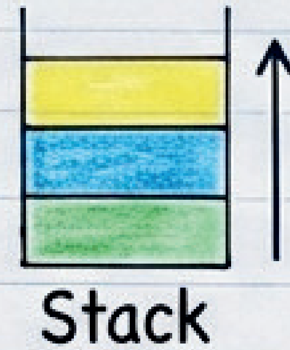
Chapter 1: Introduction to DSA

(notes by ~ Kamal kishor)

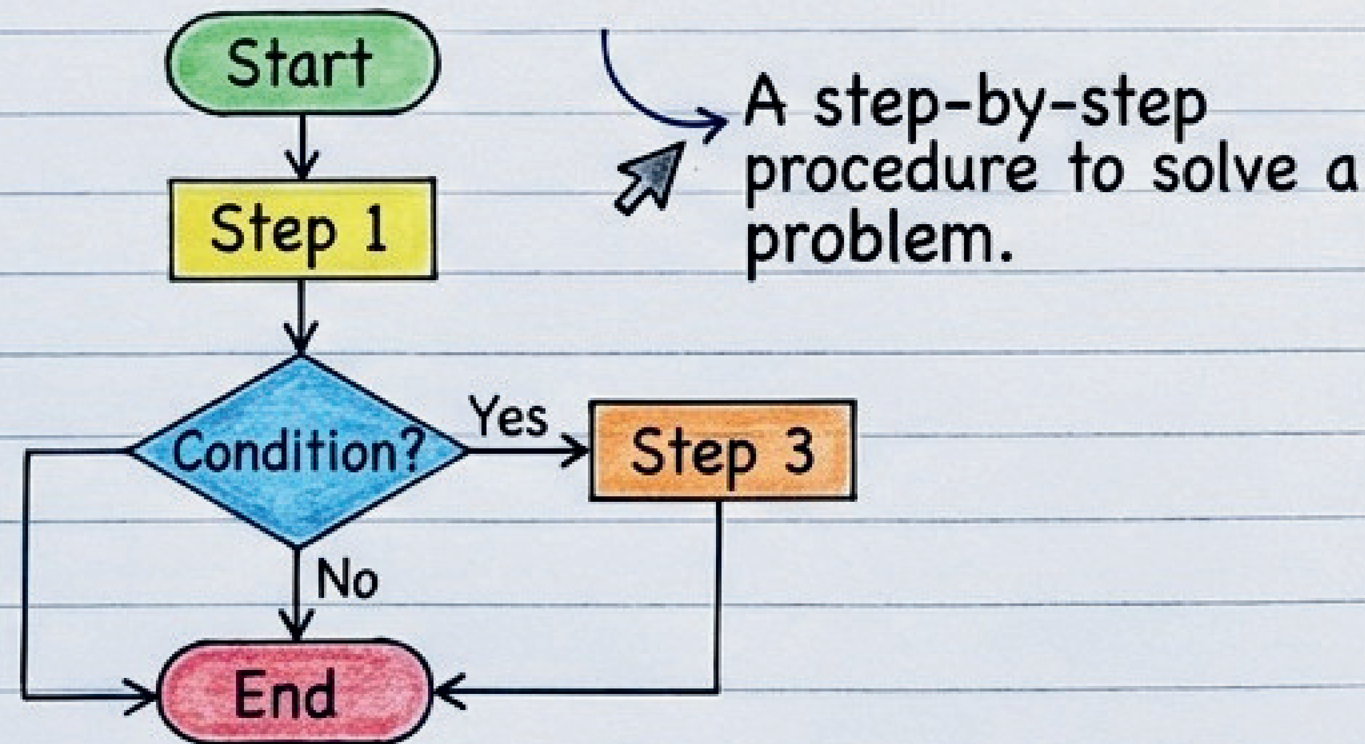
What is Data Structure?

A way to store and organize data in a computer so that it can be used efficiently.



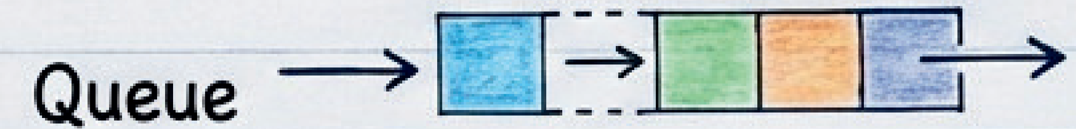
What is Algorithm?

A finite sequence of well-defined steps to solve a specific problem. Like a recipe!



Properties of Algorithms

- ✓ 1. Finiteness: Must terminate after a finite number of steps.
- ✓ 2. Definiteness: Steps must be precisely defined.
- ✓ 3. Input/Output: Takes inputs, produces outputs.

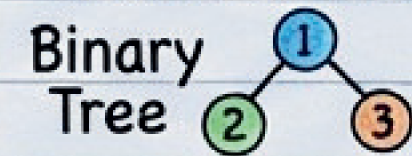


Why DSA is important?

- ✓ Efficient Problem Solving (choosing right tools).
- ✓ Optimal Resource Usage (memory/time).
- ✓ Coding Interviews (crucial for challenges).

Types of Data Structures

Linear	Non-Linear
Sequential organization.	Hierarchical.



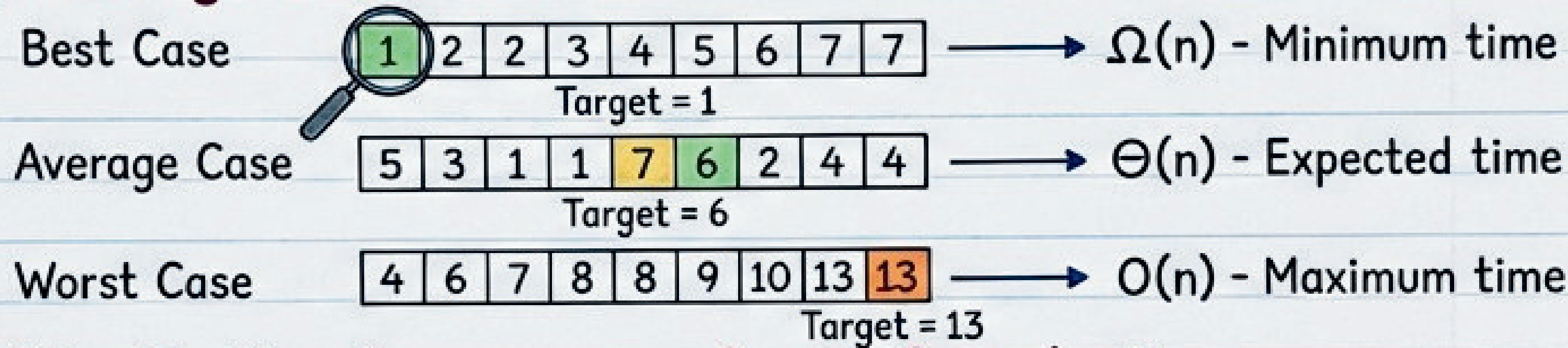
Abstract Data Types (ADT): Defined by operations (List, Stack, Queue, Map).

Time & Space Complexity

Why complexity analysis?

- ✓ Evaluate Efficiency: Measuring running time & memory usage.
- ✓ Compare Algorithms: Choosing the optimal one.
- ✓ Scalability: Handling large inputs without degrading.

Best, Average & Worst Case

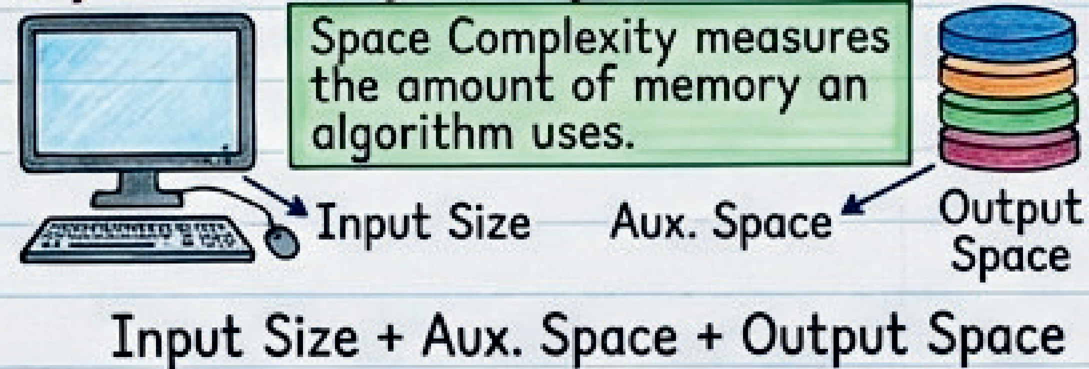


Big-O, Big-Ω, Big-Θ

1	↑	Big-O	Upper Bound	$f(n) \leq O(g(n))$
2	↓	Big-Ω	Lower Bound	$f(n) \geq \Omega(g(n))$
3	↑↓	Big-Θ	Tight Bound	$f(n) = \Theta(g(n))$

These notations provide a measure of the time complexity of an algorithm in relation to the input size.

