



# Introduction to Sorting

Course on Sorting and Searching

# Agenda

- ✓ 1. What is the sorting problem?
2. How to sort an array in ascending using JAVA?
3. Why should we care about the implementations of the sorting algorithms?
4. Some famous sorting algorithm
5. Learn our first sorting algorithm - Bubble Sort
6. Sorting Terminologies
7. Insertion Sort & Discussion
8. Selection Sort & Discussion
9. Assessment

# What is the sorting problem?

Arranging elements in a specific order.

Commonly used orders are:

- ✓ Ascending order
- ✓ Descending order

Example: Arranging the rank list in decreasing order of marks, Alphabetic ordering.

Target: Sort an array in ascending order

[ - - - ]

[1, 2, 3, 4, 5]

↘  
[2, 4, 1, 3, 5]

Name	Marks
Rohit	0
...	...

$[1, 5, 6, 1] \rightarrow [1, 5, 6, 9]$

How to sort an array in ascending order in Java?

How to sort an array in ascending order in  
Java?

- Arrays.sort()

A[]



The diagram consists of two hand-drawn arrows. One arrow originates from the text 'A[]' and points towards the 'Arrays.sort()' method. The other arrow originates from the 'Arrays.sort()' method and points towards the question 'How to sort an array in ascending order in Java?'.



# Why is it important to learn the implementations of sorting algorithms?

- ✓ Improves our grasp on the programming language.
- ✓ Teaches us various tricks that can be used independently in several CP questions.
- ✓ Allows us to gain a better understanding of the scalability and performance of our code.

## Various types of sorting algorithms:

- ✓ 1. Bubble sort
- ✓ 2. Insertion sort
- ✓ 3. Selection sort
4. Merge sort
5. Quick sort
6. Count sort



etc.

# Our First Sorting Algorithm!



Bubble Sort - The heaviest bubble sinks to the ground first!





[5, 3, 4, 10, 9, 1]

