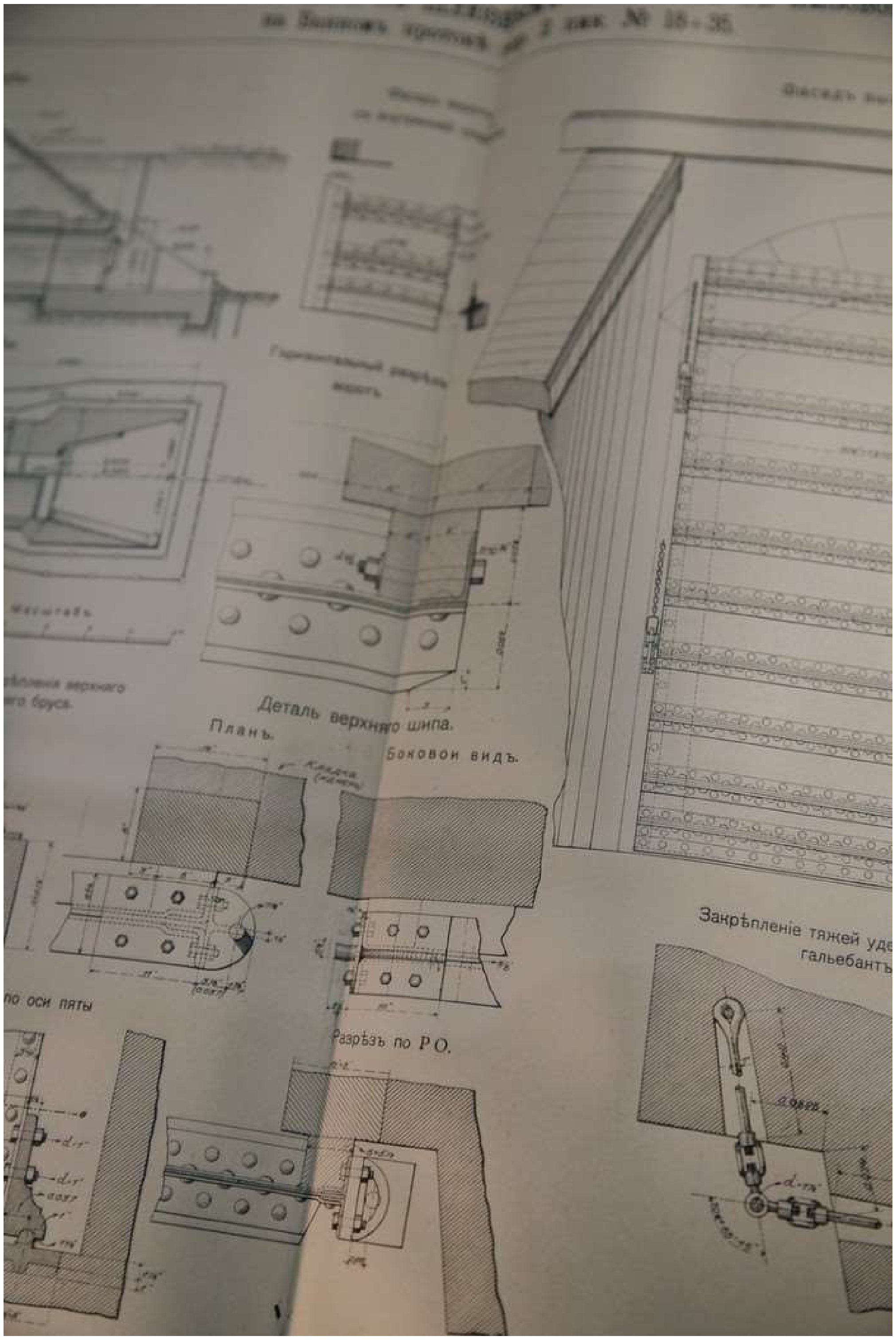
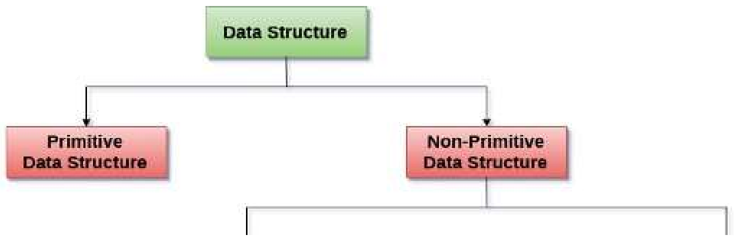
Interview Cheat Sheet: When to use which Data Structure?