Space Complexity



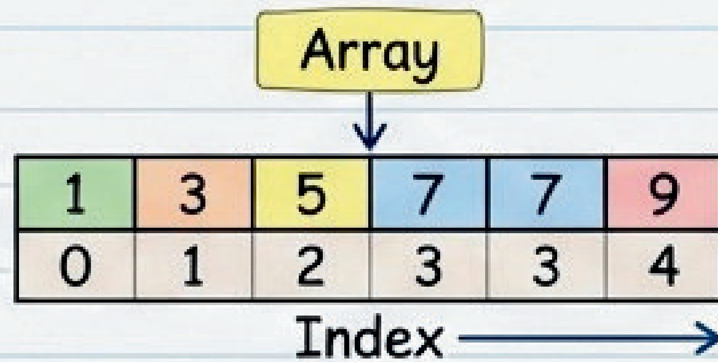
Input Size + Aux. Space + Output Space

Complexity of Loops & Recursion

```
for (int i=0; i<n; i++)   → O(n)
for (i... for j...)       → O(n²)
for (i=1; i<n; i*=2)      → O(log n)
```

Chapter 2: Arrays (Introduction & Types)

An array is a collection of elements of the same data type stored in contiguous memory locations.



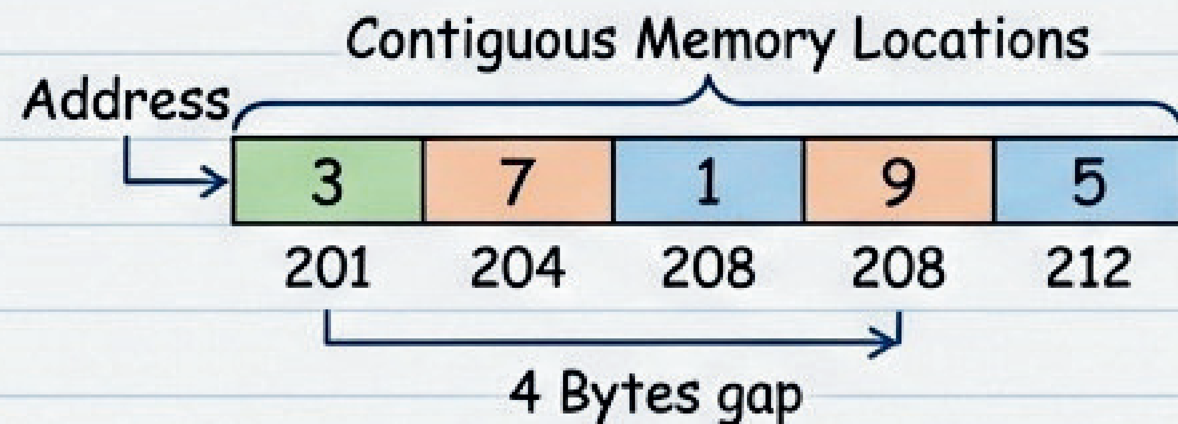
Characteristics of Arrays

Fixed Size: Defined at declaration.

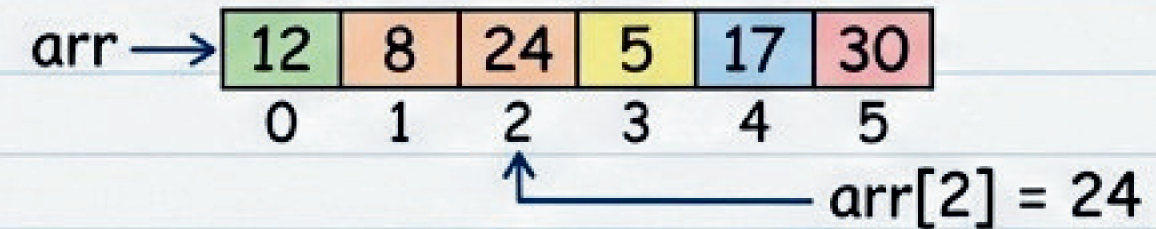
Homogeneous Elements: Same data type.

Contiguous Memory: Adjacent locations.

Memory Representation

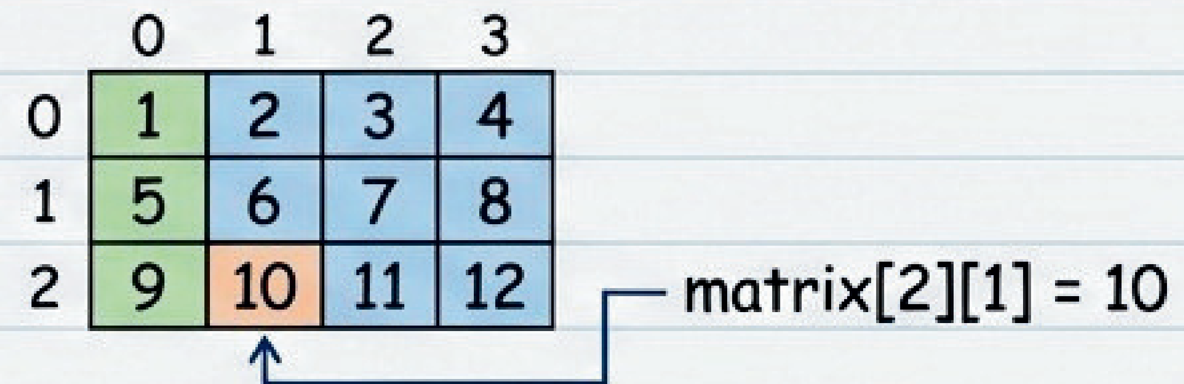


2.2 1D Arrays

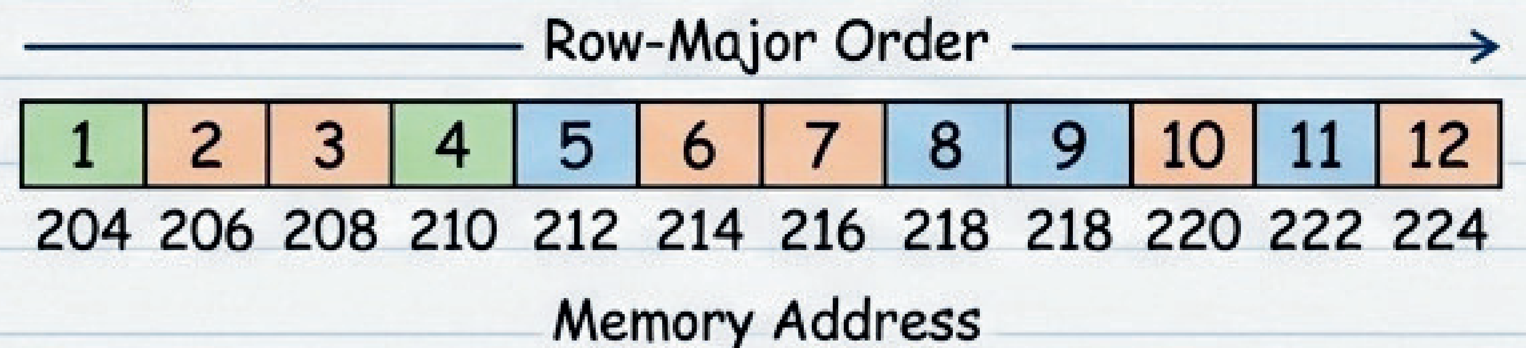



```
int arr[5] = {3, 7, 1, 9, 5};
```