5



3

4

10

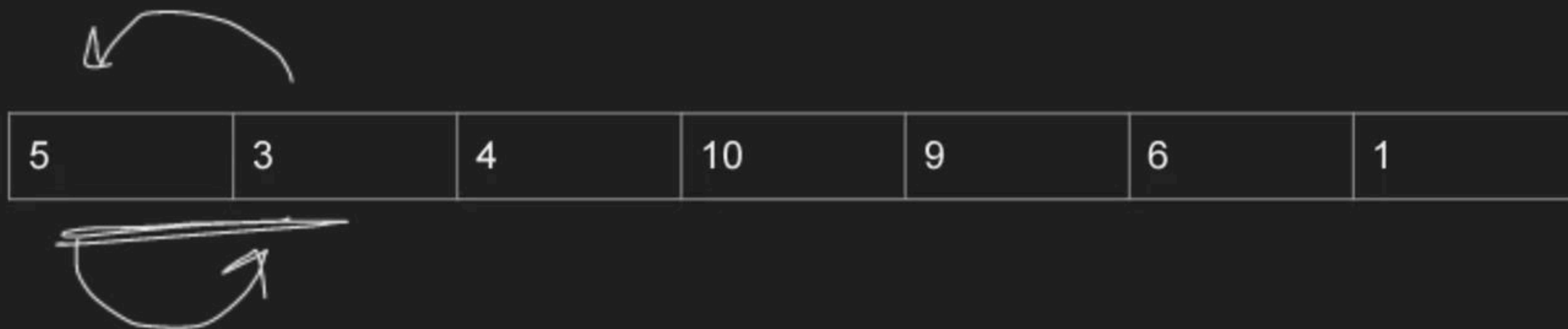


9



1

Go over every adjacent elements in the array and swap them if  $A[j + 1] < A[j]$



Go over every adjacent elements in the array and swap them if  $A[j + 1] < A[j]$

5	3	4	10	9	6	1
---	---	---	----	---	---	---

3	5	4	10	9	6	1
---	---	---	----	---	---	---



Go over every adjacent elements in the array and swap them if  $A[j + 1] < A[j]$

5	3	4	10	9	6	1
---	---	---	----	---	---	---

3	5	4	10	9	6	1
---	---	---	----	---	---	---

3	4	5	10	9	6	1
---	---	---	----	---	---	---

10 > 5





Go over every adjacent elements in the array and swap them if  $A[j + 1] < A[j]$

5	3	4	10	9	6	1
---	---	---	----	---	---	---

3	5	4	10	9	6	1
---	---	---	----	---	---	---

3	4	5	10	9	6	1
---	---	---	----	---	---	---

3	4	5	9	10	6	1
---	---	---	---	----	---	---



Go over every adjacent elements in the array and swap them if  $A[j + 1] < A[j]$

5	3	4	10	9	6	1
---	---	---	----	---	---	---

3	5	4	10	9	6	1
---	---	---	----	---	---	---

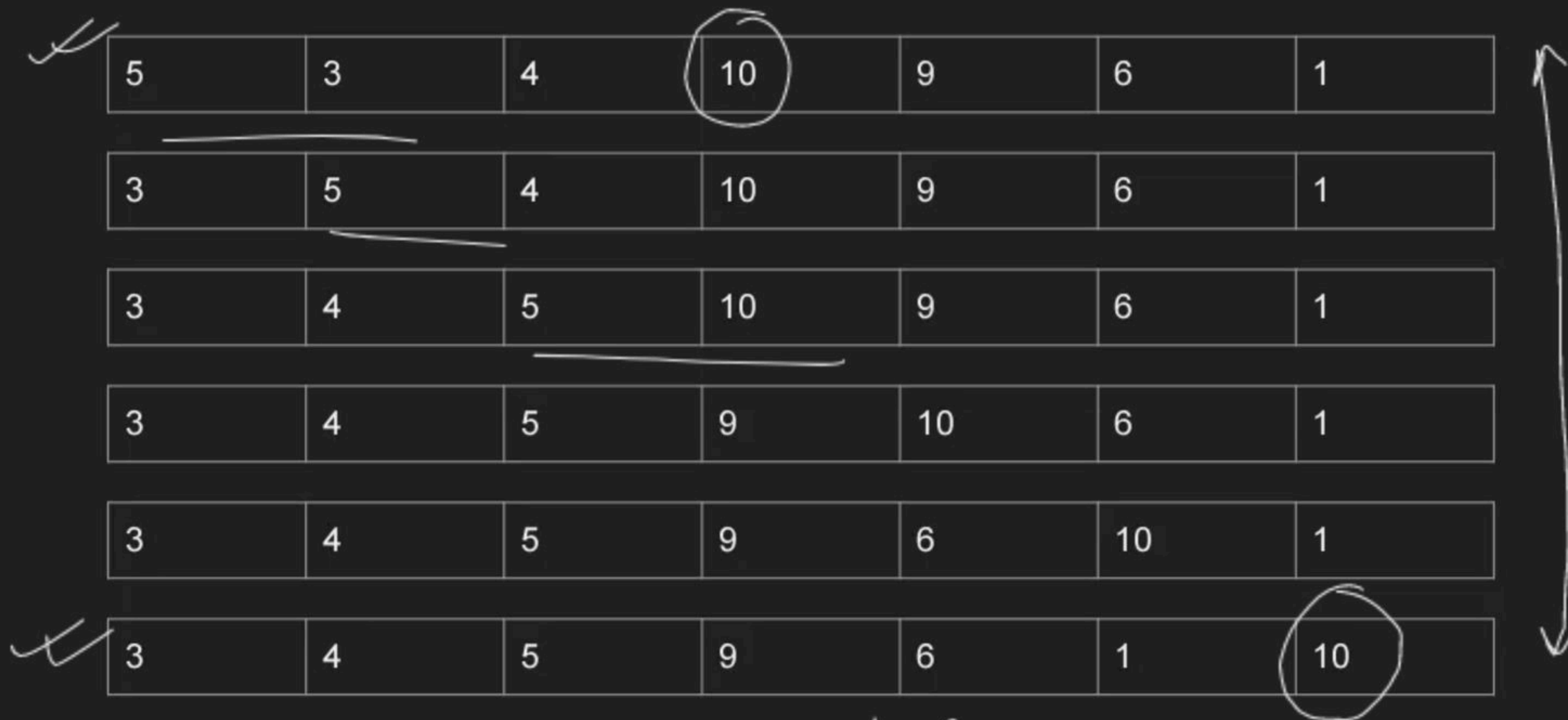
3	4	5	10	9	6	1
---	---	---	----	---	---	---

