Why Data Structures?

Learn what data structures are, why they are useful, and how you can use them effectively.

What even are data structures?

At the backbone of every program or piece of software are two entities: data and algorithms. Algorithms transform data into something a program can effectively use. Therefore, it is important to understand how to structure data so algorithms can maintain, utilize, and iterate through data quickly.

Data structures are the way we are able to store and retrieve data. You may already be familiar with Python lists and dictionaries or Javascript arrays and objects. If so, you know that lists and arrays are sequential with data accessed by index while dictionaries and objects use a named key to store and retrieve information.

The data structures that exist in programming languages are pretty similar to real-world systems that we use outside of the digital sphere. Imagine that you go to the grocery store. At this particular grocery store, the frozen pizza is stored next to the bell peppers and the toothbrushes are next to the milk. The store does not have signs that indicate where different items are located. In this disorganized grocery store, you would have a pretty difficult time trying to find what you were looking for!

Fortunately, most grocery stores have a clear order to the way the store is stocked and laid out. Similarly, data structures provide us with a way to organize information (including other data structures!) in a digital space.

How are data structures used?

Data structures handle four main functions for us:

- Inputting information
- Processing information
- Maintaining information
- Retrieving information

Inputting is largely concerned with how the data is received. What kind of information can be included? Will the new data be added to the beginning,

end, or somewhere in the middle of the existing data? Does an existing point of data need to be updated or destroyed?

Processing gets at the way that data is manipulated in the data structure. This can occur concurrently or as a result of other processes that data structures handle. How does existing data that has been stored need to change to accommodate new, updated, or removed data?

Maintaining is focused on how the data is organized within the structure. Which relationships need to be maintained between pieces of data? How much memory must the system reserve (*allocate*) to accommodate the data?

Retrieving is devoted to finding and returning the data that is stored in the structure. How can we access that information again? What steps does the data structure need to take to get the information back to us?

Different types and use cases for data will be better suited to different manners of inputting, processing, storing, and retrieving. This is why we have several data structures to choose from… and the ability to create our own!

Choosing the best data structure

Your plumber probably would not be the best professional to diagnose an illness and your doctor probably wouldn't be the best person to do your taxes — they are each better suited for other tasks! Similarly, different data structures are better suited for different tasks. Choosing the wrong data structure can result in slow or unresponsive code (and mess up your program!), so it's important to consider a few factors as you make your decision:

1. What is the intended purpose for the data? Do any data structures have built-in functionality that is ideally suited for this purpose? Do you want to search, sort, or iterate data in a way in which certain data structures would be better suited than others?
2. Do you want or need control over how memory is set aside to store your data? Data structures that use *static memory allocation* (e.g., stacks or arrays) will manage memory for you and assume a fixed amount of memory upon instantiation with a cap on how much data may be added. Data structures that utilize *dynamic memory allocation* (e.g., heaps or linked lists) allow you to allocate and reallocate memory within the life of the program. While memory allocation is not something that you'll need to consider in languages like Python or Javascript (these

languages will manage memory for you, regardless of which data structure you use), it is something to bear in mind when working in other languages like C.
3. How long will it take different data structures to accomplish various tasks relative to other data structures? Technically, two data structures may both be able to accomplish the same task for you, but one may be quite a bit faster. This consideration, known as *runtime* will be covered further in depth when you explore all the nifty tricks of asymptotic notation.

As you've seen, data structures are the essential building blocks that we use to organize all of our digital information. Choosing the right data structure allows us to use the algorithms we want and keeps our code running smoothly. Understanding data structures and how to use them well can play a vital role in many situations including:

- technical interviews in which you may be asked to evaluate and determine runtime for data structures given specific algorithms
- day-to-day work for many software engineers who manipulate data stored in structures
- data science work where data is stored and accessed through data structures
- a whole lot more!