2.3 2D Arrays (Matrix)

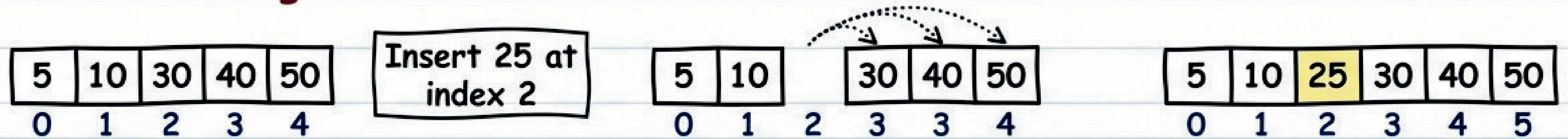


Memory Representation (Row-Major Order)



Array Operations & Problems

1. Insertion Logic

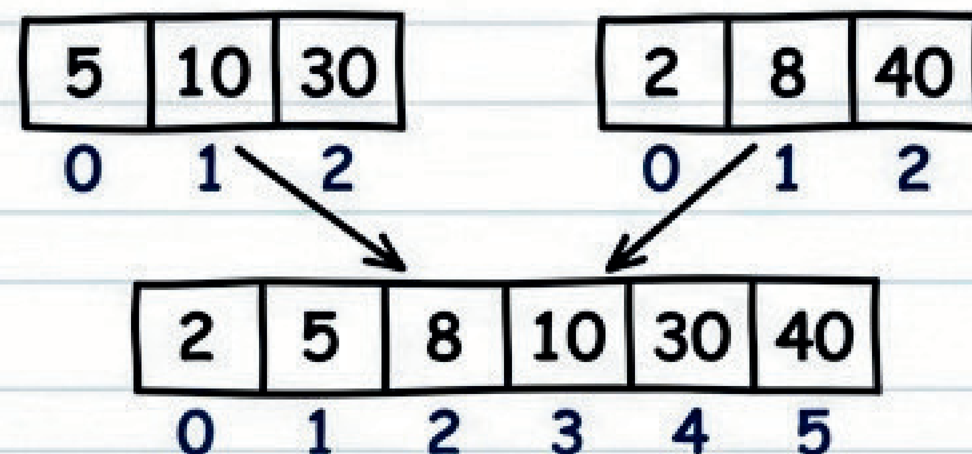


✓ Time Complexity -> O(n) (Worst case shifting)

2. Deletion Logic

✓ Time Complexity -> O(n)

3. Merge Two Sorted Arrays



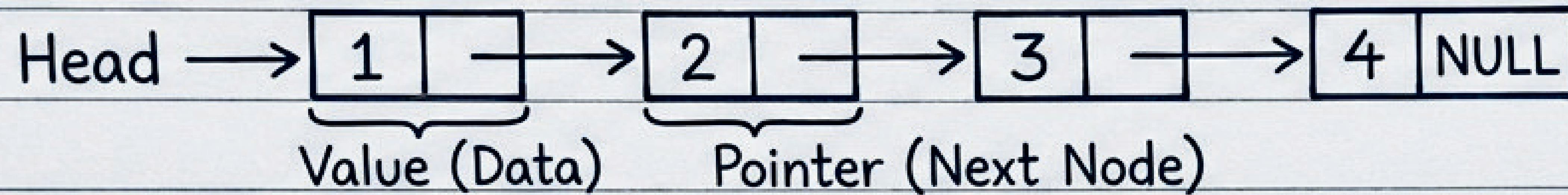
Tips for Solving Array Problems

- ✓ Plan approach before coding.
- ✓ Consider edge cases (empty array, large values).
- ✓ Watch out for fragmentation.

Chapter 3: Linked List (Basics)

3.1 Introduction

A sequence of elements (nodes) where each contains a value and a pointer to the next.

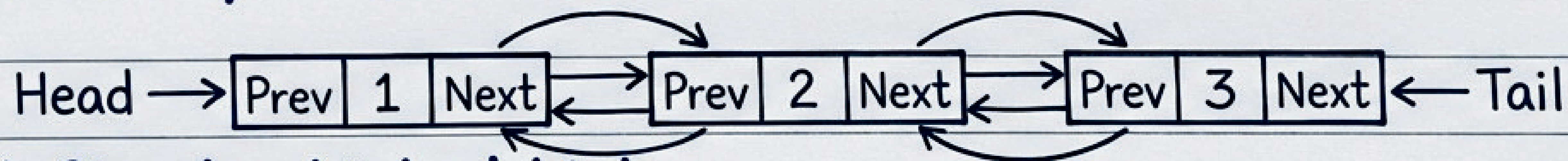


3.2 Singly Linked List

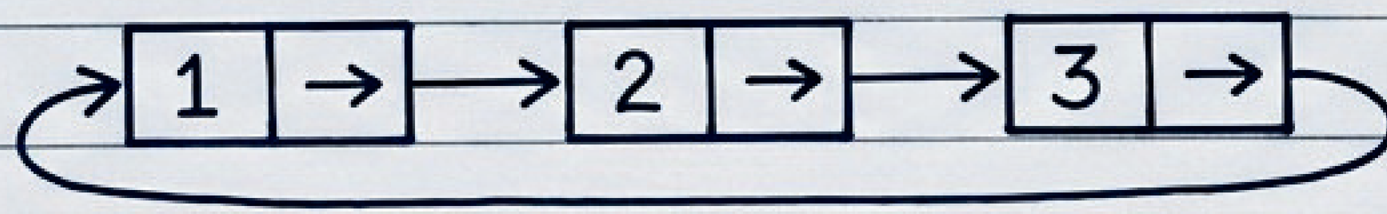
✓ Unidirectional ✓ Dynamic Size ✓ Memory Efficient

How to create a new node? • New Node → | 10 | Next |

3.3 Doubly Linked List



3.4 Circular Linked List

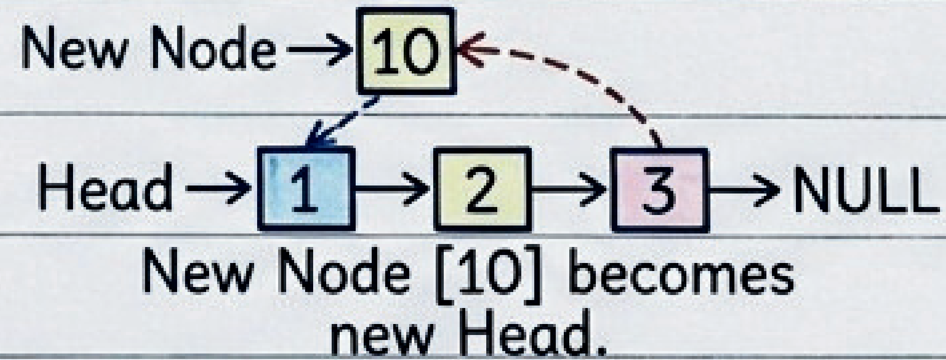


No NULL value; forms a continuous loop.