3	4	5	9	10	6	1
---	---	---	---	----	---	---

3	4	5	9	6	10	1
---	---	---	---	---	----	---



Go over every adjacent elements in the array and swap them if  $A[j + 1] < A[j]$



One pass of Bubble Sort

	5	3	4	10	9	6	1
Pass - 1	3 ✓	4 ✓	5 ✓	<del>9</del> 6	<del>6</del> <del>9</del> 7	<del>1</del> 9	10
Pass - 2	3	4	5	6	1	9	10
Pass - 3	3	4	5	1	6	9	10
Pass - 4	3	4	1	5	6	9	10
Pass - 5	3	1	4	5	6	9	10
Pass - 6	1	3	4	5	6	9	10

$n-i-1$

<  $\frac{n}{2}$

## Time & Space Complexity

Worst Case Time :  $O(n^2)$

Best Case Time :  $O(\underline{n}^2)$

Average :  $O(n^2)$

Space (Extra) :  $O(1)$





## Optimizing Bubble Sort Best Case

[1, 2, 3, 4, 5]

→ Pass - 1

swapped = false

[1, 2, 3, 4, 5]

```
for (pass) {  
  for (adjacent) { }  
}
```

# Sorting Terminologies:

- ✓ Stability
  - In-place algorithms
  - External & Internal sorts

Stability: (Only for duplicates in an array)

$[5, 4, 9, 6, 7, \textcolor{red}{1}, 2, \textcolor{yellow}{1}] \xrightarrow{\text{Sort}()} [1, 1, 2, 4, 5, 6, 7, 9]$

# Stability:

Red occurs before yellow

[5, 4, 9, 6, 7, 1, 2, 1] → [1, 1, 2, 4, 5, 6, 7, 9]

stable sorting algorithm

[1, 1, 2, 4, 5, 6, 7, 9]

unstable sorting algorithm

[1, 1, 2, 4, 5, 6, 7, 9]

## In-place algorithm:

Algorithms that do not require any auxiliary data structure to execute, i.e, extra space is  $O(1)$



# External Sorts vs Internal Sorts

Merge

+

Bubble Sort: Stable? In-place?

$O(1)$

YES

[5, 4, 1, 1]

