**### \*\*1. Data Structures and Algorithms (DSA)\*\***

**#### \*\*Basic Concepts\*\***

**Basic DSA Concepts: Arrays**

Arrays are a foundational data structure in computer science. They store a fixed-size sequence of elements of the same type, indexed starting from zero. Arrays can be categorized into **Static** and **Dynamic** arrays, and their operations include inserting, deleting, traversing, and more.

**1. Static vs. Dynamic Arrays**

|  |  |  |
| --- | --- | --- |
| **Type** | **Description** | **Example** |
| **Static Arrays** | Arrays with a fixed size. Once declared, their size cannot be changed. | int arr[10]; |
| **Dynamic Arrays** | Arrays whose size can grow or shrink during runtime. Allocated dynamically. | let arr = []; (JavaScript) |

**2. Array Operations**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Description** | **Time Complexity** | **Example Code (PHP, JS)** |
| **Insertion** | Add an element at a specific position. | O(n) (if shifting is needed) | **PHP:** $arr[1] = 10; **JS:** arr.splice(1, 0, 10); |
| **Deletion** | Remove an element from a specific position. | O(n) (if shifting is needed) | **PHP:** unset($arr[1]); **JS:** arr.splice(1, 1); |
| **Access** | Access an element using its index. | O(1) | **PHP:** $arr[1]; **JS:** arr[1]; |
| **Traversal** | Visit each element in the array. | O(n) | See examples below. |

**3. Multi-dimensional Arrays**

Multi-dimensional arrays are arrays containing other arrays as their elements. These are commonly used for grid or matrix-like data.

|  |  |
| --- | --- |
| **Type** | **Example** |
| **2D Array (Matrix)** | PHP: $arr = [[1, 2], [3, 4]]; JS: let arr = [[1, 2], [3, 4]]; |
| **3D Array** | PHP: $arr = [[[1], [2]], [[3], [4]]]; JS: let arr = [[[1], [2]], [[3], [4]]]; |

**When to Use Multi-dimensional Arrays:**

* **Grids or Matrices**: Representing game boards, image pixels, or tabular data.
* **Complex Structures**: Storing relationships like a graph’s adjacency matrix.

**4. Time and Space Complexity**

**Definitions:**

|  |  |  |
| --- | --- | --- |
| **Term** | **Definition** | **Example** |
| **Time Complexity** | The number of operations required to execute an algorithm, relative to input size n. | O(1), O(n), O(n²) |
| **Space Complexity** | The amount of memory required by the algorithm, including input and auxiliary space. | O(1), O(n) |

**5. Calculating Complexity with Real-life Examples**

**Basic Example: Traversing an Array**

**Problem:** Traverse an array of integers and print each element.

**PHP:**

$arr = [1, 2, 3, 4];

foreach ($arr as $value) {

echo $value . "\n";

}

**JS:**

let arr = [1, 2, 3, 4];

arr.forEach(value => console.log(value));

* **Time Complexity:** O(n) (You visit each element once)
* **Space Complexity:** O(1) (No extra space used)

**Intermediate Example: Inserting into a Sorted Array**

**Problem:** Insert an element into a sorted array while maintaining order.

**PHP:**

$arr = [1, 3, 5];

array\_splice($arr, 1, 0, 2); // Insert 2 at index 1

print\_r($arr);

**JS:**

let arr = [1, 3, 5];

arr.splice(1, 0, 2); // Insert 2 at index 1

console.log(arr);

* **Time Complexity:** O(n) (Shifting elements after the insertion point)
* **Space Complexity:** O(1) (In-place modification)

**Advanced Example: Finding Pairs with a Given Sum**

**Problem:** Find all pairs in an array that add up to a target sum.

**PHP:**

function findPairs($arr, $target) {

$map = [];

foreach ($arr as $num) {

$complement = $target - $num;

if (isset($map[$complement])) {

echo "Pair: ($complement, $num)\n";

}

$map[$num] = true;

}

}

findPairs([1, 2, 3, 4], 5);

**JS:**

function findPairs(arr, target) {

let map = {};

for (let num of arr) {

let complement = target - num;

if (map[complement]) {

console.log(`Pair: (${complement}, ${num})`);

}

map[num] = true;

