ARRAY DSA Questions EASY

	COMPANY	QUESTION NAME	APPROACH	GFG	LEETCODE	NOTES
1	*Infosys*, *Oracle*, *Wipro*, *Morgan Stanley*	Largest Element in Array TC: O(n) SC : O(1)		GFG		If n is not initialized, u r using the size of the vector &arr, you must explicitly initialize `int n = arr.size();`

2	*SAP Labs*,*Rockstand*	Second Largest elem in array without sorting
		TC: O(n) SC : O(1)
				GFG		Visible array→ `arr[0]`. Invisible data→ `INT_MIN`
						`Can't trust the data→ INT_MAX`
						`INT_MIN → start with the` **`worst max`**
						`INT_MAX → start with the` **`worst min`**

3		Check if array is sorted TC: O(n) SC : O(1)		GFG		
4	*Zoho*, *Morgan Stanley*, *Microsoft*, *Samsung*, *Google*, *Wipro*, *Xome*	Remove Duplicates Sorted Array TC: O(n) SC : O(1)	*2 pointers*	GFG	LEETCODE	
5		Intersection of two sorted arrays TC: O(n1 +n2) SC :O(n1+n2) WC Intersection of two unsorted arrays (LC)	*2 pointers*	GFG	LEETCODE	Intersection of two unsorted arrays (LC) We used unordered map
6	*Flipkart, Morgan Stanley, Accolite, D-E-Shaw, OlaCabs, Pay, Visa, Intuit, Adobe, CISCO, Qualcomm TCS*	Missing number	*XOR*	GFG	LEETCODE	*Range Type: `1 to n` (like GFG)* *→ XOR loop range: from 1 to `n-1`* *→ ✅ Yes, you need to add `xor1 ^= n` outside the loop to include the last number.* *Range Type: `0 to n` (like Leetcode)* *→ XOR loop range: from `0` to n* *→ ❌ No extra step needed — the loop already covers the full range.*
7	*Amazon, Google ,META*	Max Consecutive 1's			LEETCODE	
8	*Amazon*	Single Number		GFG	LEETCODE	
9	*Zoho, Flipkart, Morgan Stanley, Accolite, Amazon, Microsoft, FactSet, Hike, Adobe, Google, Wipro, SAP Labs, CarWale*	Two Sum - Pair with Given Sum	*2 pointers, HASHMAP*	GFG	LEETCODE	***HashMAP -->*** Return True/False OR Return Indices TC: O(NlogN) MAP O(N) UMAP(B/A) O(N^2) worst SC: O(N) ***2 POINTERS*** →Return True/False TC: O(N) + O(NlogN) sorting SC: O(N)
10	*Bloomberg, Facebook, Intel, Infosys, Zoho, Morgan Stanley, Amazon, Microsoft, Samsung, Yahoo, PayPal, Nvidia, Oracle, Visa, Walmart, Goldman Sachs, TCS, Adobe, Google, IBM, Accenture, Apple, Uber*	Best Time to Buy and Sell Stock (1 Transaction)		GFG	LEETCODE	
		Valid Anagram		GFG	LEETCODE	

		Find All Numbers Disappeared in an Array		GFG	LEETCODE	
		Convert 1D Array Into 2D Array		GFG	LEETCODE	
				GFG	LEETCODE	

