De La Salle University
DASMARIÑAS

CS Algorithms Demystified:

A Beginner's Guide to Searching, Sorting, and Solving

Christian Elijah DC. Darvin

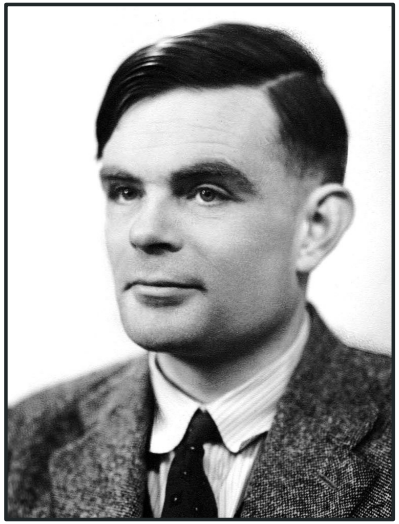
March 25, 2026

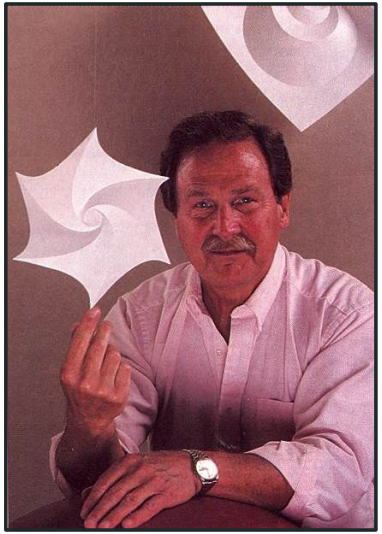
OUTLINE

Chapter 1	Algorithms
Chapter 2	Time & Space Complexity
Chapter 3	Searching & Sorting Algorithms

Chapter 1: Algorithms

HISTORY OF COMPUTING




algorithm noun

al·go·rithm ˈal-gə-ˌri-thəm

: a procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation

broadly : a step-by-step procedure for solving a problem or accomplishing some end

PROPERTIES

Definiteness
Finiteness
Effectiveness

REPRESENTATION

PROBLEM A

PSEUDOCODE

Pseudo code: **Bubble Sort(Array** $a[\ ]$**)**

```
1. begin
2.     for i = 1 to n − 1
3.         for j = 1 to n − i
4.             if (a[j] > a[j + 1]) then
5.                 Swap (a[j], a[j + 1])
8. end
```

PROGRAMMING LANGUAGES

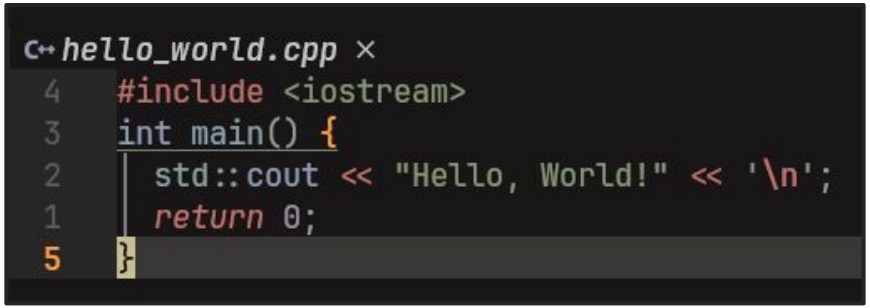
PSEUDOCODE

Pseudo code: **Bubble Sort(Array** $a[\ ]$**)**

```
1. begin
2.    for i = 1 to n − 1
3.       for j = 1 to n − i
4.          if (a[j] > a[j + 1]) then
5.             Swap (a[j], a[j + 1])
8. end
```

- A human-readable, step-by-step description of an algorithm's logic
- Design phase
- “Pseudo”

PROGRAMMING LANGUAGE



- Instructs a computer to perform tasks
- Implementation phase
- Compiler/Interpreter

