full linked list, it's not as fast as a well-implemented array with binary search.