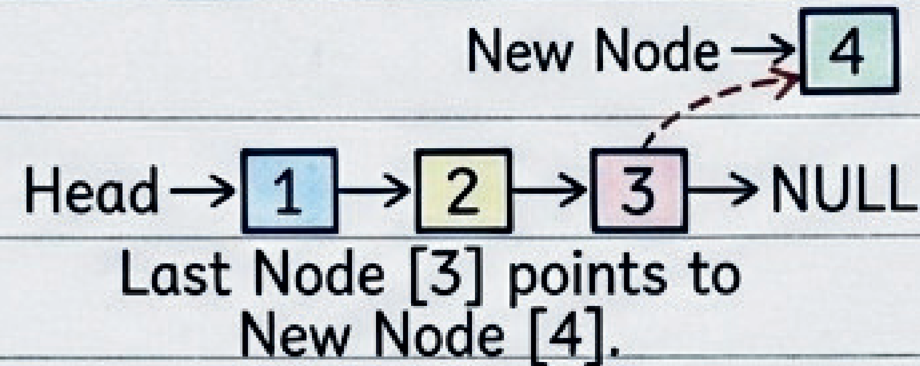
Linked List Operations

1. Insertion Logic

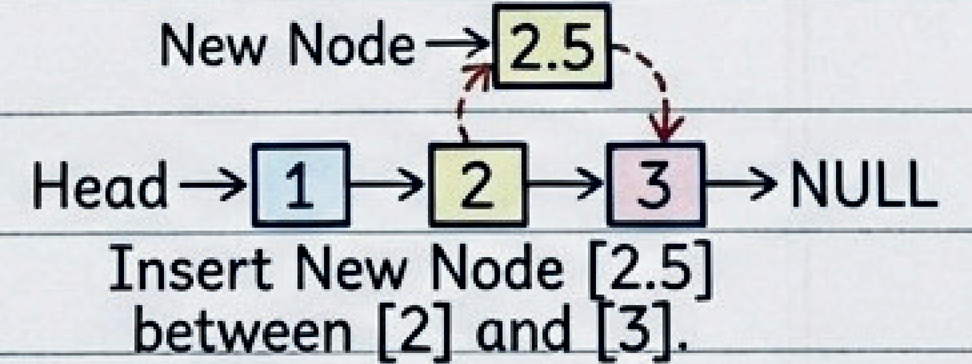
Beginning



End

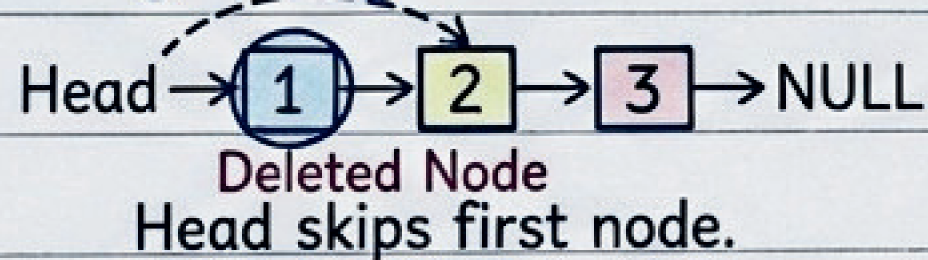


Specific Position

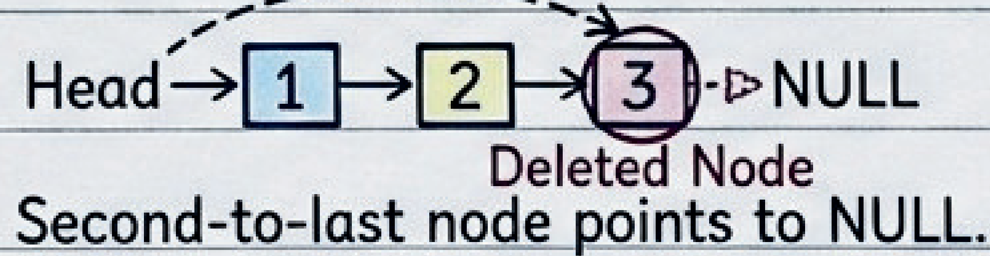


2. Deletion Logic

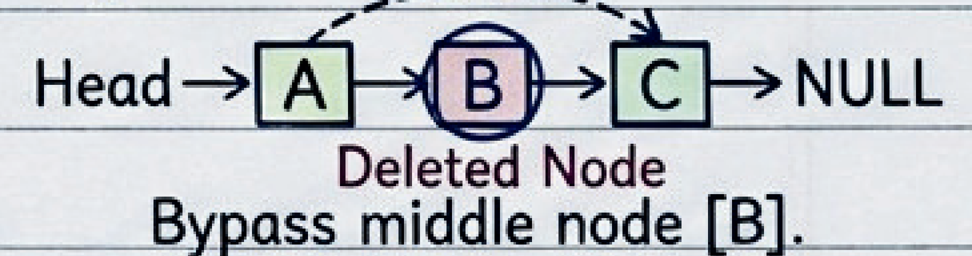
Beginning



End



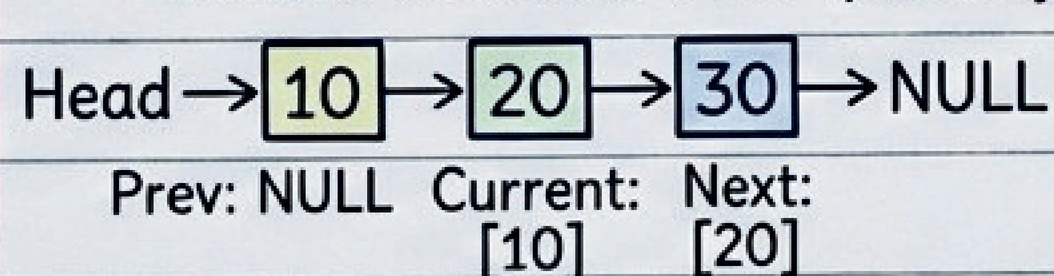
Specific Position

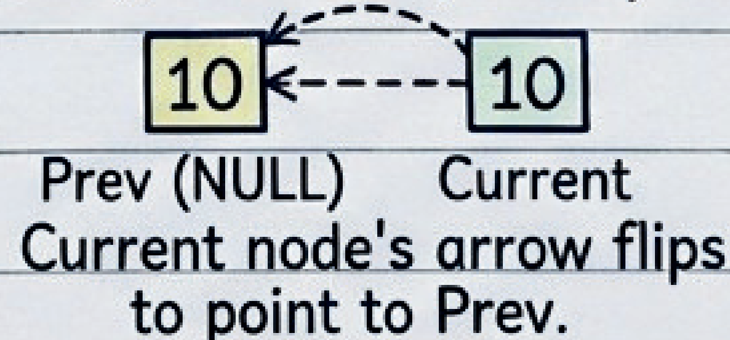


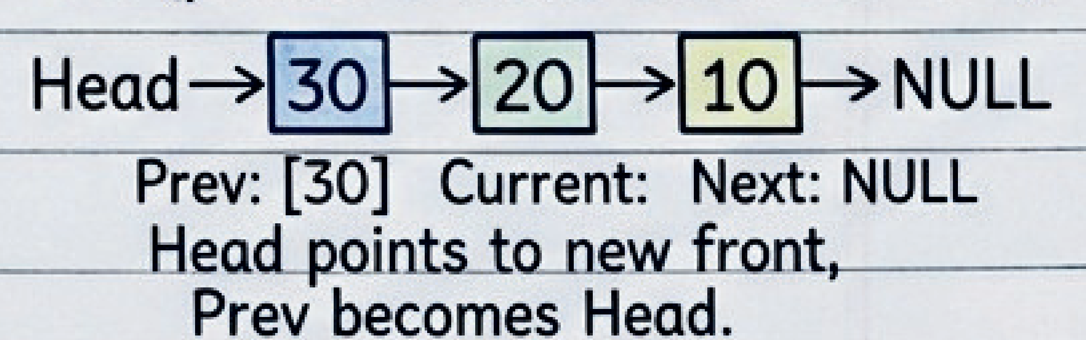
3. Reversing a List

How to reverse a Linked List?

- Define Pointers: Prev (initially NULL), Current (initially Head), Next (points to node after Current).

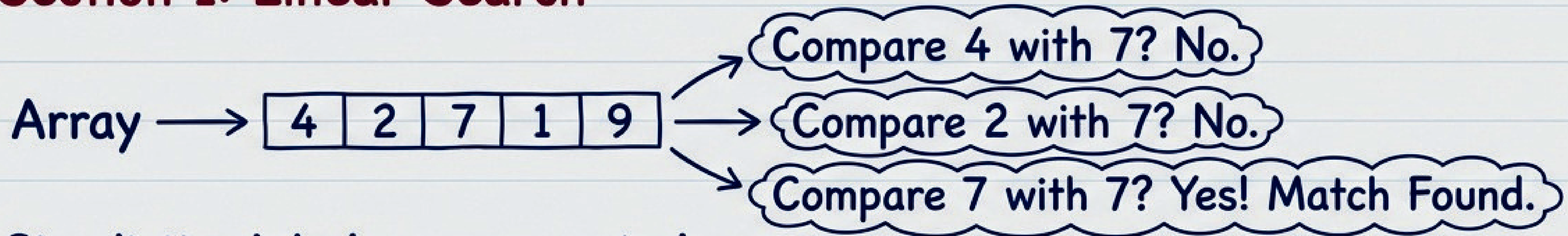






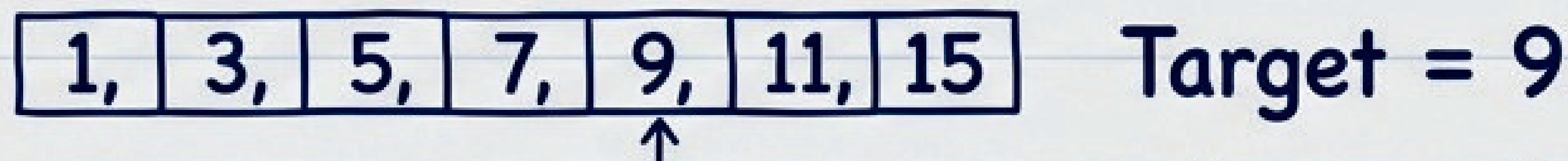
Chapter 4: Searching Algorithms

Section 1: Linear Search



Simplicity: Works on unsorted arrays.

