

Backend

Arland Barrera

September 6, 2025

You see here kid? You gotta just go for it; don't think about what comes after or what came before. You just gotta bend your knees, take a deep breath, and jump. And you might think; what if I fall? Well, what if you don't? what if you fly?

---

# Contents

---

<b>1</b>	<b>Algorithms</b>	<b>4</b>
1.1	XOR Swap . . . . .	4
1.2	Sum of Arithmetic Series . . . . .	5
1.3	Search . . . . .	6
1.3.1	Linear Search . . . . .	6
1.3.1.1	Time and Space Complexity . . . . .	6
1.3.1.2	Advantages and Disadvantages . . . . .	7
1.3.2	Binary Search . . . . .	7
1.3.3	Hash Table Search . . . . .	7
1.4	Sort . . . . .	7
1.4.1	Bubble Sort . . . . .	7
1.4.2	Selection Sort . . . . .	7
1.4.3	Insertion Sort . . . . .	7
1.4.4	Quick Sort . . . . .	7
1.4.5	Merge Sort . . . . .	7
1.4.6	Tim Sort . . . . .	7
<b>2</b>	<b>Protocols</b>	<b>8</b>
2.1	IP . . . . .	8
2.1.1	Public IP Adress . . . . .	8
2.1.2	Private IP Adress . . . . .	8
2.2	TCP . . . . .	8
2.3	TLS . . . . .	8
2.4	UDP . . . . .	8
2.5	DNS . . . . .	8

---

# Algorithms

---

## 1.1 XOR Swap

Given two values **a** and **b**, they can be swapped without the need of an temporary variable using the **xor**, exclusive or, operator. This works by changing the bits of the values. The caret symbol ‘^’ is the most common operator for the XOR operation in many programming languages like c, c++, Java and Javascript .The process is the following:

Listing 1.1: XOR swap

```
1 function xorSwap(a, b) {  
2   if (a == b) return;  
3   a = a ^ b;  
4   b = a ^ b;  
5   a = a ^ b;  
6 }
```

XOR only returns true (*1*) if the compared values are in an *or* state, otherwise returns false (*0*).

There are three steps that involve the operation between **a** and **b** using **xor**. In the first step the result is stored in *a*, then in *b* in the second and lastly in the third in *a* again.

This method is used in low level languages such as assembly.

### Example:

*a* = 5 and *b* = 7, in binary *a* = 101 and *b* = 111.

First step, the result is stored in *a*:

*a* = 101

*b* = 111

*a* = 010

Second step, the result is stored in *b*:

*a* = 010

*b* = 111

$$b = 101$$

Third step, the result is stored in  $a$  again:

$$\begin{aligned} a &= 010 \\ b &= 101 \end{aligned}$$

$$a = 111$$

$a = 111$  and  $b = 101$ , in other terms  $a = 7$  and  $b = 5$ . The values have been swaped.

## 1.2 Sum of Arithmetic Series

The sum of an arithmetic series is is given by the expression:

$$S_n = \frac{n}{2} (a_i + a_f)$$

Computers process multiplications faster, so the division  $n/2$  can be replaced by the product  $n * 0.5$ .

**Elements:**

- $S_n$  = sum of series.
- $n$  = number of terms.
- $a_i$  = initial term.
- $a_f$  = final term.

To find  $n$ , the formula for the  $n^{th}$  term of an arithmetic progression can be used:

$$a_f = a_i + (n - 1)d$$

Where  $d$  is the step, the standard difference between each term.

By isolating  $n$ , the formula ends up like this:

$$n = \left( \frac{a_f - a_i}{d} \right) + 1$$

The algorithm has the following structure:

```

1 function sumArithmeticSeries(n, ai, af) {
2   return n * 0.5 * (ai + af);
3 }
4
5 // if n is unknown
6 n = ((af - ai) / d) + 1;
```

This algorithm is  $O(1)$ , constant.

**Example:**

The sum of odd numbers (1, 3, 5, 7, ...) between 1 and 100. The range of odd values is 1-99, the first term is 1 and the last is 99. The step between each term is 2, with  $n$  can be found.

$$\begin{aligned} n &= \left( \frac{99 - 1}{2} \right) + 1 \\ n &= \left( \frac{98}{2} \right) + 1 \\ n &= 49 + 1 \\ n &= 50 \end{aligned}$$

The sum of the arithmetic series with  $n = 50$  is:

$$\begin{aligned} S_{50} &= 50 * 0.5 (1 + 99) \\ S_{50} &= 25 * 100 \\ S_{50} &= 2500 \end{aligned}$$

## 1.3 Search

### 1.3.1 Linear Search

Given an array `arr[]` of  $n$  integers, and an integer element  $x$ , find whether element  $x$  is **present** in the array. Return the **index** of the first occurrence of  $x$  in the array, or the invalid index **-1** if it doesn't exist.

Linear search iterates over all the elements of the array one by one and checks if the current element is equal to the target element. It is also known as **sequential search**.

```

1 function linearSearch(arr, len, x) {
2     for (int i = 0; i < len; i++)
3         if (arr[i] == x)
4             return i;
5     return -1;
6 }
7
8 int arr = [2, 5, 65, 12, 84];
9 int len = arr.length;
10 int x = 65;
11 int index = linearSearch(arr, len, x);

```

#### 1.3.1.1 Time and Space Complexity

##### Time Complexity

- **Best case:** the key might be at the first index. The best case is complexity  $O(1)$ .

- **Worst case:** the key might be at the last index. The worst case is complexity  $O(n)$ , where  $n$  is the size of the array.
- **Average case:**  $O(n)$ .

**Auxiliary Space:**  $O(1)$  as except for the variable to iterate through the list, no other variable is used.

#### 1.3.1.2 Advantages and Disadvantages

##### Advantages

- Works on sorted or unsorted arrays. It can be used on arrays of any data type.
- Does not require any additional memory.
- Well-suited for small datasets.

##### Disadvantages

- Time complexity  $O(n)$ , which is slow for large datasets.

#### 1.3.2 Binary Search

#### 1.3.3 Hash Table Search

### 1.4 Sort

#### 1.4.1 Bubble Sort

#### 1.4.2 Selection Sort

#### 1.4.3 Insertion Sort

#### 1.4.4 Quick Sort

#### 1.4.5 Merge Sort

#### 1.4.6 Tim Sort

---

# Protocols

---

A protocol is a set of rules for data communication in a network. It allows devices to send and receive data in packets. Data packets are addressed, routed and sent across networks.

## 2.1 IP

IP stands for *Internet Protocol*.

There are two main versions of the Internet Protocol: the older, but still widely used, **IPv4** and the more recent and scalable **IPv6**.

An IP Address is a unique numerical label assigned to every device, such as a computer, phone or server; that is connected to a computer network or the internet.

When information is sent online, it is broken into smaller pieces called data packets. The IP address acts as the "electronic return address" for these packets, ensuring they arrive at the correct destination.

IP addresses can be public or private.

### 2.1.1 Public IP Address

### 2.1.2 Private IP Address

192.186.0.0

## 2.2 TCP

TCP stands for *Transmission Control Protocol*.

It establishes a connection between the sender and receiver before any data is sent, maintaining this connection until communication is complete.

## 2.3 TLS

## 2.4 UDP

## 2.5 DNS