# Data Structures and Algorithms in Java

Parth Sinha
Student of AI & ML
Oxford Brookes University
Email: sinhaparth555@gmail.com



***Abstract*—This paper presents a comprehensive study guide for Data Structures and Algorithms (DSA) implementation in Java, motivated by the educational content provided through FreeCodeCamp's online learning platform. The study focuses on fundamental DSA concepts, their practical implementation, and real-world applications using Java programming language. Drawing inspiration from FreeCodeCamp's open-source educational resources, particularly their comprehensive video tutorial on DSA, this work aims to bridge the gap between theoretical understanding and practical implementation of key programming concepts. The paper systematically covers essential data structures including arrays, linked lists, trees, and graphs, along with fundamental algorithms for sorting, searching, and graph traversal. Special emphasis is placed on time and space complexity analysis, providing learners with a solid foundation in algorithmic efficiency. This resource serves as both a learning tool for beginners and a reference guide for experienced programmers seeking to strengthen their DSA knowledge in Java.**

***Index Terms*—Data Structures, Algorithms, Java Programming, Educational Computing, FreeCodeCamp, Online Learning**

## I. Introduction

### A. Data Structure

A data structure is a specialized format for organizing, processing, retrieving, and storing data. It's like a blueprint that dictates how data elements are arranged and connected[1], making it easier to work with and manipulate data.

*Types of data structure*

*1) Linear:* Data elements are arranged in a sequential or linear order, one after another. Each element is connected to its previous and next adjacent elements.[2]

Types Linear data structure:

- Array
- Linked List
- Stack
- Queue

*2) Non-linear:* Data elements are not arranged in a linear order but rather in a hierarchical or networked manner.[2]

Types of Non-Linear data structure:

- Tree
- Graph

### B. Algorithms

An algorithm is a precise sequence of instructions designed to solve a specific problem or perform a computation. It's like a recipe, but for solving problems rather than cooking.

Let say you want to prepare a tea. So, the steps would be -

- Boil water.
- Put tea in tea pot
- Add hot water
- Put hot tea into cups.
- Do you need sugar?
  - If yes, put it into tea cups
  - If no, do nothing
- Stir, drink and enjoy

Lets take another example, we want to print average of 3 given numbers. Lets say you want to write algorithm for it. So, the steps would be -

- Perform sum of 3 numbers.
- Store it in a variable sum.
- Divide the sum by 3.
- Store the value in variable avg.
- Print the value stored in avg.

Here's an example of how Java code will be displayed:

```
public void findAvg(int a, int b, int c) {
    int sum = a + b + c;
    int avg = sum/3;
    System.out.println(avg);
}
```

Listing 1: Avg of 3 numbers

*1) Analysis of Algorithms:*

- An Algorithm is a set of instructions to perform a task or to solve a given problem.
- There are several different algorithms to solve a given problems.
- Analysis of algorithm deals in finding best algorithm which runs fast and takes in less memory.
- For example -
  - Find sum of first $n$ natural numbers.
    a) Input :- $n = 4$
    Output :- 10 $i.e.$ $(1 + 2 + 3 + 4)$
  - b) Input :- $n = 5$
    Output :- 15 $i.e$ $(1 + 2 + 3 + 4 + 5)$

Let's see two different approach to find to sum of $n$ number, See Listing 2 and 3.

```
    public int findSum(int n) {
        return n * (n + 1) / 2;
    }
```

Listing 2: Sum of $n$ numbers

```java
public int findSum(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum = sum + i;
    }
    return sum;
}
```

Listing 3: Sum of $n$ numbers 2

## II. Time Complexity

Time complexity is a fundamental concept in computer science that quantifies the amount of time it takes an algorithm to run as a function of the size of its input. It's a crucial metric for evaluating the efficiency of algorithms, especially as input sizes grow.

Let's take the code snippet Listing 2 and 3, and I tried to run them by adding timer and check which one runs faster, ***Important** *The timer I used is not the correct way to use cause there are different factors influence the timing of running the code, but for the just an example, I have used it to show, what* ***Time Complexity*** is. So the result for listing 1 takes 0.0 $millisecs$ and listing 2 takes 1.0 $millisecs$, so allegedly the code snippet listing 1 is faster than listing 2.

## III. Space Complexity

Space Complexity in coding refers to the amount of memory space that an algorithm uses during its execution. It's a crucial aspect of algorithm analysis, alongside time complexity.

## IV. Asymptotic Analysis

Asymptotic analysis is a crucial tool in computer science for evaluating the efficiency of algorithms. It helps us understand how the running time or space complexity of an algorithm changes as the input size grows. By focusing on the limiting behavior of the algorithm, we can make informed decisions about which algorithms to choose for a given problem.

### A. Asymptotic Notation

Asymptotic notation is a powerful tool used in computer science to analyze the efficiency of algorithms. It allows us to describe the growth rate of an algorithm's running time or space usage as the input size increases. By understanding the asymptotic behavior of an algorithm, we can make informed decisions about which algorithms to choose for a particular problem.