MEDIUM

1		Rotate Array by One (clockwise direction/right) TC: O(n) SC : O(1)			.	Whenever they say rotate left/right, look at the key elem that moves, either first/last & shift accordingly. Right Rotation by 1→**last element** (`arr[n-1]`) Left Rotation by 1→**first element** (`arr[0]`) Dont focus on shifting all elements
2	*Amazon, Microsoft, MAQ Software*	Rotate the array to the left (counter-clockwise) rotate the array to the right		GFG	LEETCODE	**Left = F-R-A** → First–Rest–All **Right = A-F-R** → All–First–Rest
3	*Paytm,Amazon Microsoft, Samsung, SAP Labs, Linkedin, Bloomberg*	Move All Zeroes to End TC: O(n) SC : O(1)	*2 pointers*	GFG	LEETCODE	
4	*Amazon*	Union of 2 Sorted Arrays TC: O(n1 +n2) SC :O(n1+n2) WC	*2 pointers*	GFG		
5.	*Paytm, Flipkart, Morgan Stanley, Amazon, Microsoft, OYO Rooms, Samsung, Snapdeal, Hike, MakeMyTrip, Ola Cabs, Walmart, MAQ Software, Adobe, Yatra.com, SAP Labs, Qualcomm*	Sort 0s, 1s and 2s	*Dutch National flag algo*	GFG	LEETCODE	
6	*Zoho, Flipkart, Morgan Stanley, Accolite, Amazon, Microsoft, Samsung, Snapdeal, 24*7 Innovation Labs, Citrix, D-E-Shaw, FactSet, Hike, Housing.com, MetLife, Ola Cabs, Oracle, Payu, Teradata, Visa, Walmart, Adobe, Google, Arcesium*	Maximum Subarray TC: O(n) SC : O(1) (when we have to find the highest sum with the longest subarray)	*Kadanes algo*	GFG	LEETCODE	
7	*Amazon*	Longest Subarray with Sum K (When the k is already given)	*2 pointers, HASHMAP*	GFG		***HashMAP** → **FOR +VE, -VE AND 0'S*** TC: O(NlogN) MAP O(N) UMAP(B/A) O(N^2) worst SC: O(N) ***2 POINTERS*** → FOR POSITIVE NUMS IN ARRAY TC: O(2N) SC: O(1)
8	*Flipkart, Accolite, Amazon, Microsoft, D-E-Shaw, Google,*	Majority Element TC: O(n) + (O(n)--> only where maynot	*Moore's Voting algo*	GFG	LEETCODE	

	Nagarro, Atlassian	condition occurs) SC : O(1)				
9		Best Time to Buy and Sell Stock (Multiple Transaction)	*Greedy apporach*	GFG	LEETCODE	
		Group Anagrams				
		Top K Frequent Elements		GFG	LEETCODE	
		Encode and Decode Strings		GFG	LEETCODE	
		Product of Array Except Self		GFG	LEETCODE	
		Find the Duplicate Number		GFG	LEETCODE	
		Find All Duplicates in an Array				
		Set Matrix Zeroes				
		Spiral Matrix				
		Rotate Image				
		Valid Sudoku				
		Factor Combinations				

Day 1: Core Patterns + Frequency + Logic (8)

1. Two Sum
2. Best Time to Buy and Sell Stock (1 Transaction)
3. Best Time to Buy and Sell Stock II (Multiple)
4. Maximum Subarray (Kadane's)
5. Find the Duplicate Number
6. Find All Duplicates in an Array
7. Find All Numbers Disappeared in an Array
8. Remove Duplicates from Sorted Array

✅ **Day 2: Hashing + Sorting + Subarray Based (9)**

9. Valid Anagram
10. Group Anagrams

11. Top K Frequent Elements
12. Majority Element (> n/2)
13. Majority Element (> n/3)
14. Longest Consecutive Sequence
15. Product of Array Except Self
16. Next Permutation
17. Rearrange Array in Alternating Positive & Negative

✅ **Day 3: Matrix + Prefix + Subarray Sum (8)**

18. Set Matrix Zeroes
19. Spiral Matrix
20. Rotate Image (90 degrees)
21. Valid Sudoku
22. Convert 1D Array Into 2D Array
23. Count Subarrays with Given Sum (Prefix + Hash)
24. Factor Combinations (Backtracking Preview)
25. Leaders in Array

Topic 1 — ARRAYS (Aug 11 → Aug 18)

High-impact must-do (solve first, in order):

1. **Two Sum (hashmap / two-pointer)**
2. **Best Time to Buy & Sell Stock (1 transaction)**
3. **Product of Array Except Self.** dwf.devGeeksforGeeks
4. **Maximum Subarray (Kadane's).** dwf.dev
5. **Move Zeroes to End**
6. **Rotate Array (left/right)**
7. **Sort Colors (0/1/2 — Dutch national flag)**
8. **Missing Number (XOR trick)**
9. **Trapping Rain Water (harder — must master).** GeeksforGeeks
10. **Top K Frequent Elements / Kth Largest (heap)**
11. **Longest Consecutive Sequence (hashset)**
12. **Count subarrays with sum K (prefix + hashmap)**