Section 2: Binary Search (Sorted Arrays Only)



Target = 9

Pass 1: Low=0, High=6. Mid=3 (Value 7). 7 < 9, so "Search Right" ⟶

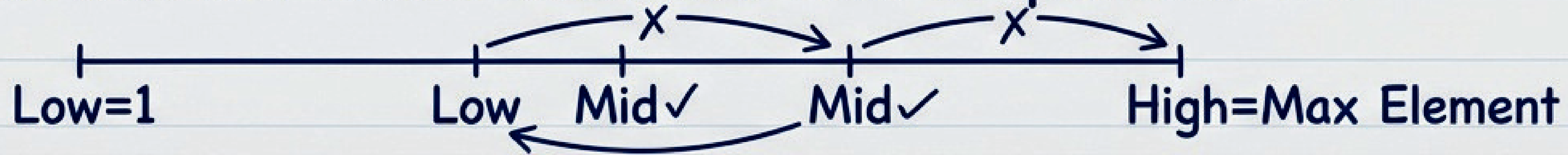
Pass 2: New Low=4. New Mid (Value 11). 11 > 9, so "Search Left" ⟵

Pass 3: Match 9! ⟶ 4

$$\text{Mid} = \frac{\text{Low} + \text{High}}{2}$$

Section 3: Application (Example: Smallest Divisor)

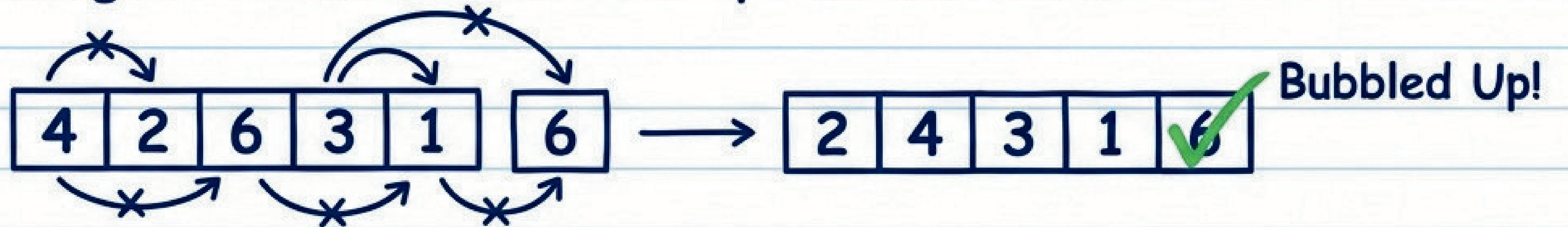
Find smallest divisor such that sum of quotients ≤ 7



Chapter 5: Elementary Sorting Algorithms

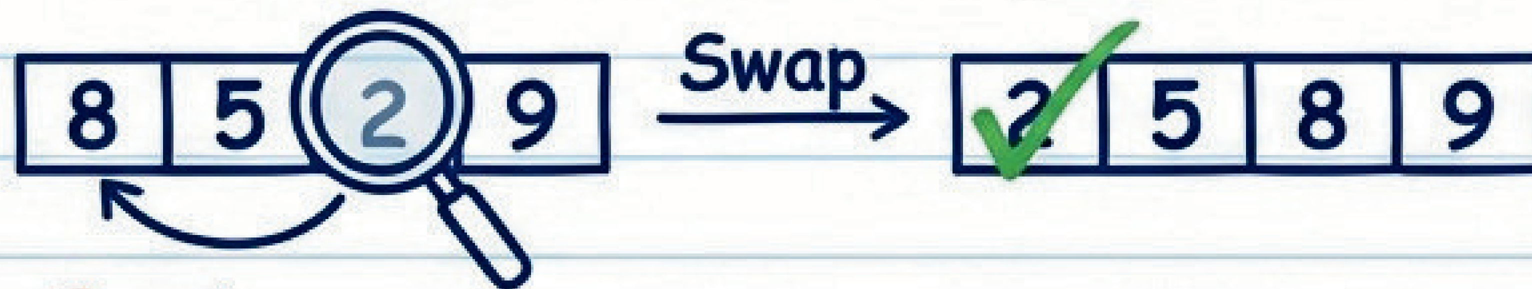
1. Bubble Sort

Concept: Largest element "bubbles up" to the end.



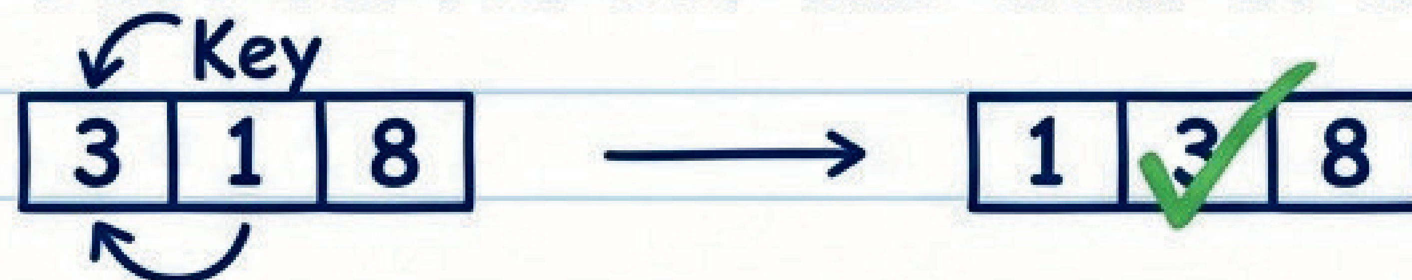
2. Selection Sort

Concept: Find minimum and swap to front.



