Introduction to Algorithms & Data Structures 2

A solid foundation for the real world of machine learning and data analytics

Bolakale Aremu

Ojula Technology Innovations

Introduction to Algorithms & Data Structures 2

First Edition

ISBN: 9791222095417

Ojula Technology Innovations is a leading provider of customized learning solutions with employees residing in nearly 45 different countries and sales in more than 130 countries around the world. For more information, please contact OjulaTech@gmail.com.

Printed in the United States of America
Print Number: 01
Print Year: April 2023

I am indebted to my mother for her love, understanding and support throughout the time of writing this textbook.

Bolakale Aremu

Table of Contents

About The Author

My name is Bolakale Aremu. My educational background is in software development. I have a few colleagues who are software developers and system engineers. I spent over 17 years as a software developer, and I've done a bunch of other things too. I've been involved in SDLC/process, data science, operating system security and architecture, and many more. My most recent project is serverless computing where I simplify the building and running of distributed systems. I always use a practical approach in my projects and courses.

Bolakale Aremu
CEO, Ojula Technology Innovations
Web developer and Software Engineer
Ojulaweb.com

0. What You Will Learn in Volume 2

This book is the second volume of a four-part series titled *Introduction to Algorithms & Data Structures*. The design of an efficient algorithm for the solution of the problem calls for the inclusion of appropriate data structures. In the field of computer science, data structures are used to store and organize data in a way that is easy to understand and use. They are used to organize and represent data in a way that will make it easier for computers to retrieve and analyze it. These are the fundamental building blocks that any programmer must know how to use correctly in order to build their own programs.

0.1. Benefits of learning about algorithms and data structures

First, they will help you become a better programmer. Another benefit is that they will make you think more logically. Furthermore, they can help you design better systems for storing and processing data. They also serve as a tool for optimization and problem-solving.

As a result, the concepts of algorithms and data structures are very valuable in any field. For example, you can use them when building a web app or writing software for other devices. You can apply them to machine learning and data analytics, which are two hot areas right now. If you are a hacker, algorithms and data structures in Python are also important for you everywhere.

Now, whatever your preferred learning style, I've got you covered. If you're a visual learner, you'll love my clear diagrams and illustrations throughout this book. If you're a practical learner, you'll love my hands-on lessons so that you can get practical with algorithms and data structures and learn in a hands-on way.

0.2. Course Structure

There are three volumes in this course. This is volume two. In volume one, I took a deep dive into the world of algorithms. I covered what **algorithms** are, how they work, and where they can be found (real life applications).

In this part of the series (volume two), we'll work through an introduction to data structures. You're going to learn about two introductory data structures - **arrays** and **linked lists**. You'll look at common operations and how the runtimes of these operations affect our everyday code.

In the third volume, you're going to bring your knowledge of algorithms and data structures together to solve the problem of sorting data using the Merge Sort algorithm. We will look at algorithms in two categories: **sorting** and **searching**. You'll implement well-known sorting algorithms like Selection Sort, Quicksort, and Merge Sort. You'll also learn basic search algorithms like Sequential Search and Binary Search.

At the end of many sections of this course, short practice exercises are provided to test your

understanding of the topic discussed. Answers are also provided so you can check how well you have performed in each section. At the end of the course, you will find a **link to download more helpful resources, such as codes and screenshots used in this book, and more practice exercises**. You can use them for quick references and revision as well. My **support link is also provided** so you to contact me any time if you have questions or need further help.

By the end of this course, you will understand what algorithms and data structures are, how they are measured and evaluated, and how they are used to solve real-life problems. So, everything you need is right here in this book. I really hope you'll enjoy it. Are you ready? Let's dive in!

2. Introduction to Data Structures

In this part of the course, I will introduce data structures. First, we're going to answer one fundamental question: **why do we need more data structures than a programming language provides?** This question will be answered in section 2.3.1. For now, let's do some housekeeping.

In this course, we're going to rely on the concepts we learned in volume one (*Introduction to Algorithms & Data Structures 1*), namely big O notation, space and time complexity and recursion. If you're unfamiliar with those concepts or just need a refresher, check out volume one. In addition, this course does assume that you have some programming experience.