Topic 2 — STRINGS (Aug 19 → Aug 26)

Must-do list (priority):

1. **Longest Substring Without Repeating Characters**
2. **Valid Anagram**
3. **Group Anagrams**
4. **Longest Palindromic Substring (expand/DP/Manacher)**
5. **Minimum Window Substring**
6. **String to Integer (atoi) / parsing edge cases**
7. **Wildcard Matching / Regex Matching (understand DP approach)**
8. **Encode / Decode Strings (small design)**
9. **Palindrome Partitioning (backtracking / DP)**
10. **Count Anagrams in string / Substring with K distinct, etc.**

Topic 3 — LINKED LISTS (Aug 27 → Sep 3)

Must-do list:

1. **Reverse Linked List (iterative + recursive)**
2. **Detect Cycle (Floyd's cycle) + find start of cycle**
3. **Merge Two Sorted Lists**
4. **Remove Nth Node From End (two-pointer single pass)**
5. **Add Two Numbers (carry + list)**
6. **Merge K Sorted Lists (heap)**
7. **Reverse Nodes in k-Group**
8. **Copy List with Random Pointer (hashmap / interleave trick)**
9. **Intersection of Two Linked Lists**

Topic 4 — TREES & BST (Sep 4 → Sep 11)

Must-do list:

1. **Tree traversals (inorder, preorder, postorder — recursive + iterative)**
2. **Level order (BFS) / Zigzag traversal**
3. **Max Depth / Min Depth**
4. **Validate BST**
5. **Lowest Common Ancestor (BST & BT)**
6. **Serialize / Deserialize Binary Tree**
7. **Path Sum / Root→leaf sum variations**
8. **Balanced Binary Tree (height check)**
9. **Construct tree from inorder+preorder or inorder+postorder**

Topic 5 — GRAPHS (Sep 12 → Sep 19)

Must-do list:

1. **BFS & DFS templates (iterative & recursive)**
2. **Number of Islands (grid BFS/DFS)**
3. **Course Schedule (topological sort / detect cycle in directed)**
4. **Shortest path in unweighted graph (BFS)**
5. **Dijkstra (single-source shortest for weighted)**
6. **Minimum spanning tree basics (Kruskal / Prim) — conceptual**
7. **Union-Find basics (connected components, cycle detect)**
8. **Word Ladder (BFS in word graph)**
9. **Graph DFS backtracking problems (all paths)**

Topic 6 — DYNAMIC PROGRAMMING & GREEDY (Sep 20 → Sep 27)

Must-do list:

1. **Classic DP patterns: memoization + tabulation templates**
2. **Fibonacci / Climbing Stairs (intro DP)**
3. **Longest Increasing Subsequence (Patience sorting O(n log n))**
4. **0/1 Knapsack / Subset sum / Partition equal subset sum**
5. **Coin Change (min coins / count ways)**
6. **Edit Distance (DP on strings)**
7. **Word Break, Decode Ways**
8. **House Robber / DP on arrays (max non-adjacent sum)**
9. **Greedy classics: Activity selection, Jump Game (greedy/DP variants)**

Day 1 – Easy Array Basics

Goal: Warm up with fundamentals & Python syntax

1. **Reverse an array (Python slicing and two-pointer method)**
2. **Find second largest element in an array (without sorting & with sorting)**
3. **Find all duplicates in an array (using dictionary/Counter)**
4. **Check if array is sorted (ascending/descending)**
5. **Rotate an array by `k` positions (right rotation)**

Day 2 – Easy String Basics

Goal: Learn manipulation, parsing, and edge cases

1. **Reverse a string (slicing, loop, recursion)**
2. **Check palindrome string (normal & ignoring spaces/punctuation)**
3. **Count words in a sentence (split method & manual count)**
4. **First non-repeating character in a string**
5. **String compression →** `"aaabbc"` **→** `"a3b2c1"`

Day 3 – Medium Arrays

Goal: Introduce problem-solving patterns

1. **Maximum sum of non-adjacent elements (DP – house robber problem)**
2. **Find missing number from 1 to n (sum formula & XOR)**
3. **Pair sum equals target (two-pointer & hash map)**
4. **Move all zeros to end of array (in-place)**
5. **Kadane's Algorithm – Maximum subarray sum**

