

Data Structures and Algorithms¹

A Study Guide for Students of Sorsogon State
University - Bulan Campus²

JARRIAN VINCE G. GOJAR³

September 2, 2024

¹A course in the Bachelor of Science in Computer Science

²This book is a study guide for students of Sorsogon State University - Bulan Campus taking up the course Data Structures and Algorithms.

³<https://github.com/godkingjay>

Sorsogon State University - Bulan Campus

Contents

Contents	ii
List of Figures	viii
List of Tables	ix
List of Codes	x
1 Introduction to Data Structures and Algorithms	2
1.1 Introduction	2
1.2 Setup and Installation	2
1.2.1 C++ Compiler Installation	2
1.2.1.1 Windows	2
1.2.2 Visual Studio Code Installation	3
1.2.3 Testing the Installation	3
1.3 What are Data Structures?	4
1.4 What are Algorithms?	4
1.5 Why Study Data Structures and Algorithms?	4
1.6 Basic Terminologies	4
1.6.1 Data	4
1.6.2 Data Object	4
1.6.3 Data Type	4
1.6.3.1 Primitive Data Types	5
Integer (int)	5
Character (char)	5
Boolean (bool)	5
Floating-Point (float)	5
Double (double)	6
1.6.3.2 Non-primitive Data Types	6
1.6.4 Abstract Data Type	6
1.6.5 Complexity of an Algorithm	6
1.6.5.1 Time Complexity	6
1.6.5.2 Space Complexity	6
1.7 Asymptotic Notations	6
1.7.1 Big-O Notation	6
1.7.2 Omega Notation	6
1.7.3 Theta Notation	6
1.8 Summary	6
2 Arrays and Linked Lists	7

2.1	Introduction	8
2.2	Arrays	8
2.2.1	Types of Arrays	8
2.2.1.1	One-dimensional Array	8
2.2.1.2	Multi-dimensional Array	8
2.2.2	Array Operations	8
2.2.2.1	Insertion	8
2.2.2.2	Deletion	8
2.2.2.3	Searching	8
2.2.3	Complexity Analysis of Arrays	8
2.3	Linked Lists	8
2.3.1	Types of Linked Lists	8
2.3.1.1	Singly Linked List	8
2.3.1.2	Doubly Linked List	8
2.3.1.3	Circular Linked List	8
2.3.2	Operations on Linked Lists	8
2.3.2.1	Insertion	8
2.3.2.2	Deletion	8
2.3.2.3	Searching	8
2.3.3	Complexity Analysis of Linked Lists	8
2.4	Comparison of Arrays and Linked Lists	8
2.5	Summary	8
3	Stacks and Queues	9
3.1	Introduction	10
3.2	Stacks	10
3.2.1	Operations on Stacks	10
3.2.1.1	Push	10
3.2.1.2	Pop	10
3.2.1.3	Peek	10
3.2.1.4	isEmpty	10
3.2.1.5	isFull	10
3.2.2	Complexity Analysis of Stacks	10
3.2.3	Implementation of Stacks Using Arrays	10
3.2.4	Implementation of Stacks Using Linked Lists	10
3.3	Queues	10
3.3.1	Types of Queues	10
3.3.1.1	Linear Queue	10
3.3.1.2	Circular Queue	10
3.3.1.3	Priority Queue	10
3.3.1.4	Double-ended Queue (Deque)	10
3.3.2	Operations on Queues	10
3.3.2.1	Enqueue	10
3.3.2.2	Dequeue	10
3.3.2.3	Front	10
3.3.2.4	Rear	10
3.3.3	Complexity Analysis of Queues	10
3.3.4	Implementation of Queues Using Arrays	10
3.3.5	Implementation of Queues Using Linked Lists	10
3.4	Comparison of Stacks and Queues	10
3.5	Summary	10