}

}

findPairs([1, 2, 3, 4], 5);

* **Time Complexity:** O(n) (Each element is processed once)
* **Space Complexity:** O(n) (Hashmap stores complements)

**6. Summary Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Concept** | **Example** | **Time Complexity** | **Space Complexity** |
| **Static Array Access** | arr[2]; | O(1) | O(1) |
| **Dynamic Array Resize** | Append element | O(n) | O(n) |
| **Traversal** | Print all elements | O(n) | O(1) |
| **Search (Unsorted)** | Linear search | O(n) | O(1) |
| **Search (Sorted)** | Binary search | O(log n) | O(1) |
| **Insertion (Middle)** | Insert at index | O(n) | O(1) |
| **Deletion (Middle)** | Remove at index | O(n) | O(1) |

By understanding the **basic operations**, **complexity calculations**, and real-life examples, arrays can be efficiently used in various programming scenarios. Let me know if you'd like more examples or specific questions to explore!

**Day 1-3: Basic Array Operations & Introduction**

1. **Topic**: Introduction to Arrays
   * **Problem**: Array basics – creation, initialization, and basic operations (insert, delete, traverse).
   * **Practice**:
     + **Problem 1**: Find the largest and smallest element in an array.
     + **Problem 2**: Implement linear search in an array.
     + **Problem 3**: Reverse an array in place.
2. **Topic**: Array Traversal and Manipulation
   * **Problem**: Accessing elements, modifying elements.
   * **Practice**:
     + **Problem 4**: Find the frequency of each element in an array.
     + **Problem 5**: Implement binary search on a sorted array.

**Day 4-6: Sorting Algorithms and Techniques**

1. **Topic**: Basic Sorting Algorithms
   * **Problem**: Bubble sort, selection sort, and insertion sort.
   * **Practice**:
     + **Problem 6**: Sort an array using bubble sort.
     + **Problem 7**: Sort an array using selection sort.
2. **Topic**: Advanced Sorting Techniques
   * **Problem**: Quick sort and merge sort.
   * **Practice**:
     + **Problem 8**: Implement quicksort algorithm.
     + **Problem 9**: Implement merge sort algorithm.

**Day 7-9: Array Searching and Windowing Techniques**

1. **Topic**: Searching Techniques
   * **Problem**: Binary search and search variations.
   * **Practice**:
     + **Problem 10**: Find the first and last occurrence of a target element in a sorted array.
     + **Problem 11**: Find the number of occurrences of an element in a sorted array.
2. **Topic**: Sliding Window Technique
   * **Problem**: Finding the maximum sum subarray of size k.
   * **Practice**:
     + **Problem 12**: Find the maximum sum of a subarray of size k.
     + **Problem 13**: Find the minimum sum of a subarray of size k.

**Day 10-12: Advanced Array Problems**

1. **Topic**: Two Pointer Technique
   * **Problem**: Find pairs with a given sum.
   * **Practice**:
     + **Problem 14**: Find two elements in a sorted array whose sum equals a given target.
     + **Problem 15**: Move all zeros to the end of an array.
2. **Topic**: Dynamic Programming & Greedy Techniques
   * **Problem**: Solving problems with optimization strategies.
   * **Practice**:
     + **Problem 16**: Find the maximum product of two numbers in an array.
     + **Problem 17**: Find the longest increasing subsequence in an array.

**Day 13: Recap and Final Review**

1. **Topic**: Review All Array Concepts
   * **Problem**: Solve a mixed set of problems to test all array-related knowledge.
   * **Practice**:
     + **Problem 18**: Find the intersection of two arrays.
     + **Problem 19**: Find the union of two arrays.
     + **Problem 20**: Rotate an array by k positions.

**GFG 160 Problems**

While you work through these problems, you should also refer to **GeeksforGeeks** (GFG) for the **160 DSA problems**. Here's a list of topics from GFG 160 to guide you:

* **Arrays**:
  + Subarray Sum Problem
  + Largest Sum Contiguous Subarray
  + Sorting an Array
  + Reversing an Array
  + Rearranging an Array
* **Searching and Sorting**:
  + Binary Search
  + Merge Sort
  + Quick Sort
* **String Manipulation**:
  + Palindrome Check
  + Anagram Check
  + String Reversal