**1. What is an array?**

**Answer:**  
An array is a linear data structure that stores elements of the **same data type** in **contiguous memory locations**. It allows **random access** using indices.

**2. What are the advantages of arrays?**

**Answer:**

* Fast access using index (O(1))
* Easy to implement
* Memory-efficient for storing elements of the same type

**3. What are the disadvantages of arrays?**

**Answer:**

* Fixed size (static in nature)
* Insertion/deletion is expensive (O(n))
* Wastes memory if array is not fully used

**4. What is the time complexity of basic array operations?**

| **Operation** | **Time Complexity** |
| --- | --- |
| Access by index | O(1) |
| Search (unsorted) | O(n) |
| Search (sorted) | O(log n) with Binary Search |
| Insertion at end | O(1) (if space exists) |
| Insertion at start/middle | O(n) |
| Deletion by value | O(n) |

**5. How is an array different from a linked list?**

| **Feature** | **Array** | **Linked List** |
| --- | --- | --- |
| Size | Fixed | Dynamic |
| Memory | Contiguous | Non-contiguous |
| Access time | O(1) | O(n) |
| Insertion/Deletion | O(n) | O(1) (at head/tail with pointers) |

**6. What is a dynamic array?**

**Answer:**  
A dynamic array (like ArrayList in Java or vector in C++) is a resizable array that can grow/shrink as needed. Internally, it creates a new larger array and copies elements when resized.

**7. Can arrays store different data types?**

**Answer:**  
No, arrays store **elements of the same data type**. If needed, you can use object arrays (Object[]) to store mixed types, but that’s generally avoided.

**8. What happens if you access an index beyond array size?**

**Answer:**  
It throws a **runtime error** (like ArrayIndexOutOfBoundsException in Java).

**9. What is meant by array traversal?**

**Answer:**  
Traversing an array means visiting each element of the array **once**, typically using a loop (like for or while).

**10. How are 1D and 2D arrays different?**

**Answer:**

* **1D array**: A linear list (int[] arr = new int[5];)
* **2D array**: An array of arrays (matrix-like structure) (int[][] matrix = new int[3][3];)

### ****1. Reverse an Array****

**Q:** Reverse the given array.  
**Answer:**

java

Copy

public static void reverseArray(int[] arr) {

int left = 0, right = arr.length - 1;

while (left < right) {

int temp = arr[left];

arr[left] = arr[right];

arr[right] = temp;

left++;

right--;

}

}

**Explanation:**

* Uses **two-pointer** approach to swap elements from start and end.
* **Time:** O(n) (single pass), **Space:** O(1) (in-place).

### ****2. Find the Largest Element****

**Q:** Find the largest element in an array.  
**Answer:**

java

Copy

public static int findLargest(int[] arr) {

int max = arr[0];

for (int num : arr) {

if (num > max) max = num;

}

return max;

}

**Explanation:**

* Iterates once to track the maximum.
* **Time:** O(n), **Space:** O(1).

### ****3. Find the Second Largest Element****

**Q:** Find the second largest element in an array.  
**Answer:**

java

Copy

public static int secondLargest(int[] arr) {

int first = Integer.MIN\_VALUE, second = Integer.MIN\_VALUE;

for (int num : arr) {

if (num > first) {

second = first;

first = num;

} else if (num > second && num != first) {

second = num;

}

}

return (second != Integer.MIN\_VALUE) ? second : -1;

}

**Explanation:**

* Tracks the top two elements in a single pass.
* **Time:** O(n), **Space:** O(1).

### ****4. Check if Array is Sorted****

**Q:** Check if the array is sorted in ascending order.  
**Answer:**

java

Copy

public static boolean isSorted(int[] arr) {

for (int i = 0; i < arr.length - 1; i++) {

if (arr[i] > arr[i + 1]) return false;

}

return true;

}

**Explanation:**

* Checks if every element is ≤ the next one.
* **Time:** O(n), **Space:** O(1).

### ****5. Remove Duplicates from Sorted Array****

**Q:** Remove duplicates in-place (return new length).  
**Answer:**

java

Copy

public static int removeDuplicates(int[] arr) {

if (arr.length == 0) return 0;

int k = 1;

for (int i = 1; i < arr.length; i++) {

if (arr[i] != arr[i - 1]) {

arr[k++] = arr[i];

}

}

return k;

}

**Explanation:**

* Uses a **read-pointer (**i**)** and **write-pointer (**k**)**.
* **Time:** O(n), **Space:** O(1).

### ****6. Move All Zeros to End****

**Q:** Move zeros to the end while maintaining the order of non-zero elements.  
**Answer:**

java

Copy

public static void moveZeros(int[] arr) {

int k = 0;

for (int num : arr) {

if (num != 0) arr[k++] = num;

}

while (k < arr.length) arr[k++] = 0;

}

**Explanation:**

* Overwrites non-zero elements first, then fills the rest with zeros.
* **Time:** O(n), **Space:** O(1).