→

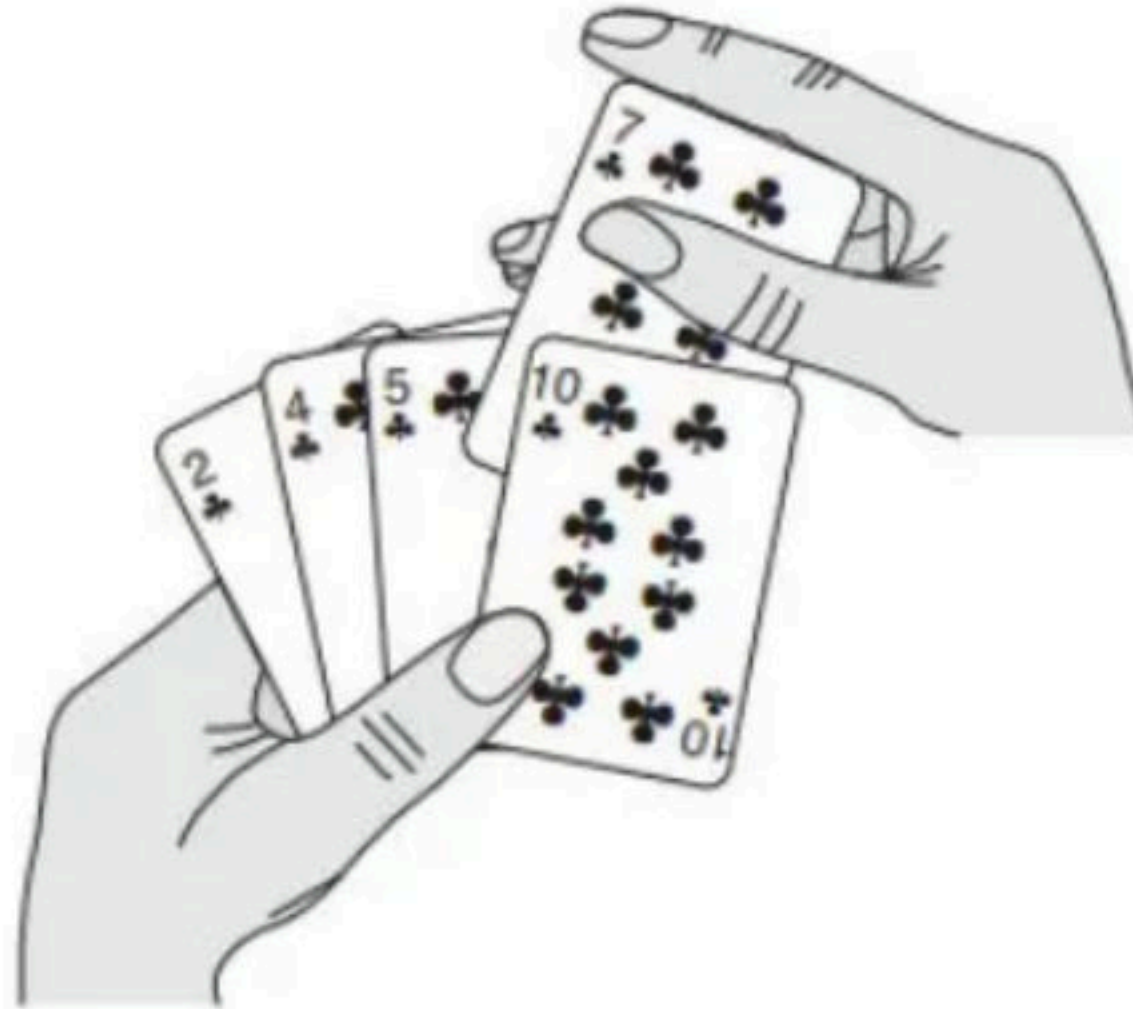
[1, 1, 4, 5]

stable  
Inplace

$A[j] > A[j+1]$

Swapping doesn't  
take place if  
numbers are equal

# Insertion sort



**Figure 2.1** Sorting a hand of cards using insertion sort.



# Example - Insertion Sort

Unsorted array : [5, 3, 4, 10, 9, 6] => Sorted Array: [3, 4, 5, 6, 9, 10]

[5, 3, 4, 10, 9, 6]

