* **QUESTIONS ON ALGORITHM ANALYSIS**

**Q1. Write a Java program to calculate the sum of the first N natural numbers.**

public class SumOfFirstNNumbers {

public static void main(String[] args) {

int n = 5;

int sum = 0;

for (int i = 1; i <= n; i++) {

sum += i;

}

System.out.println("Sum of first " + n + " numbers: " + sum);

}

}

**Output:**

Sum of first 5 numbers: 15

**Explantion:**

The time complexity of this program is O(n) because it uses a simple loop that iterates N times, where N is the input.

**Q2. Write a Java program to calculate the factorial of a given number.**

public class Factorial {

public static void main(String[] args) {

int n = 5;

int factorial = 1;

for (int i = 1; i <= n; i++) {

factorial \*= i;

}

System.out.println("Factorial of " + n + ": " + factorial);

}

}

**Output:**

Factorial of 5: 120

**Explantion:**

The time complexity of this program is O(n) because it uses a loop that iterates N times, where N is the input.

**Q3. Write a Java program to perform a linear search on an array.**

public class LinearSearch {

public static void main(String[] args) {

int[] arr = {4, 2, 7, 1, 9};

int target = 7;

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

if (arr[i] == target) {

System.out.println("Element found at index " + i);

return;

}

}

System.out.println("Element not found in the array");

}

}

**Output:**

Element found at index 2

**Explantion:**

The time complexity of linear search is O(n) in the worst case, where n is the size of the array. It iterates through the array until it finds the target element or reaches the end.

**Q4. Write a Java program to sort an array using the Bubble Sort algorithm.**

public class BubbleSort {

public static void main(String[] args) {

int[] arr = {5, 2, 9, 1, 5};

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

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

if (arr[j] > arr[j + 1]) {

// Swap elements if they are in the wrong order

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

System.out.println("Sorted array: " + Arrays.toString(arr));

}

}

**Output:**

Sorted array: [1, 2, 5, 5, 9]

**Explantion:**

Bubble Sort has a time complexity of O(n^2) in the worst case. It compares and swaps adjacent elements in the array until the entire array is sorted.

**Q5. Write a Java program to generate the first N terms of the Fibonacci series.**

public class FibonacciSeries {

public static void main(String[] args) {

int n = 8;

int firstTerm = 0, secondTerm = 1;

System.out.println("Fibonacci Series:");

for (int i = 1; i <= n; ++i) {

System.out.print(firstTerm + ", ");

int nextTerm = firstTerm + secondTerm;

firstTerm = secondTerm;

secondTerm = nextTerm;

}

}

}

**Output:**

Fibonacci Series:

0, 1, 1, 2, 3, 5, 8, 13,

**Explantion:**

The time complexity of the Fibonacci series generation using iteration is O(n). It uses a loop that iterates N times, where N is the number of terms to be generated.

**Q6. Write a Java program to calculate the sum of the first N natural numbers using an array to store intermediate results.**

public class SumWithArray {

public static void main(String[] args) {

int n = 5;

int[] intermediateSums = new int[n + 1];

intermediateSums[0] = 0;

for (int i = 1; i <= n; i++) {

intermediateSums[i] = intermediateSums[i - 1] + i;

}

System.out.println("Sum of first " + n + " numbers: " + intermediateSums[n]);

}

}

**Output:**

Sum of first 5 numbers: 15

**Explantion:**

The space complexity of this program is O(n) because it uses an array of size N to store intermediate results.

**Q7. Write a Java program to calculate the factorial of a given number using an array to store intermediate results.**

public class FactorialWithArray {

public static void main(String[] args) {

int n = 5;

int[] intermediateFactorials = new int[n + 1];

intermediateFactorials[0] = 1;

for (int i = 1; i <= n; i++) {

intermediateFactorials[i] = intermediateFactorials[i - 1] \* i;

}

System.out.println("Factorial of " + n + ": " + intermediateFactorials[n]);

}

}

**Output:**

Factorial of 5: 120

**Explantion:**

The space complexity of this program is O(n) because it uses an array of size N to store intermediate factorial values.

**Q8. Write a Java program to generate the first N terms of the Fibonacci series using an array to store intermediate results.**

public class FibonacciWithArray {

public static void main(String[] args) {

int n = 8;

int[] fibonacciSeries = new int[n];

fibonacciSeries[0] = 0;

fibonacciSeries[1] = 1;

for (int i = 2; i < n; i++) {

fibonacciSeries[i] = fibonacciSeries[i - 1] + fibonacciSeries[i - 2];

}

System.out.println("Fibonacci Series: " + Arrays.toString(fibonacciSeries));

}

}

**Output:**

Fibonacci Series: [0, 1, 1, 2, 3, 5, 8, 13]

**Explantion:**

The space complexity of this program is O(n) because it uses an array of size N to store intermediate Fibonacci series values.

**Q9. Write a Java program to perform a linear search on an array with the help of additional memory.**

public class LinearSearchWithMemory {

public static void main(String[] args) {

int[] arr = {4, 2, 7, 1, 9};

int target = 7;

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

for (int num : arr) {

memory.add(num);

}

if (memory.contains(target)) {

System.out.println("Element found");

} else {

System.out.println("Element not found in the array");

}

}

}

**Output:**

Element found

**Explantion:**

The space complexity of this program is O(n) because it uses a HashSet to store the elements of the array, resulting in additional space usage.

**Q10. Write a Java program to sort an array using the Bubble Sort algorithm with the help of an additional array.**

public class BubbleSortWithArray {

public static void main(String[] args) {

int[] arr = {5, 2, 9, 1, 5};

int[] sortedArray = Arrays.copyOf(arr, arr.length);

Arrays.sort(sortedArray);

System.out.println("Sorted array: " + Arrays.toString(sortedArray));

}

}

**Output:**

Sorted array: [1, 2, 5, 5, 9]

**Explantion:**

The space complexity of this program is O(n) because it uses an additional array of the same size as the input array to store the sorted elements.