4	Trees	11
4.1	Introduction	12
4.2	Properties of Trees	12
4.2.1	Root Node	12
4.2.2	Parent Node	12
4.2.3	Child Node	12
4.2.4	Leaf Node	12
4.2.5	Ancestors	12
4.2.6	Siblings	12
4.2.7	Descendants	12
4.2.8	Height of a Tree	12
4.2.9	Depth of a Node	12
4.2.10	Degree of a Node	12
4.2.11	Level of a Node	12
4.2.12	Subtree	12
4.3	Types of Trees	12
4.3.1	Binary Tree	12
4.3.1.1	Types of Binary Trees	12
	Left-skewed Binary Tree	12
	Right-skewed Binary Tree	12
	Complete Binary Tree	12
4.3.2	Ternary Tree	12
4.3.3	N-ary Tree	12
4.3.4	Binary Search Tree	12
4.3.5	AVL Tree	12
4.3.6	Red-Black Tree	12
4.4	Basic Operations on Trees	12
4.4.1	Creation of a Tree	12
4.4.2	Insertion	12
4.4.3	Deletion	12
4.4.4	Searching	12
4.4.5	Traversal	12
4.4.5.1	Preorder Traversal	12
4.4.5.2	Inorder Traversal	12
4.4.5.3	Postorder Traversal	12
4.4.5.4	Level-order Traversal	12
4.5	Complexity Analysis of Trees	12
4.6	Summary	12
5	Graphs	13
5.1	Introduction	14
5.2	Properties of Graphs	14
5.2.1	Vertex	14
5.2.2	Edge	14
5.2.3	Degree of a Vertex	14
5.2.4	Path	14
5.3	Types of Graphs	14
5.3.1	Finite Graph	14
5.3.2	Infinite Graph	14
5.3.3	Trivial Graph	14
5.3.4	Simple Graph	14

5.3.5	Multi Graph	14
5.3.6	Null Graph	14
5.3.7	Complete Graph	14
5.3.8	Pseudo Graph	14
5.3.9	Regular Graph	14
5.3.10	Bipartite Graph	14
5.3.11	Labelled Graph	14
5.3.12	Weighted Graph	14
5.3.13	Directed Graph	14
5.3.14	Undirected Graph	14
5.3.15	Connected Graph	14
5.3.16	Disconnected Graph	14
5.3.17	Cyclic Graph	14
5.3.18	Acyclic Graph	14
5.3.19	Directed Acyclic Graph (DAG)	14
5.3.20	Digraph	14
5.3.21	Subgraph	14
5.4	Operations on Graphs	14
5.4.1	Creation of a Graph	14
5.4.2	Insertion	14
5.4.2.1	Insertion of a Vertex	14
5.4.2.2	Insertion of an Edge	14
5.4.3	Deletion	14
5.4.3.1	Deletion of a Vertex	14
5.4.3.2	Deletion of an Edge	14
5.4.4	Traversal	14
5.4.4.1	Depth First Search (DFS)	14
5.4.4.2	Breadth First Search (BFS)	14
5.4.5	Shortest Path	14
5.4.6	Minimum Spanning Tree	14
5.5	Complexity Analysis of Graphs	14
5.6	Summary	14
6	Sorting and Searching	15
6.1	Introduction	16
6.2	Sorting	16
6.2.1	Types of Sorting Algorithms	16
6.2.1.1	Bubble Sort	16
6.2.1.2	Selection Sort	16
6.2.1.3	Insertion Sort	16
6.2.1.4	Merge Sort	16
6.2.1.5	Quick Sort	16
6.2.1.6	Heap Sort	16
6.2.1.7	Radix Sort	16
6.2.1.8	Counting Sort	16
6.2.1.9	Bucket Sort	16
6.2.2	Comparison of Sorting Algorithms	16
6.3	Searching	16
6.3.1	Types of Searching Algorithms	16
6.3.1.1	Linear Search	16
6.3.1.2	Binary Search	16