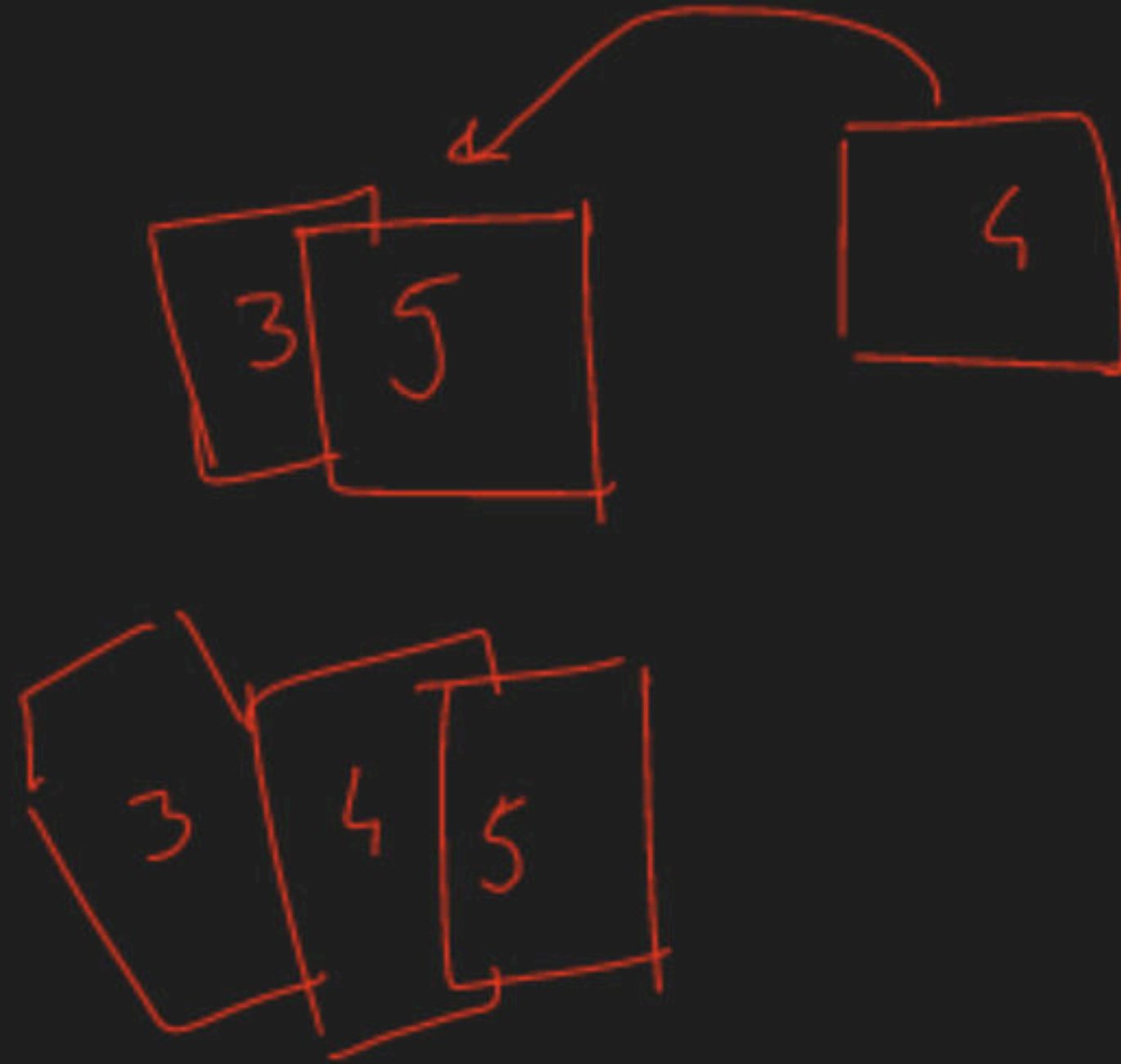
[3, 5, 4, 10, 9, 6]

[3, 4, 5, 10, 9, 6]

[3, 4, 5, 10, 9, 6]

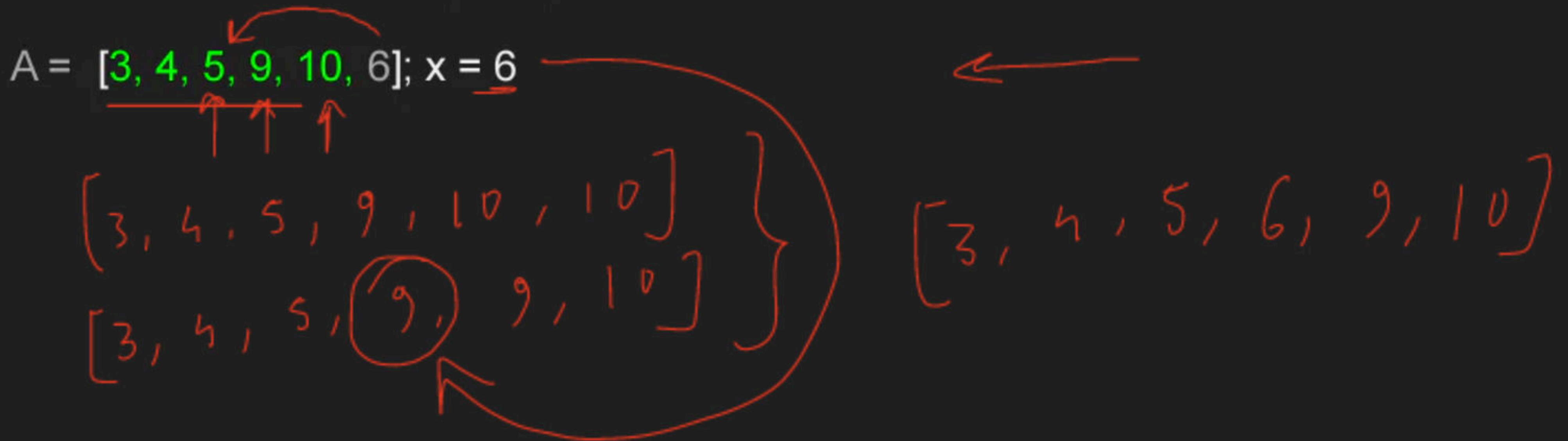
[3, 4, 5, 9, 10, 6]