For example, if a person goes into car showroom and chooses a car and asked the salesperson 'what's the mileage of the car?' and he/she replies with three different values i.e., for Highway (min traffic) - 25 $km/litre$, City (max traffic) - 15 $km/lire$ and City + Highway (avg traffic) - 20 $km/litre$. Now for such scenario Asymptotic Notations help us in such scenario to determine -

- Best Case
- Average Case
- Worst Case

### B. Types of Asymptotic Notations

There are three notations for performing runtime analysis of an algorithm -

- Omega ($\Omega$) Notation
- Big O (O) Notation
- Theta ($\Theta$) Notation

*1) Omega ($\Omega$) Notation:*

- It is the formal way to express the lower bound of algorithm's running time.
- Lower bound means for any given input this notation determines best amount of time an algorithm can take to complete.
- For example -
  If we say certain algorithm takes $100\,\text{secs}$ as best amount of time. So, $100\,\text{secs}$ will be lower bound of that algorithm. The algorithm can take more than $100\,\text{secs}$ but it will not take less than $100\,\text{secs}$.

*2) Big O (O) Notation:*

- It is the formal way to express the upper bound of algorithm's running time.
- Upper bound means for any given input this notation determines longest amount of time an algorithm can take to complete.
- For example -
  If we say certain algorithm takes $100\,\text{secs}$ as longest amount of time. So, $100\,\text{secs}$ will be upper bound of that algorithm. The algorithm can take less than $100\,\text{secs}$ but it will not take more than $100\,\text{secs}$.

*3) Theta ($\Theta$) Notation:*

- It is the formal way to express both the upper and lower bound of an algorithm;s running time.
- By lower and upper bound means for any given input this notation determines average amount of time an algorithm cna take to complete.
- For example -
  If we run certain algorithm and it takes $100\,\text{secs}$ for first run, $120\,\text{secs}$ for second run, $110\,\text{secs}$ for third run and so on. So, theta notations gives an average of running time of that algorithm.

### C. Analysis of Time Complexity (Big O Notation)

It is the formal wau to express the upper bound of an algorithm's running time. Upper bound means for any given input this notation determines longest amount of time an algorithm can take to complete.

*1) Rules of Big O Notation:* Lets say we are given a machine and it follows -

- It's a single processor
- It performs Sequential Execution of statements
- Assignment operation takes 1 unit of time
- Return statement takes in 1 unit of time
- Arithmetical operation takes 1 unit of time
- Logical operation takes 1 unit of time
- Other small/single operations takes 1 unit of time

TABLE I: Time Complexity of the code

| Line no. | Operations | Unit time |
|---|---|---|
| 2 | 1 + 1 + 1 + 1 | 4 |
| 3 | 1 + 1 | 2 |

- Drop lower order terms $T = n^2 + 3n + 1 \Rightarrow O(n^2)$
- Drop constant multipliers $T = 3n^2 + 6n + 1 \Rightarrow O(n^2)$

*2) Calculating Time Complexity of Constant Algorithm:* Let's take the code snippet given below.

```
public int sum(int x, int y) {
    int result = x + y;
    reutrn result;
}
```

Listing 4: Sum of two number

As you can see the table I total time complexity is:

$$T = 4 + 2 = 6$$

$$T \approx C \ (constant)$$

```
public int get(int[] arr, int i) {
    return arr[i];
}
```

Listing 5: Array

Fig 1 refers to the constant of time regardless of the size of the input.

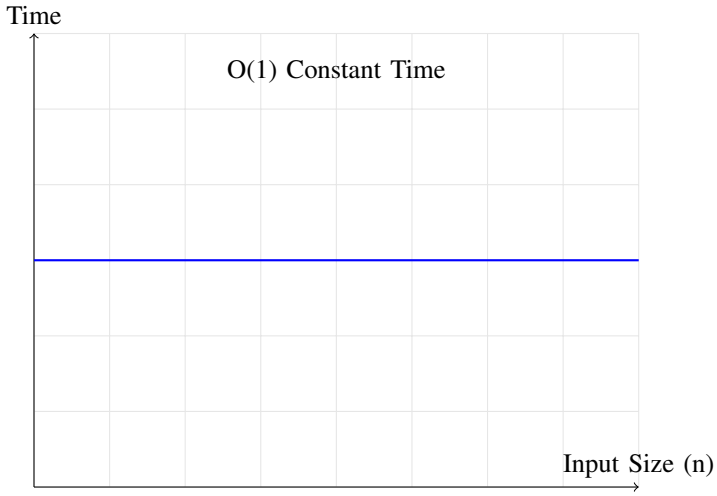


Fig. 1: Constant time complexity showing execution time remains constant regardless of input size

## REFERENCES

[1] mahdi, "Data structures: a comprehensive introduction," Apr 2024. [Online]. Available: https://dev.to/m__mdy__m/data-structures-a-comprehensive-introduction-2o13

[2] A. Biswal, "What is data structure: Types, classifications and applications," May 2021. [Online]. Available: https://www.simplilearn.com/tutorials/data-structure-tutorial/what-is-data-structure