6.3.1.3	Jump Search	16
6.3.1.4	Interpolation Search	16
6.3.1.5	Exponential Search	16
6.3.1.6	Fibonacci Search	16
6.3.1.7	Ternary Search	16
6.3.2	Comparison of Searching Algorithms	16
6.4	Summary	16
7	Hashing	17
7.1	Introduction	17
7.2	Hash Table	17
7.3	Hash Function	17
7.4	Collision Resolution Techniques	17
7.4.1	Separate Chaining	17
7.4.2	Open Addressing	17
7.4.2.1	Linear Probing	17
7.4.2.2	Quadratic Probing	17
7.4.2.3	Double Hashing	17
7.5	Complexity Analysis of Hashing	17
7.6	Summary	17
8	Advanced Data Structures and Algorithms	18
8.1	Introduction	19
8.2	Advanced Data Structures	19
8.2.1	Segment Tree	19
8.2.2	Fenwick Tree	19
8.2.3	Suffix Tree	19
8.2.4	Suffix Array	19
8.2.5	Trie	19
8.2.6	Heap	19
8.2.7	Disjoint Set	19
8.2.8	Skip List	19
8.2.9	Splay Tree	19
8.2.10	Bloom Filter	19
8.2.11	KD Tree	19
8.2.12	Quad Tree	19
8.2.13	Octree	19
8.2.14	B-Tree	19
8.2.15	B+ Tree	19
8.2.16	R-Tree	19
8.2.17	X-Tree	19
8.2.18	Y-Tree	19
8.2.19	Z-Tree	19
8.3	Advanced Algorithms	19
8.3.1	Dynamic Programming	19
8.3.2	Greedy Algorithms	19
8.3.3	Backtracking	19
8.3.4	Divide and Conquer	19
8.3.5	Branch and Bound	19
8.3.6	Randomized Algorithms	19
8.3.7	Approximation Algorithms	19
8.3.8	String Matching Algorithms	19

8.3.9	Pattern Searching Algorithms	19
8.3.10	Cryptography Algorithms	19
8.3.11	Geometric Algorithms	19
8.3.12	Graph Algorithms	19
8.3.13	Network Flow Algorithms	19
8.3.14	Game Theory Algorithms	19
8.3.15	Quantum Algorithms	19
8.4	Summary	19
9	Applications of Data Structures and Algorithms	20
9.1	Applications in Computer Science	21
9.1.1	Operating Systems	21
9.1.2	Database Management Systems	21
9.1.3	Compiler Design	21
9.1.4	Networking	21
9.1.5	Artificial Intelligence	21
9.1.6	Machine Learning	21
9.1.7	Computer Graphics	21
9.1.8	Computer Vision	21
9.1.9	Robotics	21
9.1.10	Web Development	21
9.1.11	Mobile Development	21
9.1.12	Game Development	21
9.1.13	Cybersecurity	21
9.1.14	Quantum Computing	21
9.2	Applications in Real Life	21
9.2.1	Social Media	21
9.2.2	E-commerce	21
9.2.3	Healthcare	21
9.2.4	Finance	21
9.2.5	Transportation	21
9.2.6	Education	21
9.2.7	Agriculture	21
9.2.8	Manufacturing	21
9.2.9	Entertainment	21
9.2.10	Sports	21
9.2.11	Travel	21
9.2.12	Telecommunications	21
9.2.13	Energy	21
9.2.14	Environment	21
9.2.15	Politics	21
9.2.16	Military	21
9.3	Summary	21
10	References	22

List of Figures

List of Tables

List of Codes

1.1	Hello World Program	3
1.2	Compiling the Program	3
1.3	Running the Program	3
1.4	Integer Data Type	5
1.5	Character Data Type	5
1.6	Boolean Data Type	5
1.7	Floating-Point Data Type	5
1.8	Double Data Type	6

Preface

“Bad programmers worry about the code. Good programmers worry about data structures and their relationships.”

– Linus Torvalds

Jarrian Vince G. Gojar

<https://github.com/godkingjay>

1

Introduction to Data Structures and Algorithms

1.1 Introduction

Data structures and algorithms are one of the fundamental components of computer science. They are essential for solving complex problems efficiently and effectively. Data structures are used to store and organize data in a computer so that it can be accessed and manipulated efficiently. Algorithms are step-by-step procedures or formulas for solving a problem. They are the instructions that tell a computer how to perform a task.

In this course, we will learn about the fundamental data structures and algorithms that are used in computers. We will study how to design, implement, and analyze data structures and algorithms to solve real-world problems. By the end of this course, you will have a solid foundation in data structures and algorithms that will help you become a better programmer and problem solver.

1.2 Setup and Installation

In this course, we will be using the C++ programming language to implement data structures and algorithms. C++ is a powerful and versatile programming language that is widely used in the field of computer science. To get started, you will need to install a C++ compiler and an integrated development environment (IDE) on your computer.

1.2.1 C++ Compiler Installation

The first step is to install a C++ compiler on your computer. A compiler is a program that translates source code written in a programming language into machine code that can be executed by a computer. There are several C++ compilers available, but we recommend using the GNU Compiler Collection (GCC) which is a free and open-source compiler that supports multiple programming languages including C++.