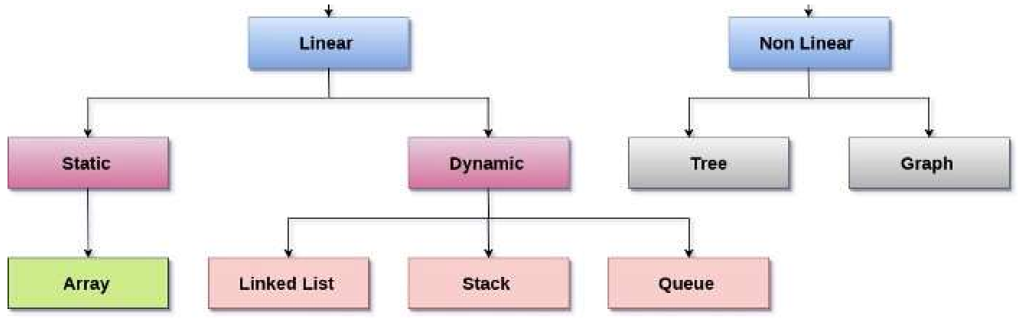
Sometimes,we get confuse when to use which data structures using interviews it can be overwhelming to think about which data structure you need to use. We will briefly discuss the use cases for the various categories of data structures.

Prateek Tiwari Follow
Apr 23, 2020 · 4 min read ★



A **data structure** is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data.





Lists are generally used when you need to just store data in some ordered manner. Runtime isn't too important when you're using a list. The two main types of lists are LinkedLists and ArrayLists.

LinkedList

1. get() = O(N)
2. add() = Θ(1)
3. remove() = Θ(N)
4. space = O(N)

ArrayList

1. get() = O(1)
2. add() = Θ(N)
3. remove() = Θ(N)
4. space = O(N)

Sets

are generally used when you have unordered data. In sets, you are not allowed to have any duplicates and you can only store keys! Essentially, sets are just maps with some dummy value. Data Structures that use sets are used when you just need to store elements.

Maps

are data structures that store a key and a value. If you insert a key value pair into a map where that key already exists, you replace that key's value with the value of the key value pair you are inserting. You declare Maps in the following manner:

1. Map<Key type, Value type>

2. In maps, you have an immutable key and values that can be any data type. Frequently, maps can be used

3. to see how frequently a key turns up.

Trees

These can be used whenever the keys in questions are implementing comparable. Trees can be used when you are attempting to find some sort of order. Specific types of self balancing trees are 2–3 Trees and Left Leaning Red Black Trees. You would choose to use trees over hashing when the key is hard to hash. This can occur when a key is very long, and would take a lot of time to perform arithmetic on it or analyze it.

Binary Search Tree

1. search() = Average: O(logN) Worst Case: O(N)

2. insert() = Average: O(logN) Worst Case: O(N)

3. delete() = Average: O(logN) Worst Case: O(N)

4. space = O(N)

Balanced Search Trees (2–3 Trees and Left Leaning Red Black Trees)

1. get() = Θ(logN)

2. insert() = Θ(logN)

3. remove() = Θ(logN)

4. space = O(N)

Hashing data structures

are used when you need Θ(1) runtime for all operations assuming that you have a good hashcode. You would also want to use hashing data sets when your data is unordered.

Stacks

These are used when you want to look at the most recent thing that has been done. Stacks are known as first in last out data types (FILO). This can be applied to search history, delivery items etc. Stacks are often used in depth first search in trees and also graphs, which we will go over in the next chapter.

Stack

1. push() = Θ(1)
2. pop() = Θ(1)
3. peek() = Θ(1)
4. space() = O(N)

Queues

These are used when we need to view items by the order in which they were done. Queues are known as first in last out data types (FIFO). In the real world, this is used when we have to keep track of waitlists. Additionally Queues are used for breadth first search in trees as well as graphs.

Queue

1. add() = Θ(1)
2. remove() = Θ(1)
3. peek() = Θ(1)

Heaps or Priority Queues

These can be used whenever the keys we are examining implement Comparable. Anytime you need the most or least of something, use a Heap. These are not for overall order, unless it's taking out one item at a time.

Heap

1. search() = Θ(1)
2. insert() = Average: O(1) Worst Case: O(logN)
3. delete() = O(log(N))

4. peek() = O(1)

5. space = O(N)

Tries

are used when you want to perform searches and insertions using prefixes. There are 2 types of tries, normal tries and ternary search tries. Normal tries have a very fast speed but in exchange they take up a lot of space. Ternary search tries on the other hand are relatively slow but take up much less space.

Tries

1. M is the length of the string you are attempting to insert/search for, N is amount of keys, L is

2. the length of the longest key, and R is the size of the alphabet.

3. search() = Θ(M)

4. insert() = Θ(M)

5. space = O(N ∗L∗R)

2. Ternary Search Tries

1. N is the amount of keys and L is the length of the longest key

2. search() = Θ(N)

3. insert() = Θ(N)

4. space = O(N ∗L)

Final Thoughts:

A cheat sheet for the time complexities of the data structure operations and it can be overwhelming to think about which data structure you need to use.

I hope you found this article useful as a simple introduction to data structures. I would love to hear your thoughts.

With that in mind, I am going to end this article if you like this just follow and clap. 👏

Datastrucutre Interview Cheatsheet Interview Preparation Software Engineering