Chapter 2: Time & Space Complexity

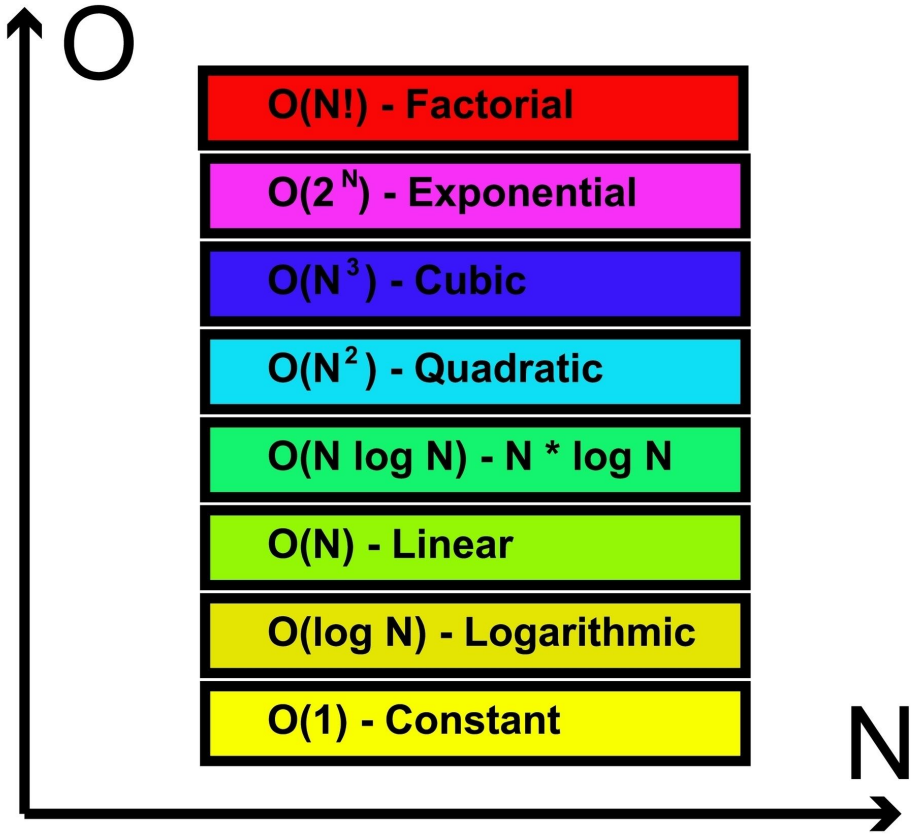

INPUT
ALGORITHM
OUTPUT

TIME

- How fast? (CPU Usage)
- Measured by the number of operations or steps the CPU has to take

SPACE

- How heavy? (RAM Usage)
- Measured by how much extra RAM the program needs to run.



Big O Notation

O(1)	1
O(log n)	6.64
O(n)	100
O(n log n)	664
$O(n^2)$	10,000
$O(n^3)$	1,000,000
$O(2^n)$	1.26×10^{30}
O(n!)	9.33×10^{37}

Bubble Sort

N	Time (s)	Comparisons
1000	0.00438599	498465
5000	0.099257	12495289
10000	0.419583	49992150
20000	2.05291	199966995
50000	14.0708	1249948435
100000	57.7297	4999887519
120000	83.5827	7199767422

Quick Sort

N	Time (s)	Comparisons
1000	9.9383e-05	9904
5000	0.00061064	72178
10000	0.00144613	157158
20000	0.00280229	341807
50000	0.00763538	952103
100000	0.0149471	2054544
120000	0.0175353	2430767

Bubble Sort $O(n^2)$

Input Size (N)	n^2
1,000	1,000,000
5,000	25,000,000
10,000	100,000,000

Quick Sort $O(n * \log_2 n)$

Input Size (N)	$n * \log_2 n$
1,000	≈ 9,970
5,000	≈ 61,450
10,000	≈ 132,900

Time-Space Trade-Off

Is there a universal way to measure the efficiency of algorithms? **YES.**

ASYMPTOTIC NOTATIONS

O (Oh) | **Ω** (Omega) | **Θ** (Theta)