1.2.1.1 Windows

To install GCC on Windows, you can use the MinGW (Minimalist GNU for Windows) project which provides a port of GCC to Windows. You can download the MinGW installer from the MinGW website and follow the installation instructions. You can install MinGW

by following the instructions here: https://code.visualstudio.com/docs/languages/cpp#_example-install-mingwx64-on-windows

1.2.2 Visual Studio Code Installation

The next step is to install an integrated development environment (IDE) on your computer. An IDE is a software application that provides comprehensive facilities to computer programmers for software development. We recommend using Visual Studio Code which is a free and open-source IDE developed by Microsoft. You can download Visual Studio Code from the official website and follow the installation instructions: <https://code.visualstudio.com/Download>

Other than Visual Studio Code, you also need to install the C/C++ extension for Visual Studio Code. You can install the C/C++ extension by following the instructions here: <https://code.visualstudio.com/docs/languages/cpp>

1.2.3 Testing the Installation

To test if the installation was successful, you can create a simple C++ program and compile it using the C++ compiler. Open Visual Studio Code and create a new file with the following C++ code:

```
1 #include <iostream>
2 namespace std;
3
4 int main() {
5     cout << "Hello, World!" << endl;
6     return 0;
7 }
```

Code 1.1: Hello World Program

Save the file with a .cpp extension (e.g., hello.cpp) and open a terminal window in Visual Studio Code. Compile the program using the following command:

```
1 g++ hello.cpp -o hello
```

Code 1.2: Compiling the Program

If there are no errors, you can run the program by executing the following command:

```
1 ./hello
```

Code 1.3: Running the Program

If everything is set up correctly, you should see the output "Hello, World!" printed on the screen.

1.3 What are Data Structures?

A **data structure** is a way of organizing and storing data in a computer so that it can be accessed and manipulated efficiently. Data structures provide a way to manage large amounts of data effectively for various applications. They define the relationship between the data, and the operations that can be performed on the data. There are many different types of data structures that are used in computer science, each with its own strengths and weaknesses. The use of the right data structure can significantly improve the performance of an algorithm and make it more efficient.

1.4 What are Algorithms?

An **algorithm** is a step-by-step procedure or formula for solving a problem. It is a sequence of well-defined instructions that take some input and produce an output. Algorithms are used to solve complex problems and perform various tasks efficiently. They are the instructions that tell a computer how to perform a task. Algorithms are essential for writing computer programs and developing software applications. The efficiency of an algorithm is measured by its time complexity and space complexity.

1.5 Why Study Data Structures and Algorithms?

Data structures and algorithms are essential topics in computer science and software engineering. They are one of the fundamental components of computer science and are used in various applications such as operating systems, database management systems, networking, artificial intelligence, and many others. A good understanding of data structures and algorithms will help you become a better programmer and problem solver. In addition, many companies use data structures and algorithms as part of their technical interviews to assess the problem-solving skills of candidates. Therefore, studying data structures and algorithms is essential for anyone pursuing a career in software engineering or software development.

1.6 Basic Terminologies

Before we dive into the details of data structures and algorithms, let's understand some basic terminologies that might be helpful in understanding the concepts better.

1.6.1 Data

Data is a collection of facts, figures, or information that can be used for analysis or reference. It can be in the form of numbers, text, images, audio, video, or any other format. Data is the raw material that is processed by a computer to produce meaningful information.

1.6.2 Data Object

A **data object** is an instance of a data structure that contains data along with the operations that can be performed on the data. It is an abstraction of a real-world entity that is represented in a computer program.

1.6.3 Data Type

A **data type** is a classification of data that tells the compiler or interpreter how the programmer intends to use the data. It defines the operations that can be performed on the data, the values that can be stored in the data, and the memory space required to store the data.