Day 4 – Medium Strings

Goal: Manipulation + logic combined

1. **Lexicographically smallest string by removing one character**
2. **Anagram check between two strings**
3. **Longest common prefix from array of strings**
4. **Minimize difference between two equal-length numeric strings by swapping digits**
5. **Sentence with maximum words from paragraph**

Day 5 – Mixed Practice & Mock

Goal: Combine & review

1. **Practice 2 easy + 2 medium problems from arrays & strings without help**
2. **Do timed coding for 60–90 mins (simulate OA)**
3. **Practice explaining your code out loud**
4. **Review time complexity of each problem you solved**

```
# ==== BASICS ====
x, y = 5, "Hi"              # Multiple assign

type(x)                     # Type check → mixing two diff datatypes : int + float → float
int("5") + 2   # ✅ 7
"5" + str(2)   # ✅ "52"


int("5"), str(5), float(5)   # Casting
# comment                     # Single-line comment

# ==== STRINGS ====
s = "Python"
s[0], s[-1], s[1:4]         # Index / Slice
len(s), s.upper(), s.lower()
" ".join(["a", "b"])        # Join list -> "a b"
"hi,there".split(",")       # Split -> ['hi','there']

# ==== LISTS ====
nums = [1, 2, 3]
nums.append(4); nums.pop(); nums.remove(2)
nums[0], nums[1:3]
nums.sort(); nums.reverse()

# ==== TUPLES ====
t = (1, 2, 3)               # Immutable

# ==== SETS ==== # Unique elements
st = {1, 2, 3}
st.add(4);
st.remove(2)

# ==== DICTS ====
d = {"a": 1, "b": 2}
d["a"]; d["c"] = 3
d.keys(); d.values(); d.items()

# ==== LOOPS ====
for i in range(5): print(i)
i = 0
while i < 5: i += 1

# ==== IF-ELSE ====
if x > 5: print("Big")
elif x == 5: print("Equal")
else: print("Small")

# ==== FUNCTIONS ====
def add(a, b): return a + b

# ==== CLASSES ====
class MyClass:
    def __init__(self, val): self.val = val  # Constructor
    def get_val(self): return self.val
obj = MyClass(10); print(obj.get_val())

# ==== EXCEPTIONS ====
try: x = 1/0
except ZeroDivisionError: print("Error")
finally: print("Done")

# ==== FILE HANDLING ====
with open("file.txt", "r") as f: data = f.read()
with open("file.txt", "w") as f: f.write("Hello")

# ==== LIST COMPREHENSION ====
squares = [x**2 for x in range(5)]

# ==== BUILT-INS ====
max(), min(), sum(), sorted(), abs(), round(), zip()

# ==== MODULES ====
import math; math.sqrt(25)
import random; random.randint(1, 10)
import datetime; datetime.datetime.now()

# ==== EXTRA MUST-KNOW ====
for i, v in enumerate(["a","b"]): print(i, v)      # enumerate
double = lambda x: x*2                              # lambda
list(map(lambda x: x+1, [1,2,3]))                   # map
list(filter(lambda x: x>1, [1,2,3]))                # filter
any([0, 1, 0]); all([1, 1, 1])                      # any/all
a, *b = [1, 2, 3, 4]                                # unpack
sq_dict = {x: x**2 for x in range(3)}               # dict comp
sq_set = {x**2 for x in range(3)}                   # set comp


# ==== SLICING====
print(arr[1:4])   # [2, 3, 4]
print(arr[::-1])  # reversed list

# ====Useful functions====
```

```
len(arr)       # length
sum(arr)       # sum of all elements
max(arr)       # largest element
min(arr)       # smallest element
```

```
# ====Iterations====
for x in arr:
    print(x)

for i in range(len(arr)):
    print(arr[i])
```

Python Lists (Arrays in other languages)

- **Creation:**

```
arr = [1, 2, 3, 4, 5]
```

- **Access elements:**

```
print(arr[0])  # first element
print(arr[-1]) # last element
```

- **Slicing:**

```
print(arr[1:4])   # [2, 3, 4]
print(arr[::-1])  # reversed list
```