### ****7. Find Missing Number (1 to N)****

**Q:** Given an array of size N-1 with numbers from 1 to N, find the missing number.  
**Answer:**

java

Copy

public static int missingNumber(int[] arr) {

int n = arr.length + 1;

int expectedSum = n \* (n + 1) / 2;

int actualSum = 0;

for (int num : arr) actualSum += num;

return expectedSum - actualSum;

}

**Explanation:**

* Uses the **sum of first N natural numbers** formula.
* **Time:** O(n), **Space:** O(1).

### ****8. Find the Single Number (All Others Twice)****

**Q:** Every number appears twice except one. Find it.  
**Answer:**

java

Copy

public static int singleNumber(int[] arr) {

int xor = 0;

for (int num : arr) xor ^= num;

return xor;

}

**Explanation:**

* **XOR** cancels out duplicates (a ^ a = 0).
* **Time:** O(n), **Space:** O(1).

### ****9. Rotate Array by K Steps****

**Q:** Rotate the array right by k steps.  
**Answer:**

java

Copy

public static void rotate(int[] arr, int k) {

k %= arr.length;

reverse(arr, 0, arr.length - 1);

reverse(arr, 0, k - 1);

reverse(arr, k, arr.length - 1);

}

private static void reverse(int[] arr, int start, int end) {

while (start < end) {

int temp = arr[start];

arr[start] = arr[end];

arr[end] = temp;

start++;

end--;

}

}

**Explanation:**

* **Reverse entire array → Reverse first**k**→ Reverse rest.**
* **Time:** O(n), **Space:** O(1).

### ****10. Find All Duplicates in Array****

**Q:** Given 1 ≤ a[i] ≤ N, return all duplicates.  
**Answer:**

java

Copy

public static List<Integer> findDuplicates(int[] arr) {

List<Integer> res = new ArrayList<>();

for (int i = 0; i < arr.length; i++) {

int index = Math.abs(arr[i]) - 1;

if (arr[index] < 0) res.add(Math.abs(arr[i]));

else arr[index] = -arr[index];

}

return res;

}

**Explanation:**

* Uses **negative marking** to track seen elements.
* **Time:** O(n), **Space:** O(1) (output excluded).

### ****11. Two Sum (Find Pair with Target Sum)****

**Q:** Find two numbers that add up to target.  
**Answer:**

java

Copy

public static int[] twoSum(int[] arr, int target) {

Map<Integer, Integer> map = new HashMap<>();

for (int i = 0; i < arr.length; i++) {

int complement = target - arr[i];

if (map.containsKey(complement)) {

return new int[]{map.get(complement), i};

}

map.put(arr[i], i);

}

return new int[]{-1, -1};

}

**Explanation:**

* Uses a **hashmap** to store complements.
* **Time:** O(n), **Space:** O(n).

### ****12. Maximum Subarray Sum (Kadane’s Algorithm)****

**Q:** Find the contiguous subarray with the largest sum.  
**Answer:**

java

Copy

public static int maxSubArray(int[] arr) {

int maxSum = arr[0], currentSum = arr[0];

for (int i = 1; i < arr.length; i++) {

currentSum = Math.max(arr[i], currentSum + arr[i]);

maxSum = Math.max(maxSum, currentSum);

}

return maxSum;

}

**Explanation:**

* **Kadane’s Algorithm** tracks the best subarray ending at i.
* **Time:** O(n), **Space:** O(1).

### ****13. Merge Two Sorted Arrays****

**Q:** Merge two sorted arrays into one sorted array.  
**Answer:**

java

Copy

public static int[] mergeSortedArrays(int[] arr1, int[] arr2) {

int[] merged = new int[arr1.length + arr2.length];

int i = 0, j = 0, k = 0;

while (i < arr1.length && j < arr2.length) {

merged[k++] = (arr1[i] < arr2[j]) ? arr1[i++] : arr2[j++];

}

while (i < arr1.length) merged[k++] = arr1[i++];

while (j < arr2.length) merged[k++] = arr2[j++];

return merged;

}

**Explanation:**

* Uses **two-pointer** technique to merge in O(n + m).
* **Time:** O(n + m), **Space:** O(n + m).

### ****14. Find the Union of Two Arrays****

**Q:** Return the union of two arrays (distinct elements).  
**Answer:**

java

Copy

public static List<Integer> findUnion(int[] arr1, int[] arr2) {

Set<Integer> set = new HashSet<>();

for (int num : arr1) set.add(num);

for (int num : arr2) set.add(num);

return new ArrayList<>(set);

}

**Explanation:**

* **HashSet** automatically handles uniqueness.
* **Time:** O(n + m), **Space:** O(n + m).