[3, 4, 5, 6, 9, 10]



## How to insert an element at its correct position in a sorted array?

- To insert an element with value x, shift all elements in the array that are greater than x, by one place to its right. (i.e.  $A[j + 1] = A[j]$  if  $A[j] > x$ )
- Now insert x in the empty space created.





# Time Complexity, Space Complexity & Stability

for ( $i = 1 \dots n$ )

{ while ( ) { }

[5, 4, 3, 2, 1]

Worst Case time complexity:  $O(n^2)$

Space complexity:  $O(1)$   $\leftarrow$  In place algorithm

Best Case:

Time  $\rightarrow$

$O(N)$

$[1, 2, 3, 4, 5]$

Average:  $O(N^2)$

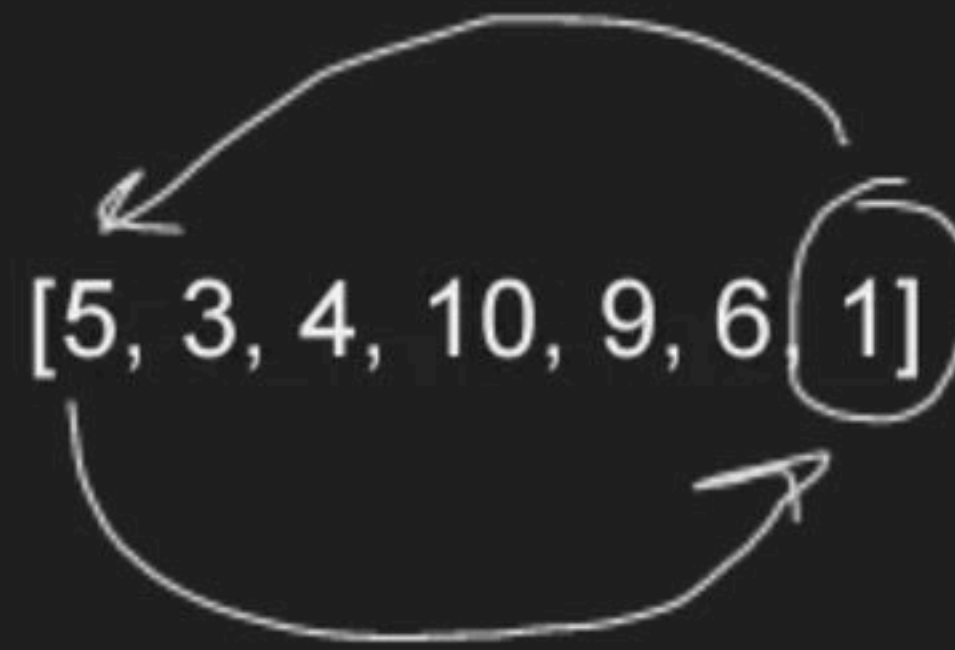
$O(N^2)$

$[5, 4, 1, 1]$

# Selection Sort

- Selects the minimum element from unsorted section and places it in the beginning of the unsorted section.

Example:



Example:

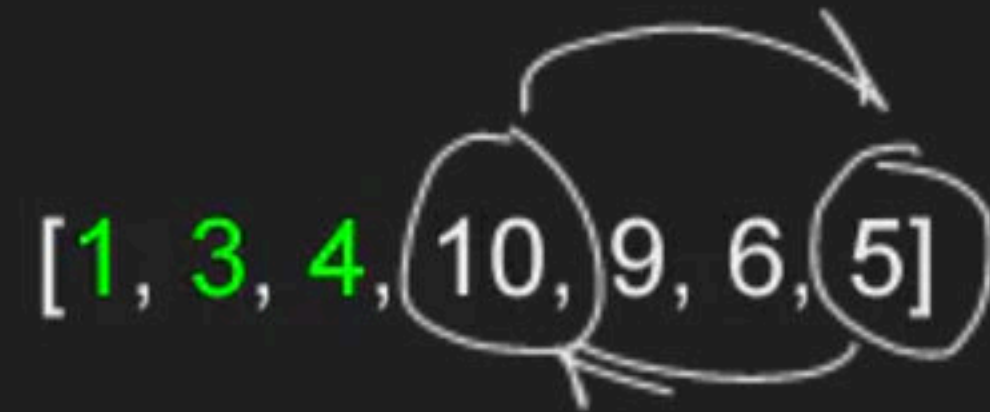
[1, 3, 4, 10, 9, 6, 5]