- **Useful functions:**

```
len(arr)       # length
sum(arr)       # sum of all elements
max(arr)       # largest element
min(arr)       # smallest element
```

- **Iteration:**

```
for x in arr:
    print(x)

for i in range(len(arr)):
    print(arr[i])

nums = [1, 2, 3]
nums.append(4)          # [1, 2, 3, 4]
nums.pop()              # [1, 2, 3]
nums.insert(1, 10)      # [1, 10, 2, 3]
nums.remove(10)         # removes first occurrence
nums.sort()             # in-place sort
nums.sort(reverse=True) # descending
sorted(nums)            # returns new sorted list
nums.reverse()          # reverses list

# List comprehension
squares = [x**2 for x in range(5)]  # [0, 1, 4, 9, 16]
```

---

2. Strings

```
s = "hello"
print(s[0])     # 'h'
print(s[-1])    # 'o'
```

- **Slicing:**

```
print(s[1:4])   # 'ell'
print(s[::-1])  # 'olleh'
```

- **Basic methods:**

```
s.strip()         # remove spaces
s.upper()       # 'HELLO'
s.lower()       # 'hello'
s.split()       # ['hello']→ It breaks a string into parts (substrings) based on spaces.
s.replace("h", "y")  # 'yello'
s.find("lo")      # 3
```

- **Joining:**

```
" ".join(["I", "love", "Python"])  # "I love Python"
```

---

3. Dictionaries (for counting / hash maps)

- **Creation:**

```
my_dict = {"a": 1, "b": 2}
```

- **Access & Update:**

```
my_dict["a"]      # 1
my_dict["c"] = 3  # add new key
print(d.get("a"))   # 1
print(d.keys())     # dict_keys(['a','b','c'])
print(d.values())   # dict_values([1,2,3])
```

- **Loop:**

```
for key, value in my_dict.items():
    print(key, value)
```

- **Counting:** → tells you how many times each item appears

```
from collections import Counter
cnt = Counter([1, 2, 2, 3])
print(cnt)  # Counter({2: 2, 1: 1, 3: 1})

  Create a dictionary
student = {"name": "Komal", "age": 22, "branch": "CSE"}

  Access values
print(student["name"])        # Komal
print(student.get("branch"))  # CSE

  Add or update
student["age"] = 23
student["city"] = "Belagavi"

 merge two dictionaries
dict1 = {"a": 1, "b": 2}
dict2 = {"c": 3, "d": 4}

      # Method 1: Using update()
      dict1.update(dict2)
      print(dict1)  # {'a': 1, 'b': 2, 'c': 3, 'd': 4}

      # Method 2: Using dictionary unpacking
      merged = {**dict1, **dict2}
      print(merged)  # {'a': 1, 'b': 2, 'c': 3, 'd': 4}

# Remove
del student["branch"]

# Loop through keys & values
for key, value in student.items():
    print(key, ":", value)
```

---

4. Sets→ automatically removes duplicates

```
s = {1, 2, 3}
s.add(4)
s.remove(2)
```

```
# Set operations
a = {1, 2, 3}
b = {3, 4, 5}
print(a | b)  # Union -> {1, 2, 3, 4, 5}
print(a & b)  # Intersection -> {3}
print(a - b)  # Difference -> {1, 2}
```

5. Creating from a list of tuples

```
pairs = [("a", 1), ("b", 2)]
print(dict(pairs))  # {'a': 1, 'b': 2}
```

5. Loops

- **For loop:**

```
for i in range(5):
    print(i)
```

- **While loop:**

```
n = 5
while n > 0:
    print(n)
    n -= 1
```

5. Conditionals

```
x = 10
if x > 5:
    print("Greater than 5")
elif x == 5:
    print("Equal to 5")
else:
    print("Less than 5")
```

6. Functions

```
def add(a, b):
    return a + b

result = add(5, 3)
print(result)
```

7. Swapping & Multiple Assignment

```
a, b = 5, 10
a, b = b, a
```

8. List Comprehensions (Shortcuts)

```
squares = [x**2 for x in range(1, 6)]  # [1, 4, 9, 16, 25]
```

9. Enumerate & Zip

```
arr = ["a","b","c"]
for i, val in enumerate(arr):
    print(i, val)
    # Output:
    # 0 a
    # 1 b
    # 2 c

names = ["A","B"]
scores = [90, 80]
for name, score in zip(names, scores):
    print(name, score)
    # Output:
    # A 90
    # B 80
```