3. Insertion Sort

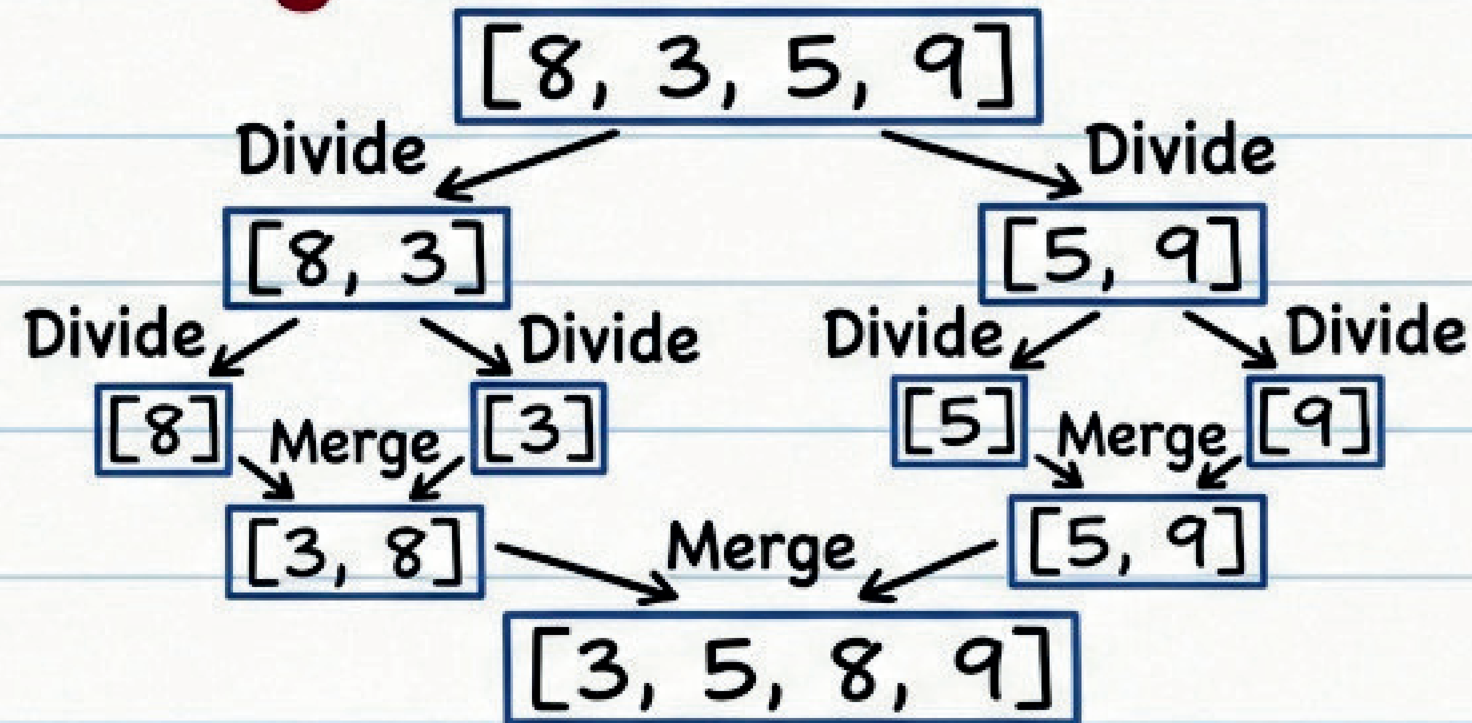
Concept: Build sorted list one item at a time (like cards).



All generally $O(n^2)$.

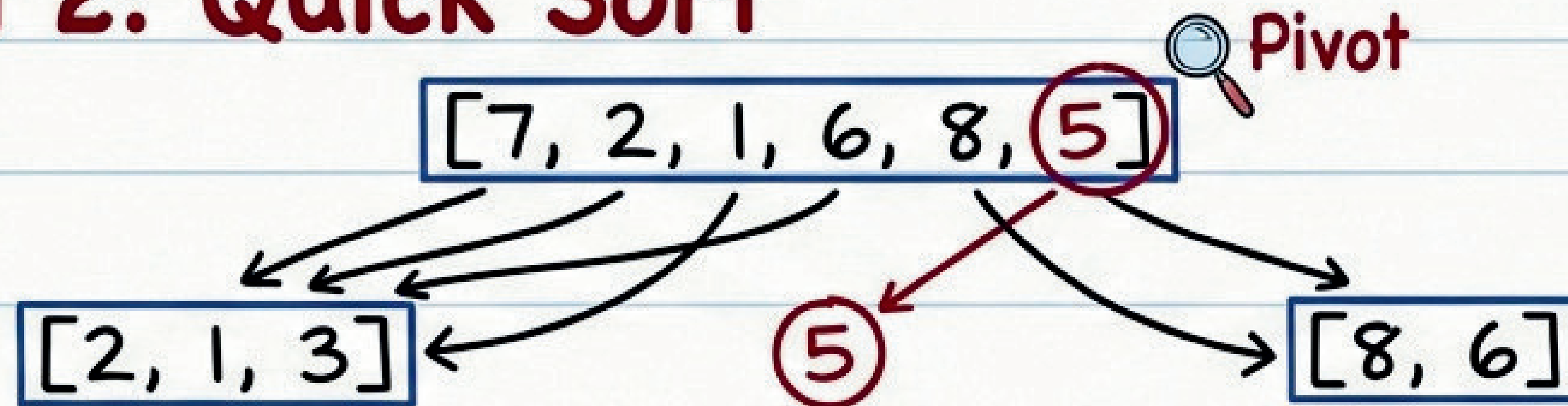
Divide & Conquer Sorting

Section 1: Merge Sort



Stable Time: $O(n \log n)$

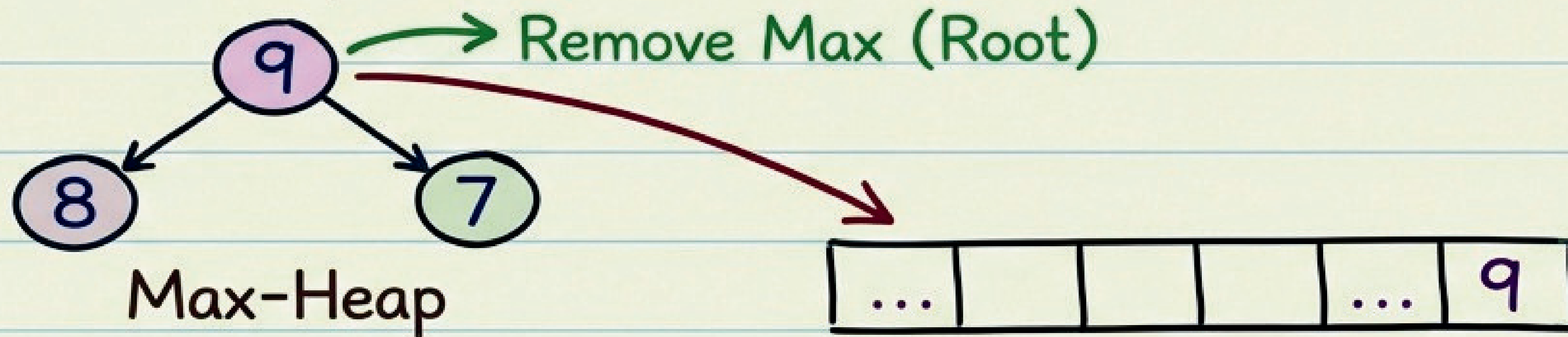
Section 2: Quick Sort



Not Stable Avg Time: $O(n \log n)$ Worst: $O(n^2)$

Special Sorting Techniques

Section 1: Heap Sort



💡 Algorithm: Build Max Heap → Swap Root with Last → Heapify

Section 2: Counting Sort

Concept: Non-comparison based.

Input: [3, 5, 3, 1]

Count Array

1	2	3	4	5
1		2		1

Output: [1, 3, 3, 5]

✓ Complexity: $O(n + k)$

Radix Sort & Stability

Section 1: Radix Sort

Sorts numbers digit by digit (LSD to MSD).

Step 1 (Units): Unsorted list: | 170, | 45, | 75, | 90, | 802 |

Sorted by last digit: | 170, | 90, | 802, | 45, | 75 |

Step 2 (Tens): Sorted by second digit.

(Tens): Sorted by second digit. [170, 45, 75, 90, 802] → [45, 75, 170, 802, 90]

Step 3 (Hundreds): Sorted by first digit. [45, 75, 170, 802, 90] → [45, 75, 90, 170, 802]

Uses Counting Sort as a subroutine.

Section 2: Stability

Definition: A sort is stable if it preserves the relative order of equal elements.

Unsorted: | 4A, | 2 | 4B |

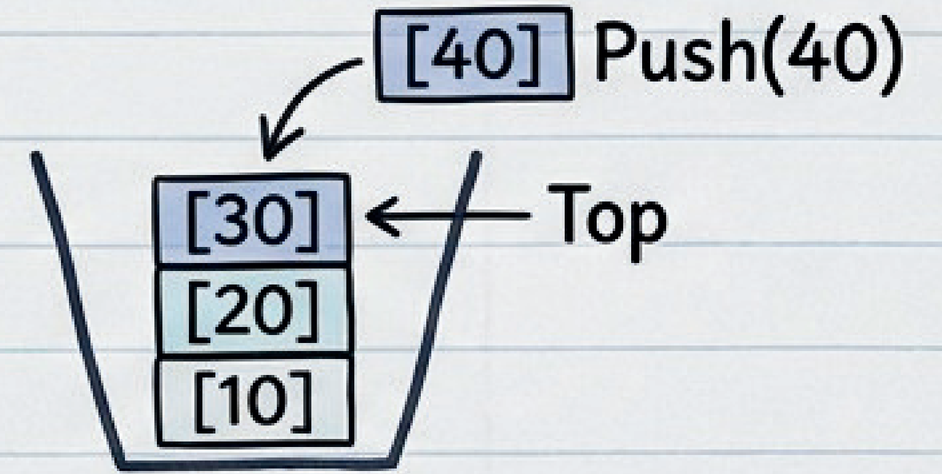
Stable Sort: | 2 | 4A, | 4B |
(Correct)

Unstable Sort: | 2 | 4B, | 4A |
(Order swapped)

Radix Sort requires stability!