Chapter 3: Searching & Sorting Algorithms

Searching

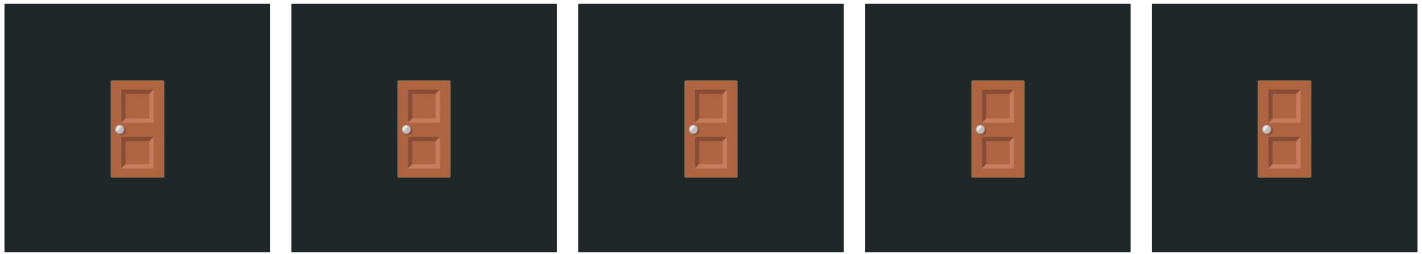
Locating a specific item in a dataset.

Sorting

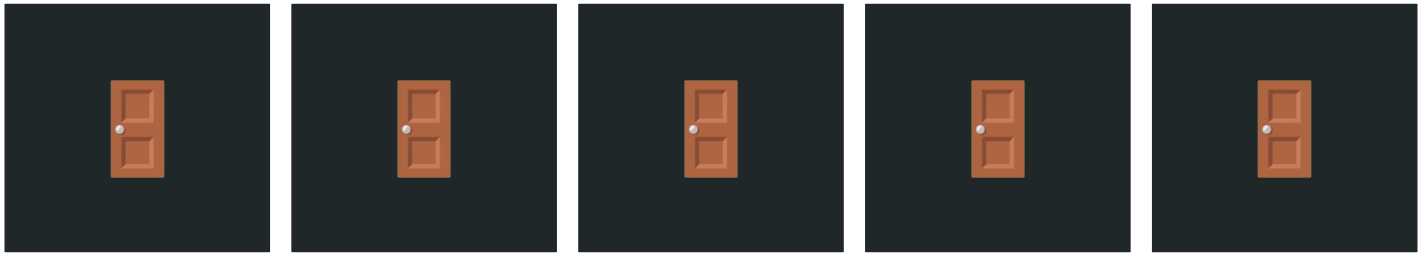
Arranging data into a specific sequence.