1.6.3.1 Primitive Data Types

Primitive data types are the basic data types that are built into the programming language. They are used to store simple values such as integers, floating-point numbers, characters, and booleans. Examples of primitive data types include `int`, `float`, `char`, and `bool`. The following are the common primitive data types used in programming:

Integer (`int`)

The integer data type is used to store whole numbers without any decimal points. It can be either signed or unsigned, depending on whether it can store negative values or not. An integer's value can range from -2,147,483,648 to 2,147,483,647 and takes 4 bytes of memory.

```
1 int x = 10;
```

Code 1.4: Integer Data Type

Character (`char`)

The character data type is used to store a single character such as a letter, digit, or special symbol. It is represented by a single byte of memory. A `char` value can range from -128 to 127 or 0 to 255, depending on whether it is signed or unsigned. These values are represented using ASCII codes.

```
1 char c = 'A';
```

Code 1.5: Character Data Type

Boolean (`bool`)

The boolean data type is used to store true or false values. It is represented by a single byte of memory. A `bool` value can be either true or false.

```
1 bool flag = true;
```

Code 1.6: Boolean Data Type

Floating-Point (`float`)

The floating-point data type is used to store real numbers with decimal points. It can represent both integer and fractional parts of a number. It can be either single precision or double precision, depending on the number of bits used to store the value. A `float` value can range from 1.2E-38 to 3.4E+38 and takes 4 bytes of memory.

```
1 float y = 3.14;
```

Code 1.7: Floating-Point Data Type

Double (double)

The double data type is used to store real numbers with double precision. It can represent both integer and fractional parts of a number with higher precision than the float data type. A double value can range from $2.3\text{E-}308$ to $1.7\text{E+}308$ and takes 8 bytes of memory.

```
1 double z = 3.14159;
```

Code 1.8: Double Data Type

1.6.3.2 Non-primitive Data Types**1.6.4 Abstract Data Type****1.6.5 Complexity of an Algorithm****1.6.5.1 Time Complexity****1.6.5.2 Space Complexity****1.7 Asymptotic Notations****1.7.1 Big-O Notation****1.7.2 Omega Notation****1.7.3 Theta Notation****1.8 Summary**