Chapter 6: The Stack

Section 1: Concept



✓ Definition: LIFO (Last In, First Out) principle.

Section 2: Operations

✓ Push: Add to top.

✓ Pop: Remove from top.

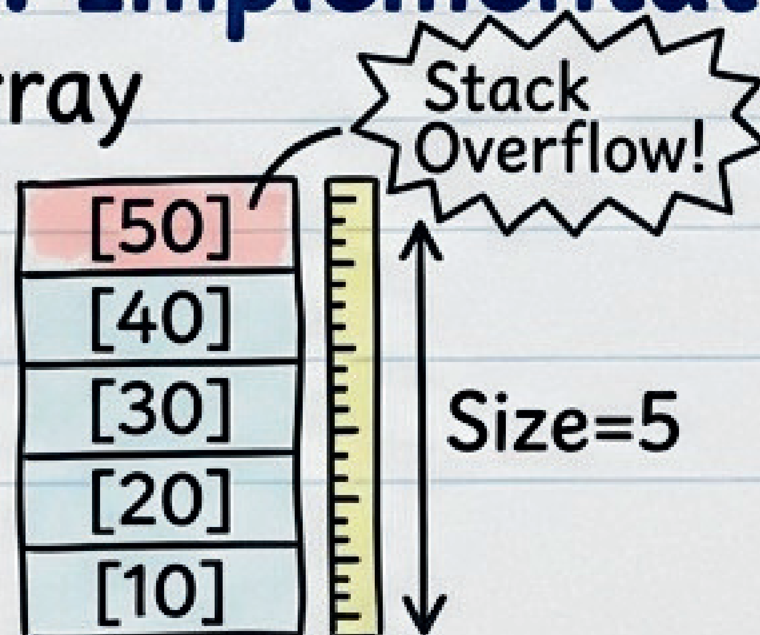
History of Operations	10	20	30	40
Order of Removal	40	30	20	10

LIFO Order: Last In (40), First Out

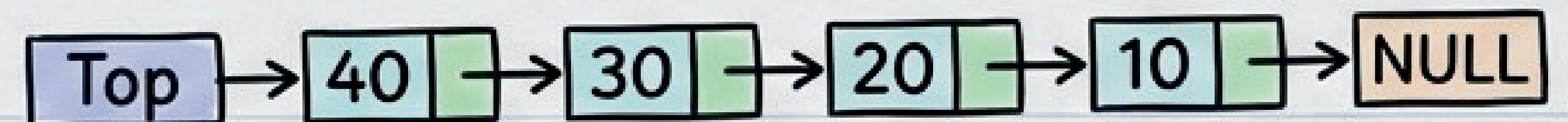
Section 3: Implementation

Using Array

Stack (Fixed Size)



Using Linked List



Dynamic Size - No Overflow

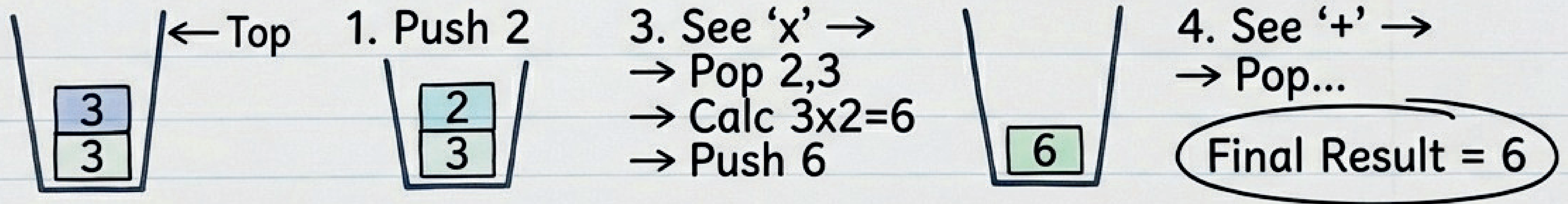
Stack Applications & Expressions

Section 1: Notations

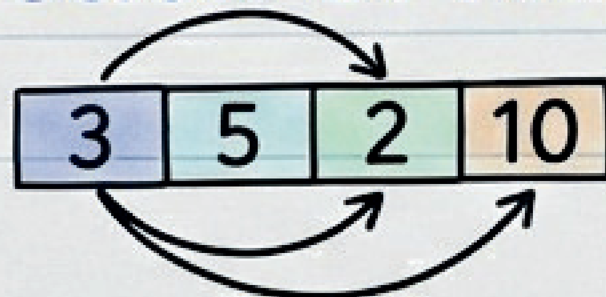
Notation	Operator Position	Example
Infix	Operator Between	A + B
Prefix	Operator Before	+ A B
Postfix	Operator After	A B +

Section 2: Expression Evaluation

Evaluate Postfix using Stack: 3 2 x +



Section 3: Next Greater Element (NGE)

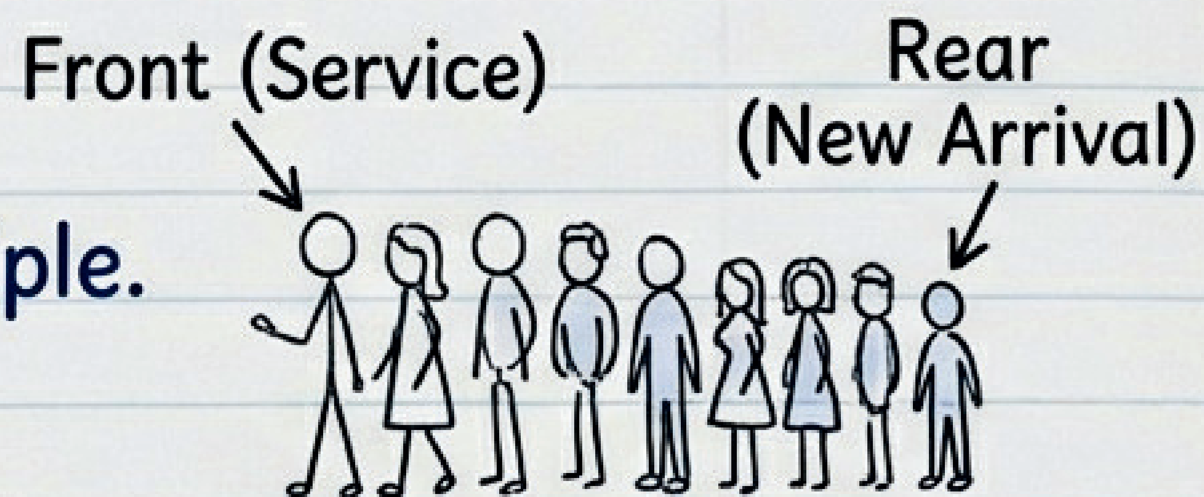


Find first greater element to the right using a Stack.

Chapter 9: The Queue

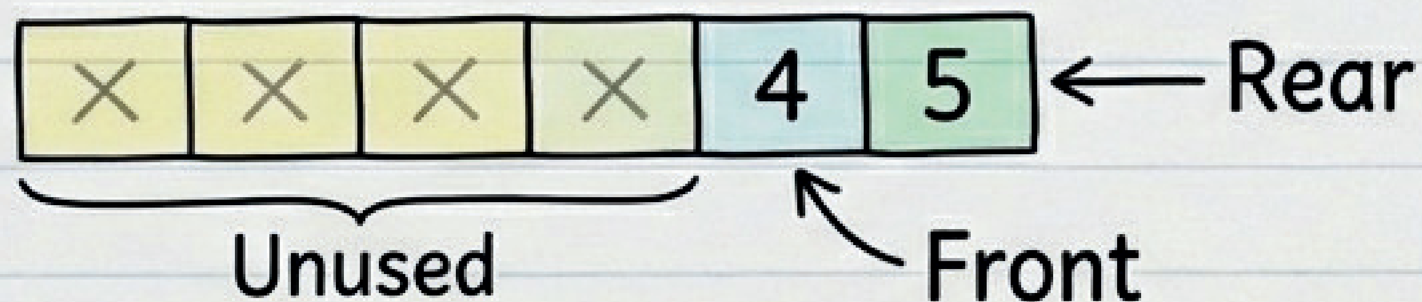
Section 1: Concept

✓ Definition: FIFO (First In, First Out) principle.