HUMAN VIEW

COMPUTER VIEW

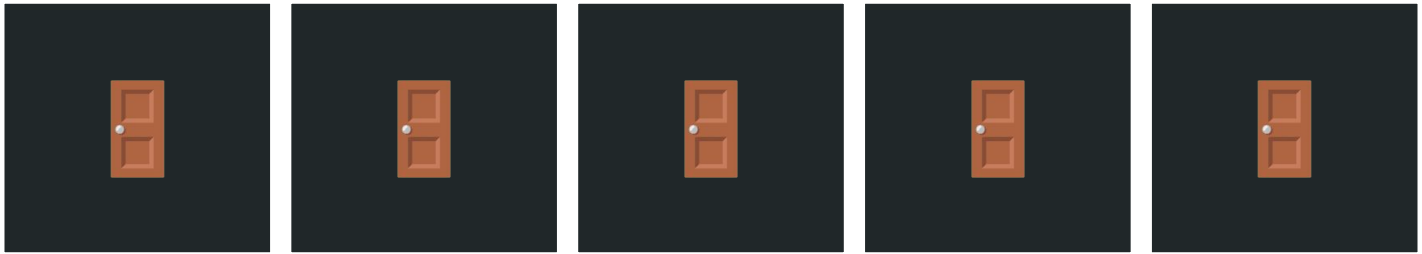


COMPUTER VIEW

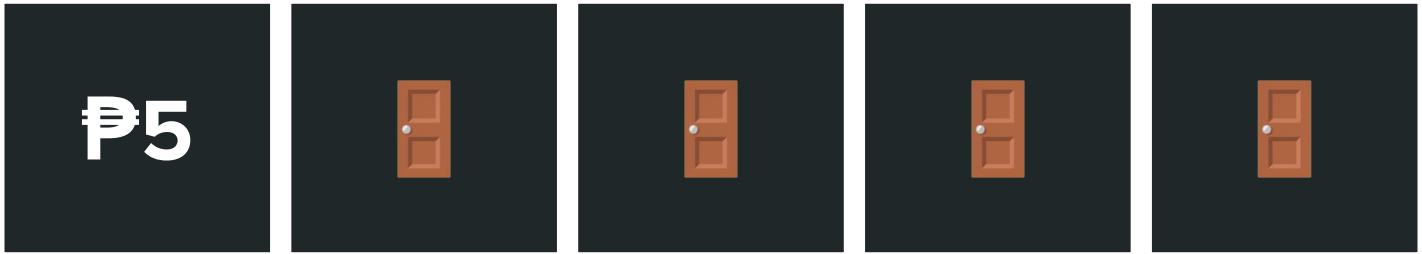


[0] [1] [2] [3] [4]

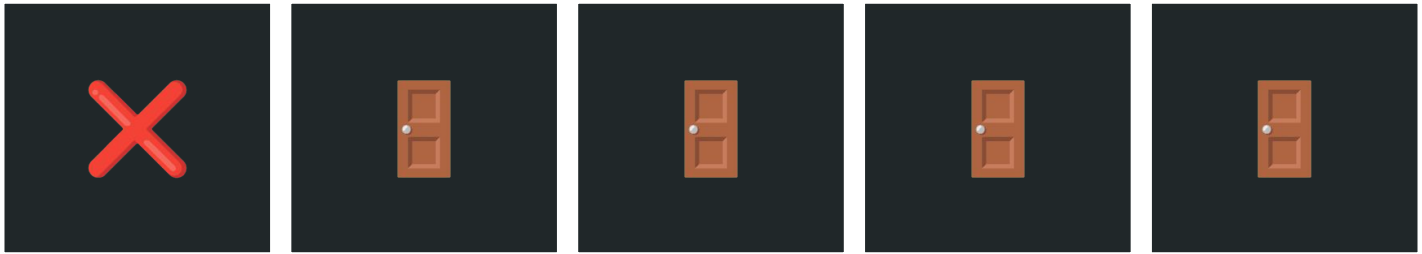
FIND THE ₱100 MONEY



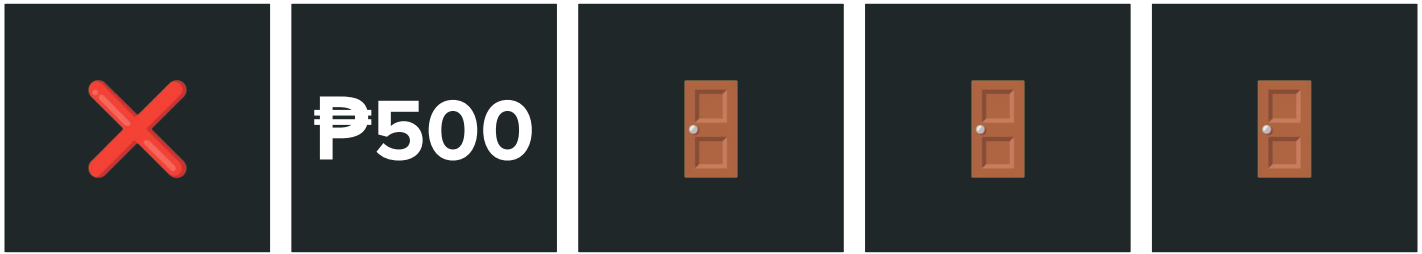
APPROACH A: Start from left
TARGET: ₱100