We're going to use data structures that come built into nearly all programming languages as our point of reference. While we will explain the basics of how these structures work, we won't be going over how to use them in practice. If you're looking to learn how to program in Python before digging into this content, check out the following book that completely simplifies Python programming language for beginners:

https://www.scribd.com/book/513773394/Python-Programming-from-Beginner-to-Paid-Professional-Part-1-Learn-Python-for-Automation-IT-with-Video-Tutorials (or use this BIT.LY short link: http://bit.ly/3ZKREhB).

Even if you know nothing about Python. as long as you understand the fundamentals of programming, you should be able to follow along pretty well. If you're good to go, then awesome.

2.1. Course Overview

Let's start with an overview of this course. The first thing we're going to do is to explore a data structure we are somewhat already familiar with: arrays. If you've written code before, there's a high chance you have used an array. In this course, we're going to spend some time understanding how arrays work, what are the common operations on an array, and what are the run times associated with those operations.

Once we've done that, we're going to build a data type of our own called a **linked list**. In doing so, we're going to learn that there's more than one way to store data. In fact, there's way more than just one way. We're also going to explore what motivates us to build specific kinds of structures and look at the pros and cons of these structures. We'll do that by exploring four common operations: accessing a value, searching for a value, inserting a value, and deleting a value.

After that, we're going to circle back to algorithms and implement a new one: a sorting algorithm. In the introduction to algorithms course (volume one), we implemented a binary search algorithm. A precondition to binary search was that the list needed to be sorted. We're

going to try our hand at sorting a list and open the door to an entirely new category of algorithms. We're going to implement our sorting algorithm on two different data structures and explore how the implementation of one algorithm can defer based on the data structure being used.

We'll also look at how the choice of data structure potentially influences the runtime of the algorithm. In learning about sorting, we're also going to encounter another general concept of algorithmic thinking called divide and conquer. Along with recursion, divide and conquer will be a fundamental tool that we will use to solve complex problems, all in due time. In the next section let's talk about arrays.

2.2. Exploring Arrays

2.2.1. Array Basics

A common data structure built into nearly every programming language is the array. Arrays are a fundamental data structure and can be used to represent a collection of values. But it is much more than that. Arrays are also used as building blocks to create even more custom data types and structures. In fact, in most programming languages, text is represented using the *string* type and, under the hood, strings are just a bunch of characters stored in a particular order in an array.

What is a Data Structure?

Before we go further and dig into arrays, what exactly is a data structure? A data structure is a way of storing data when programming. It's not just the collection of values and the format they're stored in, but the relationship between the values in the collection as well as the operations applied on the data stored in the structure.

An array is one of very many data structures in general. An array is a data structure that stores a collection of values where each value is referenced using an **index** or a **key**. An example is shown in Figure 2.2.1.

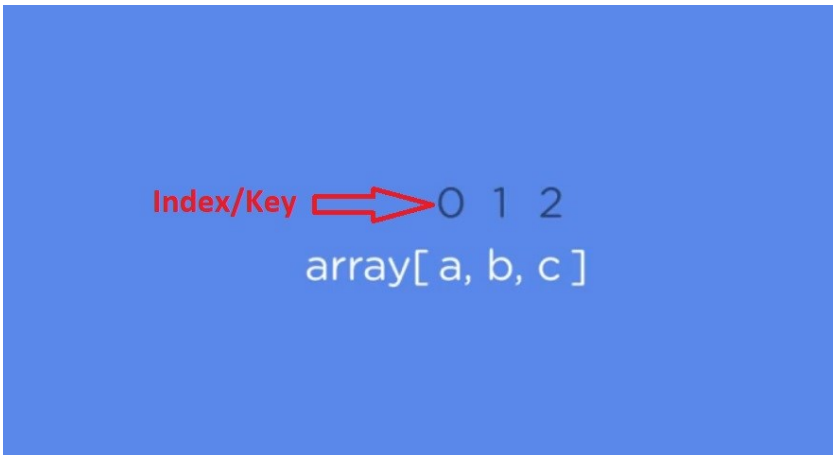

Figure 2.2.1: An example of an array with its values (a, b, c) and keys (0, 1, 2)

A common analogy for thinking about arrays is as a set of train cars. Each car has a number, and these cars are ordered sequentially. Inside each car (the array in this analogy) some data are stored. While this is the general representation of an array, it can differ slightly from one language to another, but for the most part, all these fundamentals remain the same. In a language like Swift or Java, arrays are **homogenous structures**, which means they can only contain values of the same type. If you use an array to store integers in Java, it can only store integers.