Section 2: The Problem

Simple Queue Limitation

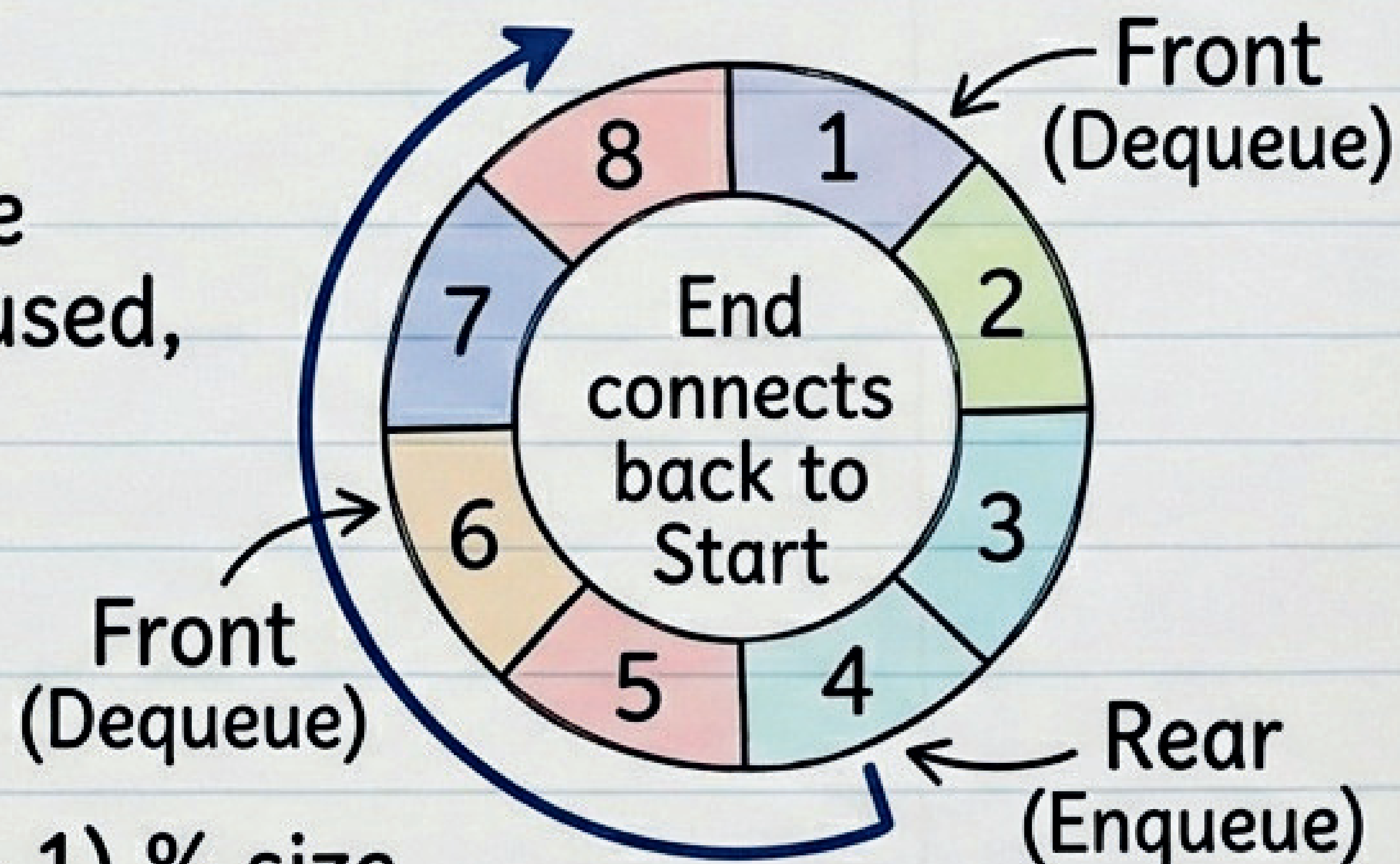


⚠ Once elements are removed from the front, the empty space cannot be reused, which leads to False Overflow.

Section 3: The Solution

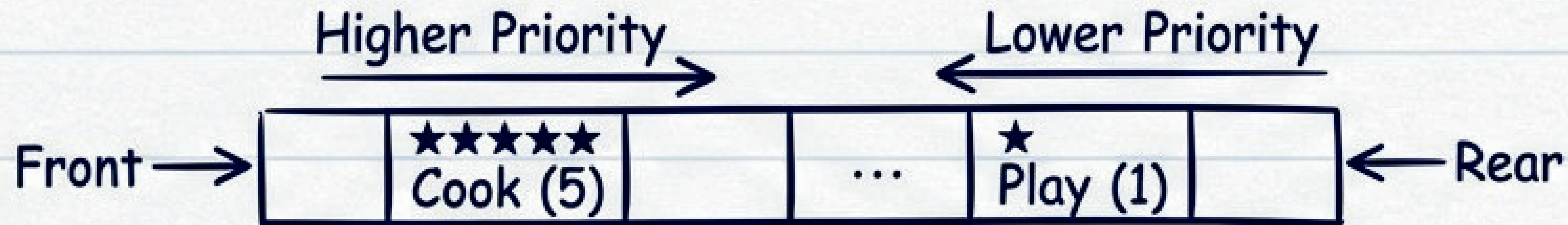
Circular Queue

Condition to move circularly: (index + 1) % size



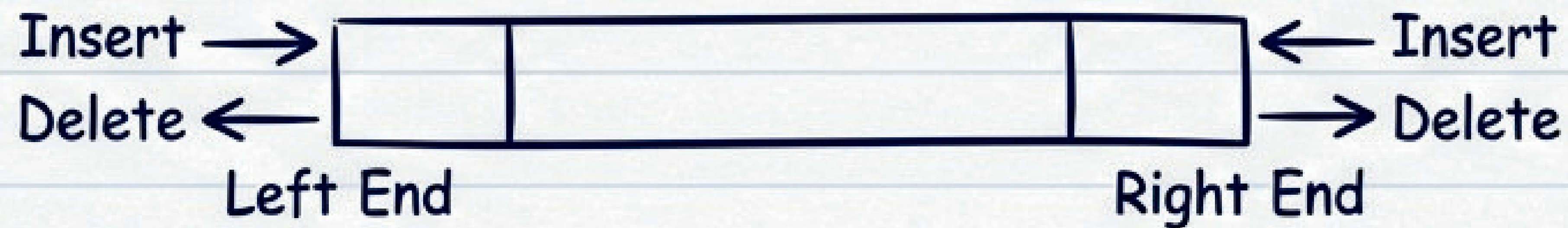
Advanced Queues & Conversions

Section 1: Priority Queue



📌 Use Case: CPU Scheduling

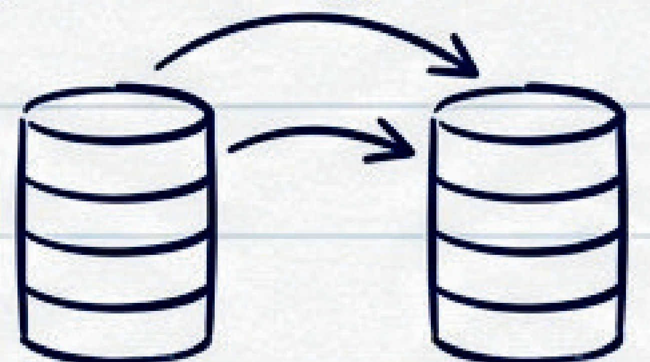
Section 2: Deque (Double Ended Queue)



Section 3: Conversions

Queue using Stacks

Stack 1 for enqueueing,
Stack 2 for dequeueing



Reverse order by transferring

Stack using Queues

Q1: Main, Q2: Auxiliary