Example:

[1, 3, 4, 10, 9, 6, 5]

Example:



Example:

[1, 3, 4, 5, 9, 6, 10]



Example:

[1, 3, 4, 5, 6, 9, 10]



Example:

[1, 3, 4, 5, 6, 9, 10]



Example:

[1, 3, 4, 5, 6, 9, 10]

# Time Complexity, Space Complexity & Stability

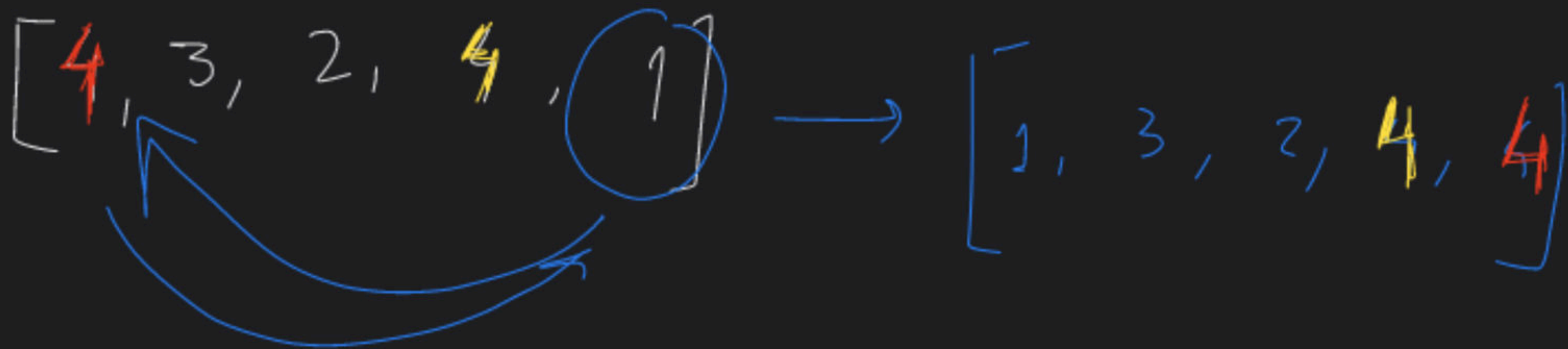
Worst Time:  $O(N^2)$

Best Time:  $O(N^2)$

Av. Time:  $O(N^2)$

Unstable

Space:  $O(1)$  ← In place




Unstable!

\* Selection sort takes at most  $N$   
swaps

H.W Make Selection Sort

✓ Able

Which of the following sorting algorithms in its typical implementation gives best performance when applied on an array which is sorted or almost sorted (maximum 1 or two elements are misplaced)?

- A. Quick Sort
  - ✓ B. Selection Sort
  - C. Bubble Sort
  - ✓ D. Insertion Sort
- 
- A hand-drawn red bracket groups options B, C, and D. A red arrow points from the bottom of this bracket towards the right side of the slide.



What is the best case time complexity we can achieve in bubble sort?

A.  $O(n^2)$

B.  $O(n \log n)$

☒ C.  $O(n)$

D.  $O(n^3)$

Selection Sort never makes more than  $n$  swaps.

~~A. True~~

B. False

What is the best auxiliary space complexity a sorting algorithm can have?

A.  $O(n)$

☒ B.  $O(1)$

C.  $O(n \log n)$

D.  $O(n^2)$

Consider the array  $\{4, 3, 5, 2\}$ . How many swaps will be needed to sort the array using selection sort?

- A. 1
- ~~B. 2~~
- C. 3
- D. 4