In other languages like Python, arrays are **heterogeneous structures** because they can store any kind of value. So, in Python for example, you can mix numbers and text with no issues. Now,

regardless of this nuance, the fundamental concept of an array is the index. This index value is used for every operation on the array, from accessing values to inserting, updating and deleting.

In Python, the language we're going to be using for this course, is a little bit confusing. The type that we generally refer to as an array in most languages is best represented by the **list** type in python. Python does have a type called array as well, but it's something different, so we're not going to use it. While Python calls it a list, when we use a list in this course, we'll be talking about concepts that apply to arrays as well in other languages, so definitely don't skip any of this.

There's one more thing. In computer science, a list is actually a different data structure than an array, and in fact, we're going to build a list later on in this course. Generally, though, this structure is called a linked list as opposed to just list, so hopefully the terminology isn't too confusing. To properly understand how arrays work, let's take a peek at how arrays are stored under the hood.

An array is a contiguous data structure. This means that the array is stored in blocks of memory that are right beside each other with no gaps. The advantage of doing this is that retrieving values is very easy. In a non-contiguous data structure, and we're going to build one soon, the structure stores a value *as well as* a **reference** to where the next value is.

To retrieve that next value, the language has to follow that reference, also called a **pointer**, to the next block of memory. This adds some overhead, which, as you'll see, increases the runtime of common operations. A moment ago, I mentioned that depending on the language, arrays can either be homogenous (containing the same type of value) or heterogeneous where any kind of value can be mixed. This choice also affects the memory layouts of the array.

For example, in a language like C, swift or Java, where arrays or homogenous, when an array is created, since the kind of value is known to the language **compiler**, and you can think of the compiler as the brains behind the language, it can choose a contiguous block of memory that fits the array size and values created.

If the values were integers, assuming an integer took up space represented by one block, then for a five-item array, the compiler can allocate five blocks of equally sized memory. Figure 2.2.2.

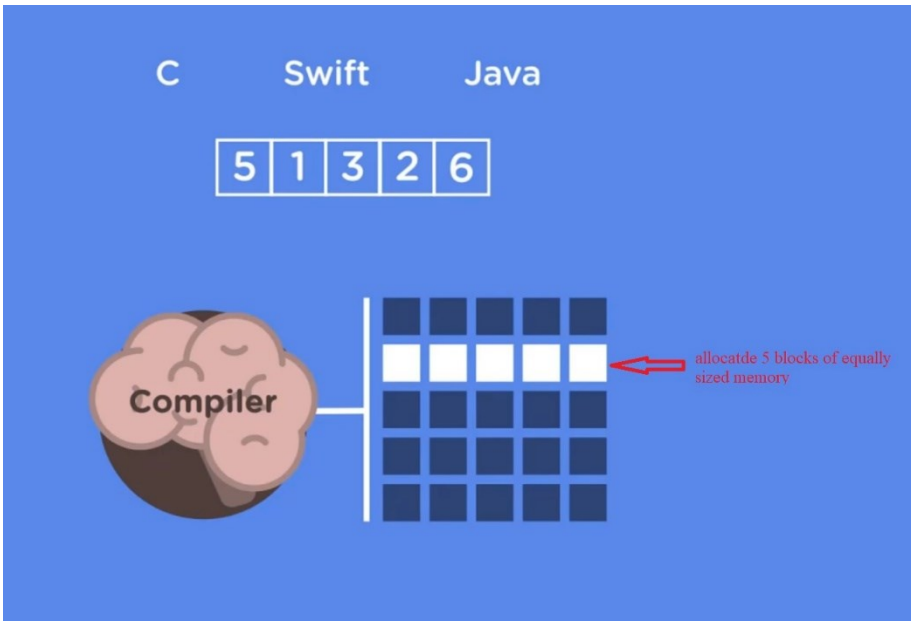

Figure 2.2.2: Illustration of how a compiler allocates 5 blocks of equally sized memory

In Python, however, this is not the case. We can put any value in a Python list. There's no restriction. The way this works is a combination of contiguous memory and the pointers or references I mentioned earlier. When we create a list in Python, there is no information about what will go into that array, which makes it hard to allocate contiguous memory of the same size.

There are several advantages to having contiguous memory. Since the values are stored beside each other, **accessing the values happens in almost constant time**. This is a characteristic we want to preserve. The way Python gets around this is by allocating contiguous memory and storing in it, not the value we want to store, but a reference or a pointer to the value that's stored somewhere else in memory. Figure 2.2.3.