### ****15. Find the Intersection of Two Arrays****

**Q:** Return common elements in two arrays.  
**Answer:**

java

Copy

public static List<Integer> findIntersection(int[] arr1, int[] arr2) {

Set<Integer> set = new HashSet<>();

List<Integer> res = new ArrayList<>();

for (int num : arr1) set.add(num);

for (int num : arr2) {

if (set.contains(num)) {

res.add(num);

set.remove(num); // Avoid duplicates

}

}

return res;

}

**Explanation:**

* Uses a **HashSet** to track seen elements.
* **Time:** O(n + m), **Space:** O(min(n, m)).

### ****16. Find the First Non-Repeating Element****

**Q:** Find the first element that appears only once.  
**Answer:**

java

Copy

public static int firstNonRepeating(int[] arr) {

Map<Integer, Integer> freq = new LinkedHashMap<>();

for (int num : arr) freq.put(num, freq.getOrDefault(num, 0) + 1);

for (Map.Entry<Integer, Integer> entry : freq.entrySet()) {

if (entry.getValue() == 1) return entry.getKey();

}

return -1;

}

**Explanation:**

* **LinkedHashMap** preserves insertion order.
* **Time:** O(n), **Space:** O(n).

### ****17. Find All Triplets with Zero Sum****

**Q:** Find all triplets (a, b, c) such that a + b + c = 0.  
**Answer:**

java

Copy

public static List<List<Integer>> threeSum(int[] arr) {

Arrays.sort(arr);

List<List<Integer>> res = new ArrayList<>();

for (int i = 0; i < arr.length - 2; i++) {

if (i > 0 && arr[i] == arr[i - 1]) continue; // Skip duplicates

int left = i + 1, right = arr.length - 1;

while (left < right) {

int sum = arr[i] + arr[left] + arr[right];

if (sum == 0) {

res.add(Arrays.asList(arr[i], arr[left], arr[right]));

while (left < right && arr[left] == arr[left + 1]) left++; // Skip duplicates

while (left < right && arr[right] == arr[right - 1]) right--;

left++;

right--;

} else if (sum < 0) left++;

else right--;

}

}

return res;

}

**Explanation:**

* **Sort + Two-pointer** approach.
* **Time:** O(n²), **Space:** O(1) (ignoring output).

### ****18. Maximum Product Subarray****

**Q:** Find the contiguous subarray with the largest product.  
**Answer:**

java

Copy

public static int maxProduct(int[] arr) {

int maxProd = arr[0], minProd = arr[0], res = arr[0];

for (int i = 1; i < arr.length; i++) {

if (arr[i] < 0) {

int temp = maxProd;

maxProd = minProd;

minProd = temp;

}

maxProd = Math.max(arr[i], maxProd \* arr[i]);

minProd = Math.min(arr[i], minProd \* arr[i]);

res = Math.max(res, maxProd);

}

return res;

}

**Explanation:**

* Tracks **max and min products** (since negatives can flip min to max).
* **Time:** O(n), **Space:** O(1).

### ****19. Longest Consecutive Sequence****

**Q:** Find the length of the longest consecutive elements sequence.  
**Answer:**

java

Copy

public static int longestConsecutive(int[] arr) {

Set<Integer> set = new HashSet<>();

for (int num : arr) set.add(num);

int maxStreak = 0;

for (int num : set) {

if (!set.contains(num - 1)) { // Start of a sequence

int current = num, streak = 1;

while (set.contains(current + 1)) {

current++;

streak++;

}

maxStreak = Math.max(maxStreak, streak);

}

}

return maxStreak;

}

**Explanation:**

* **HashSet** for O(1) lookups, checks for sequence starts.
* **Time:** O(n), **Space:** O(n).

### ****20. Container with Most Water****

**Q:** Find two lines that form the largest container (max area).  
**Answer:**

java

Copy

public static int maxArea(int[] height) {

int left = 0, right = height.length - 1, maxArea = 0;

while (left < right) {

int area = Math.min(height[left], height[right]) \* (right - left);

maxArea = Math.max(maxArea, area);

if (height[left] < height[right]) left++;

else right--;

}

return maxArea;

}

**Explanation:**

* **Two-pointer** approach, moves the shorter line inward.
* **Time:** O(n), **Space:** O(1).

### ****Summary of Patterns Covered:****

1. **Two-pointer** (Reverse, Two Sum, Container Water).
2. **Sliding Window** (Max Subarray).
3. **HashSet/HashMap** (Duplicates, Union, Intersection).
4. **Sorting** (Three Sum, Merge Sorted Arrays).
5. **In-place Modification** (Move Zeros, Remove Duplicates).
6. **Math Tricks** (Missing Number, Single Number).

These cover **90% of fresher array interview questions** with **optimal time/space**! 🚀