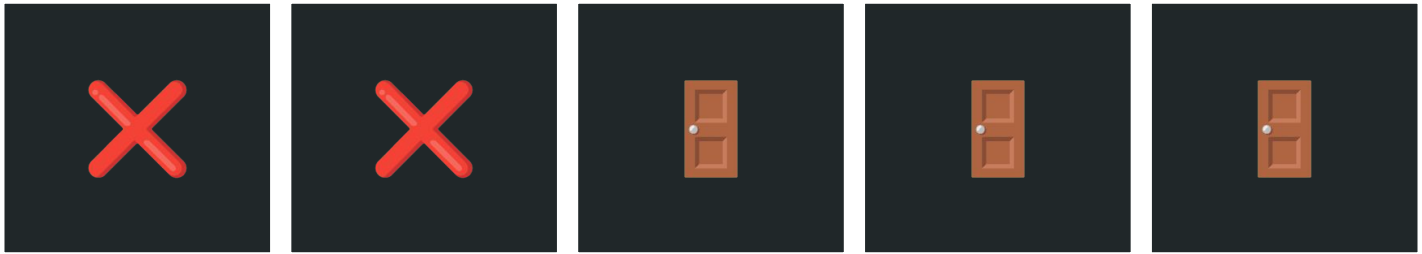
APPROACH A
TARGET: ₱100



APPROACH A
TARGET: ₱100



APPROACH A
TARGET: ₱100



APPROACH A
TARGET: ₱100

APPROACH A
TARGET: ₱100

APPROACH A
TARGET: ₱100

APPROACH A TARGET: ₱100

APPROACH A
TARGET: ₱100

LINEAR SEARCH

The idea of the algorithm is to **iterate** across the array from left to right, searching for a specified element.

LINEAR SEARCH

If the **current** element is equal to the **target** element, return the **index** (position) of that element

SCENARIO A

Target
9

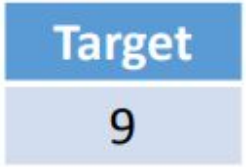
n = 15

Target
9


Target
9
11
23
8
14
30
9
6
17
22
28
25
15
7
10
19

Target
9

Target
9


Target
9


11
23
8
14
30
9
6
17
22
28
25
15
7
10
19


Target
9
11
23
8
14
30
9
6
17
22
28
25
15
7
10
19


Target
9
11
23
8
14
30
9
6
17
22
28
25
15
7
10
19
0
1
2
3
4
5

SCENARIO B

Target
90

11	23	8	14	30	9	6	17	22	28	25	15	7	10	19

SCENARIO B

Element not found!

Best-Case	$\Omega(1)$
Worst-Case	$O(n)$

BUBBLE SORT

The algorithm shifts **higher** values **right** and **lower** values **left**.

BUBBLE SORT

Set swap counter to a non-zero value
Repeat until the swap counter is 0:
    Reset swap counter to 0
    Look at each adjacent pair
        If two adjacent elements are not in order, swap them and add one to the swap counter