2

Arrays and Linked Lists

2.1 Introduction

2.2 Arrays

2.2.1 Types of Arrays

2.2.1.1 One-dimensional Array

2.2.1.2 Multi-dimensional Array

2.2.2 Array Operations

2.2.2.1 Insertion

2.2.2.2 Deletion

2.2.2.3 Searching

2.2.3 Complexity Analysis of Arrays

2.3 Linked Lists

2.3.1 Types of Linked Lists

2.3.1.1 Singly Linked List

2.3.1.2 Doubly Linked List

2.3.1.3 Circular Linked List

2.3.2 Operations on Linked Lists

2.3.2.1 Insertion

2.3.2.2 Deletion

2.3.2.3 Searching

2.3.3 Complexity Analysis of Linked Lists

2.4 Comparison of Arrays and Linked Lists

2.5 Summary

3

Stacks and Queues

3.1 Introduction

3.2 Stacks

3.2.1 Operations on Stacks

3.2.1.1 Push

3.2.1.2 Pop

3.2.1.3 Peek

3.2.1.4 isEmpty

3.2.1.5 isFull

3.2.2 Complexity Analysis of Stacks

3.2.3 Implementation of Stacks Using Arrays

3.2.4 Implementation of Stacks Using Linked Lists

3.3 Queues

3.3.1 Types of Queues

3.3.1.1 Linear Queue

3.3.1.2 Circular Queue

3.3.1.3 Priority Queue

3.3.1.4 Double-ended Queue (Deque)

3.3.2 Operations on Queues

3.3.2.1 Enqueue

3.3.2.2 Dequeue

3.3.2.3 Front

3.3.2.4 Rear

3.3.3 Complexity Analysis of Queues

3.3.4 Implementation of Queues Using Arrays

3.3.5 Implementation of Queues Using Linked Lists

3.4 Comparison of Stacks and Queues

4

Trees

4.1 Introduction

4.2 Properties of Trees

4.2.1 Root Node

4.2.2 Parent Node

4.2.3 Child Node

4.2.4 Leaf Node

4.2.5 Ancestors

4.2.6 Siblings

4.2.7 Descendants

4.2.8 Height of a Tree

4.2.9 Depth of a Node

4.2.10 Degree of a Node

4.2.11 Level of a Node

4.2.12 Subtree

4.3 Types of Trees

4.3.1 Binary Tree

4.3.1.1 Types of Binary Trees

Left-skewed Binary Tree

Right-skewed Binary Tree

Complete Binary Tree

4.3.2 Ternary Tree

4.3.3 N-ary Tree

4.3.4 Binary Search Tree

4.3.5 AVL Tree

4.3.6 Red-Black Tree

4.4 Basic Operations on Trees

5

Graphs

5.1 Introduction

5.2 Properties of Graphs

5.2.1 Vertex

5.2.2 Edge

5.2.3 Degree of a Vertex

5.2.4 Path

5.3 Types of Graphs

5.3.1 Finite Graph

5.3.2 Infinite Graph

5.3.3 Trivial Graph

5.3.4 Simple Graph

5.3.5 Multi Graph

5.3.6 Null Graph

5.3.7 Complete Graph

5.3.8 Pseudo Graph

5.3.9 Regular Graph

5.3.10 Bipartite Graph

5.3.11 Labelled Graph

5.3.12 Weighted Graph

5.3.13 Directed Graph

5.3.14 Undirected Graph

5.3.15 Connected Graph

5.3.16 Disconnected Graph

5.3.17 Cyclic Graph

5.3.18 Acyclic Graph

5.3.19 Directed Acyclic Graph (DAG)

6

Sorting and Searching

6.1 Introduction

6.2 Sorting

6.2.1 Types of Sorting Algorithms

6.2.1.1 Bubble Sort

6.2.1.2 Selection Sort

6.2.1.3 Insertion Sort

6.2.1.4 Merge Sort

6.2.1.5 Quick Sort

6.2.1.6 Heap Sort

6.2.1.7 Radix Sort

6.2.1.8 Counting Sort

6.2.1.9 Bucket Sort

6.2.2 Comparison of Sorting Algorithms

6.3 Searching

6.3.1 Types of Searching Algorithms

6.3.1.1 Linear Search

6.3.1.2 Binary Search

6.3.1.3 Jump Search

6.3.1.4 Interpolation Search

6.3.1.5 Exponential Search

6.3.1.6 Fibonacci Search

6.3.1.7 Ternary Search

6.3.2 Comparison of Searching Algorithms

6.4 Summary

7

Hashing

7.1 Introduction

7.2 Hash Table

7.3 Hash Function

7.4 Collision Resolution Techniques

7.4.1 Separate Chaining

7.4.2 Open Addressing

7.4.2.1 Linear Probing

7.4.2.2 Quadratic Probing

7.4.2.3 Double Hashing

7.5 Complexity Analysis of Hashing

7.6 Summary

8

Advanced Data Structures and Algorithms

8.1 Introduction

8.2 Advanced Data Structures

8.2.1 Segment Tree

8.2.2 Fenwick Tree

8.2.3 Suffix Tree

8.2.4 Suffix Array

8.2.5 Trie

8.2.6 Heap

8.2.7 Disjoint Set

8.2.8 Skip List

8.2.9 Splay Tree

8.2.10 Bloom Filter

8.2.11 KD Tree

8.2.12 Quad Tree

8.2.13 Octree

8.2.14 B-Tree

8.2.15 B+ Tree

8.2.16 R-Tree

8.2.17 X-Tree

8.2.18 Y-Tree

8.2.19 Z-Tree

8.3 Advanced Algorithms

8.3.1 Dynamic Programming

8.3.2 Greedy Algorithms

9

Applications of Data Structures and Algorithms

9.1 Applications in Computer Science

9.1.1 Operating Systems

9.1.2 Database Management Systems

9.1.3 Compiler Design

9.1.4 Networking

9.1.5 Artificial Intelligence

9.1.6 Machine Learning

9.1.7 Computer Graphics

9.1.8 Computer Vision

9.1.9 Robotics

9.1.10 Web Development

9.1.11 Mobile Development

9.1.12 Game Development

9.1.13 Cybersecurity

9.1.14 Quantum Computing

9.2 Applications in Real Life

9.2.1 Social Media

9.2.2 E-commerce

9.2.3 Healthcare

9.2.4 Finance

9.2.5 Transportation

9.2.6 Education

9.2.7 Agriculture

9.2.8 Manufacturing

9.2.9 Entertainment

References

A. Books

- Vishwas R. (2023). Data Structure Handbook. Dr. Vishwas Raval. ISBN: 978-9359063591
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to algorithms. MIT press. ISBN: 978-0262046305
- Erickson, J. (2019). Algorithms. ISBN: 978-1792644832

B. Other Sources

- Tutorialspoint. (n.d.). Data Structures Basics. Data Structure Basics. https://www.tutorialspoint.com/data_structures_algorithms/data_structures_basics.htm
- Algorithm Archive · Arcane Algorithm Archive. (n.d.). <https://www.algorithm-archive.org/>