10 Beginner-Friendly DSA Questions to Kickstart Your Journey

1. Find the Maximum Element in an Array

Description: Given an array of integers, find the maximum element.

Explanation: This problem introduces basic array traversal using a loop.

Time & Space Complexity: Time: O(n), Space: O(1)

[C++ Code]

```
int findMax(int arr[], int n) {
    int maxVal = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > maxVal) maxVal = arr[i];
    }
    return maxVal;
}
```

[Java Code]

```
int findMax(int[] arr) {
    int maxVal = arr[0];
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] > maxVal) maxVal = arr[i];
    }
    return maxVal;
}
```

[Python Code]

```
def find_max(arr):
    max_val = arr[0]
    for num in arr[1:]:
        if num > max_val:
            max_val = num
    return max_val
```

10 Beginner-Friendly DSA Questions to Kickstart Your Journey

2. Check if a Number is Prime

Description: Check whether a given number is prime or not.

Explanation: This builds logical thinking with conditionals and loops.

Time & Space Complexity: Time: O(sqrt(n)), Space: O(1)

[C++ Code]

```
bool isPrime(int n) {
    if (n <= 1) return false;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) return false;
    }
    return true;
}
```

[Java Code]

```
boolean isPrime(int n) {
    if (n <= 1) return false;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) return false;
    }
    return true;
}
```

[Python Code]

```
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True
```

10 Beginner-Friendly DSA Questions to Kickstart Your Journey

3. Reverse an Array

Description: Given an array, reverse the elements in-place.

Explanation: Teaches two-pointer technique and array manipulation.

Time & Space Complexity: Time: O(n), Space: O(1)

[C++ Code]

```
void reverseArray(int arr[], int n) {
    int start = 0, end = n - 1;
    while (start < end) {
        swap(arr[start], arr[end]);
        start++;
        end--;
    }
}
```

[Java Code]

```
void reverseArray(int[] arr) {
    int start = 0, end = arr.length - 1;
    while (start < end) {
        int temp = arr[start];
        arr[start++] = arr[end];
        arr[end--] = temp;
    }
}
```

[Python Code]

```
def reverse_array(arr):
    start, end = 0, len(arr) - 1
    while start < end:
        arr[start], arr[end] = arr[end], arr[start]
        start += 1
```

```
        end -= 1
```

4. Check Palindrome String

Description: Check whether a given string is a palindrome.

Explanation: Helps in understanding strings and two-pointer technique.

Time & Space Complexity: Time: O(n), Space: O(1)

[C++ Code]

```
bool isPalindrome(string s) {
    int start = 0, end = s.length() - 1;
    while (start < end) {
        if (s[start++] != s[end--]) return false;
    }
    return true;
}
```

[Java Code]

```
boolean isPalindrome(String s) {
    int start = 0, end = s.length() - 1;
    while (start < end) {
        if (s.charAt(start++) != s.charAt(end--)) return false;
    }
    return true;
}
```

[Python Code]

```
def is_palindrome(s):
    return s == s[::-1]
```

5. Count Frequency of Elements

Description: Count the frequency of each element in an array.

Explanation: Introduces hash maps or dictionaries for counting.

Time & Space Complexity: Time: O(n), Space: O(n)

[C++ Code]

```
#include <unordered_map>
void countFrequency(int arr[], int n) {
    unordered_map<int, int> freq;
    for (int i = 0; i < n; i++) freq[arr[i]]++;
}
```

[Java Code]

```
void countFrequency(int[] arr) {
    HashMap<Integer, Integer> freq = new HashMap<>();
    for (int num : arr) freq.put(num, freq.getOrDefault(num, 0) + 1);
}
```

[Python Code]

```
from collections import Counter

def count_frequency(arr):
    return Counter(arr)
```

6. Linear Search in Array

Description: Find if a target value exists in an array.

Explanation: A basic search problem introducing loops and conditionals.

Time & Space Complexity: Time: O(n), Space: O(1)

[C++ Code]

```
bool linearSearch(int arr[], int n, int x) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == x) return true;
```

```
    }
    return false;
}
```

[Java Code]

```
boolean linearSearch(int[] arr, int x) {
    for (int num : arr) {
        if (num == x) return true;
    }
    return false;
}
```

[Python Code]

```
def linear_search(arr, x):
    for num in arr:
        if num == x:
            return True
    return False
```

7. Sum of All Elements in Array

Description: Calculate the sum of all elements in an array.

Explanation: Builds confidence with loop basics and accumulator logic.

Time & Space Complexity: Time: O(n), Space: O(1)

[C++ Code]

```
int arraySum(int arr[], int n) {
    int sum = 0;
    for (int i = 0; i < n; i++) sum += arr[i];
    return sum;
}
```

[Java Code]

```
int arraySum(int[] arr) {
    int sum = 0;
    for (int num : arr) sum += num;
    return sum;
}
```

[Python Code]

```
def array_sum(arr):
    return sum(arr)
```

8. Factorial of a Number

Description: Find the factorial of a number using loop.

Explanation: Strengthens loop concept and recursion introduction.

Time & Space Complexity: Time: O(n), Space: O(1)

[C++ Code]

```
int factorial(int n) {
    int result = 1;
    for (int i = 2; i <= n; i++) result *= i;
    return result;
}
```

[Java Code]

```
int factorial(int n) {
    int result = 1;
    for (int i = 2; i <= n; i++) result *= i;
    return result;
}
```

[Python Code]

```
def factorial(n):
    result = 1
    for i in range(2, n+1):
        result *= i
    return result
```

9. Fibonacci Series up to N Terms

Description: Print Fibonacci series up to N terms.

Explanation: Introduces sequence generation using loops.

Time & Space Complexity: Time: O(n), Space: O(1)

[C++ Code]

```
void fibonacci(int n) {
    int a = 0, b = 1;
    for (int i = 0; i < n; i++) {
        cout << a << " ";
        int next = a + b;
        a = b;
        b = next;
    }
}
```

[Java Code]

```
void fibonacci(int n) {
    int a = 0, b = 1;
    for (int i = 0; i < n; i++) {
        System.out.print(a + " ");
        int next = a + b;
        a = b;
        b = next;
    }
```

```
}
```

[Python Code]

```
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        print(a, end=' ')
        a, b = b, a + b
```