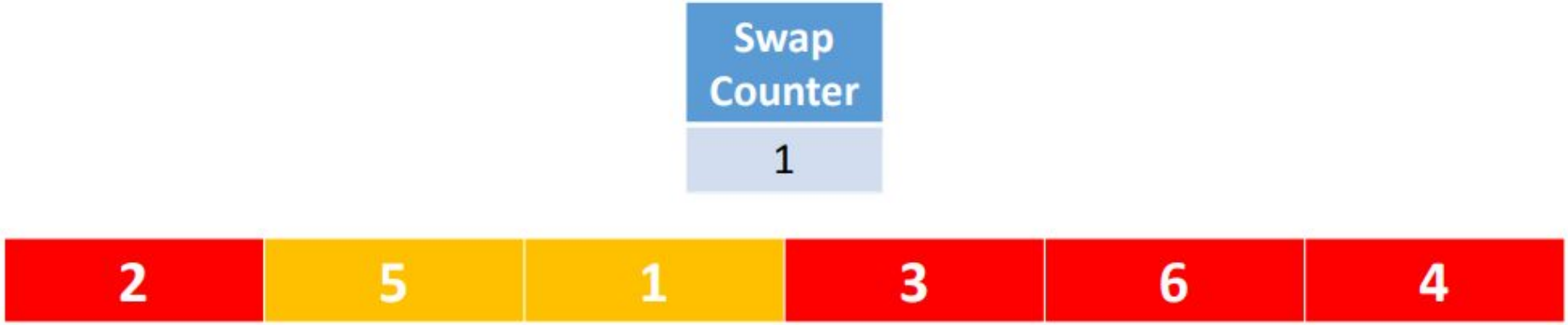
BUBBLE SORT

Swap Counter
0

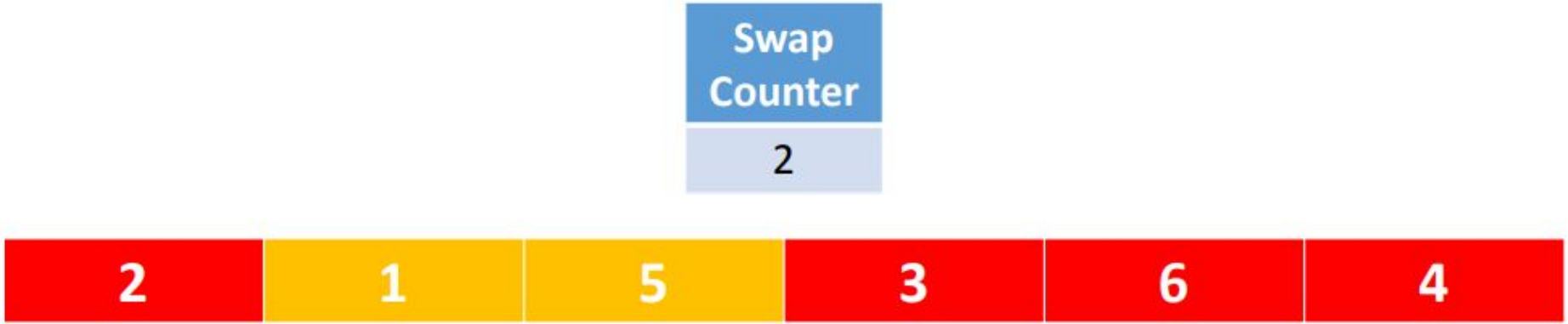
BUBBLE SORT

BUBBLE SORT

BUBBLE SORT

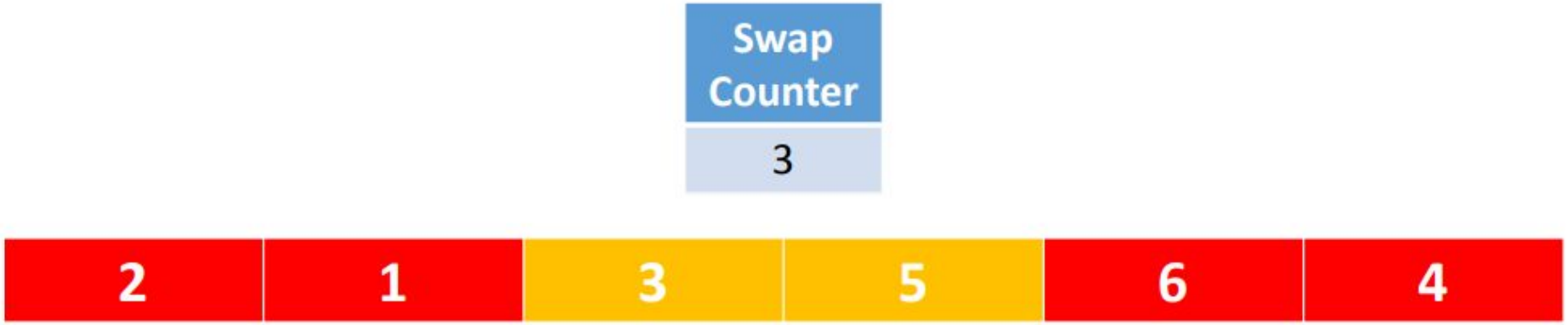


BUBBLE SORT



BUBBLE SORT

BUBBLE SORT



BUBBLE SORT

BUBBLE SORT

BUBBLE SORT

Swap Counter
0

Best-Case	$\Omega(n)$
Worst-Case	$O(n^2)$

The "best" algorithm depends on the problem you're trying to solve.
There is no one-size-fits-all solution.

We can only see a short distance ahead, but we can see plenty there that needs to be done.

— **Alan Turing**, Computing Machinery and Intelligence (1950)