10. Lambda Functions

```
add = lambda x, y: x + y
print(add(2, 3))  # Output: 5
```

11. Map, Filter, Reduce

```
nums = [1, 2, 3]
print(list(map(lambda x: x*2, nums)))     # [2, 4, 6] → doubles each element
print(list(filter(lambda x: x%2==0, nums)))  # [2] → keeps only even numbers

from functools import reduce
print(reduce(lambda a, b: a + b, nums))   # 6 → sum of all elements
```

12. Sorting Tricks

```
arr = [(1,2), (3,1), (5,0)]
arr.sort(key=lambda x: x[1])
# Sort by 2nd element → arr = [(5, 0), (3, 1), (1, 2)]
```

13. List Slicing

```
nums = [0, 1, 2, 3, 4]
print(nums[1:4])   # [1, 2, 3] → from index 1 to 3
print(nums[::-1])  # [4, 3, 2, 1, 0] → reversed list
```

14. Handling Input/Output

```
# Single integer input:
# x = int(input())  # Example: input=5 → x=5

# Two integers:
# a, b = map(int, input().split())  # Example: input="4 7" → a=4, b=7

# List of integers:
# arr = list(map(int, input().split()))  # Example: input="1 2 3" → arr=[1,2,3]
```

15. Exception Handling

```
try:
    x = 1 / 0
except ZeroDivisionError:
    print("Cannot divide by zero")  # Output: Cannot divide by zero
finally:
    print("Done")  # Output: Done
```

16. Common Interview Snippets

```
# Frequency count

from collections import Counter
cnt = Counter([1, 1, 2, 3])
print(cnt)
# Output: Counter({1: 2, 2: 1, 3: 1}) → frequency of each element

# Prefix sum

nums = [1, 2, 3, 4]
prefix = [0] * len(nums)
prefix[0] = nums[0]
for i in range(1, len(nums)):
    prefix[i] = prefix[i-1] + nums[i]
print(prefix)
# Output: [1, 3, 6, 10] → running sum of elements

# Swapping variables
a, b = 5, 10
a, b = b, a
print(a, b)  # Output: 10 5

# Reversing a string

s = "abc"
print(s[::-1])  # Output: cba
```

17. Important Built-ins

```
len(), sum(), max(), min(), sorted(), list(), set(), dict()
```

```python
# 1. len() → Returns the number of items in an object
nums = [1, 2, 3, 4]
print(len(nums))  # 4

# 2. sum() → Returns the sum of elements
nums = [10, 20, 30]
print(sum(nums))  # 60
print(sum(nums, 5))  # 65 (adds starting value)

# 3. max() → Returns the largest element
nums = [3, 8, 1, 6]
print(max(nums))  # 8
print(max("hello"))  # 'o' (based on ASCII value)

# 4. min() → Returns the smallest element
nums = [3, 8, 1, 6]
print(min(nums))  # 1
print(min("hello"))  # 'e'

# 5. sorted() → Returns a new sorted list (original unchanged)
nums = [5, 1, 3, 2]
print(sorted(nums))  # [1, 2, 3, 5]
print(sorted(nums, reverse=True))  # [5, 3, 2, 1]

# 6. list() → Converts iterable into a list
s = "abc"
print(list(s))  # ['a', 'b', 'c']

# 7. set() → Creates a set (removes duplicates)
nums = [1, 2, 2, 3, 1]
print(set(nums))  # {1, 2, 3}

# 8. dict() → Creates a dictionary
print(dict(name="Alice", age=25))
# {'name': 'Alice', 'age': 25}
```

NOTES

1.`end=" "` **means they are printed on the same line with spaces**

2. `!=` **→ not equal to one value→ use it to compare with one value only. Ex:** `if color != 'red':`

`not in` **→ not inside many values → use it to check against many values at once.**
Ex: `if fruit not in ['apple', 'orange' , 'banana']:`

3. `+=` **→ Adding many things (another list) at once:** `result += [item1, item2]`
`append()` **→ Adding one thing**

4. `join()` **works only with strings.**