Activities

1. a. In this fragment, the outer loop runs ‘n’ times, and for each iteration of the outer loop, the inner loop runs ‘n’ times. So, the output statement is executed ‘n * n = n^2’ times. This algorithm is O(n^2).

   b. In this fragment, the outer loop runs ‘n’ times, and for each iteration of the outer loop, the inner loop runs twice. So, the output statement is executed ‘n * 2 = 2n’ times. This algorithm is O(n).

   c. In this fragment, the outer loop runs ‘n’ times, and for each iteration of the outer loop, the inner loop runs from ‘n’ to ‘i’, inclusive, so it runs ‘n - 1 + 1’ times. The total of executions for all values of ‘i’ from 0 to ‘n-1’. So, the total is: ‘n + (n-1) + (n-2) + … + 2 = n(n + 1)/2. This algorithm is)(n^2).

   d. In this fragment, the outer loop runs ‘n-1’ times (from 1 to ‘n-1), and for each iteration of the outer loop, the inner loop runs ‘ i - 1’ times, the if condition inside the inner loop is true only when ‘j’ is O(since ‘j % i == 0’), which happens once for each value of ‘i’ . So, the output statement is executed once for each value of ‘i’. Therefore, the total number of executions of the output statement is ‘n - 1’. This algorithm is O(n).

2. a. This loop starts at index3 (i = 3) and iterates up to the second-to-last element of ‘anArray’ (since ‘anArray.length - 1’ is used as the loop condition). in each iteration, it shifts the value at ‘anArray[i]’ to the right, effectively overwriting the value at ‘anArray[i + 1} ‘ . After the loop execution, the contents of ‘anArray’ will be:

   {0,1,2,3,4,5,6}

   b. This loop starts at the last index of ‘ anArray’ and iterates down to index 4,effectively shifting each element one position to the right. The value at ‘anArray [i -1] ‘ is copied to ‘anArray[i]’ . After the loop execution, the contents of ‘ anArray’ will be:

   {0,1,2,3,4,5,6}

3. a. Sum of an Array
   • Time Complexity (O(n)): In the given algorithm, there's a loop that iterates through the array once. Since it iterates through each element of the array

once, the time complexity is linear with respect to the size of the array, denoted as n. Thus, the time complexity is O(n).
• Total Operations (T(n)): Within the loop, there are a few constant-time operations performed n times. Specifically, there's one addition operation and one array access operation inside the loop. Additionally, there's one operation outside the loop (returning the sum). So, the total operations T(n) is proportional to n.

b. Matrix Multiplication
• Time Complexity (O(n3)): In the matrix multiplication algorithm, there are three nested loops. Each of these loops iterates over the dimensions of the matrices. So, the time complexity is cubic with respect to the dimensions of the matrices, denoted as r1, c1, and c2. Thus, the time complexity is 0(r1.c1 · c2) or O(n3), where n represents the size of the matrices.
• Total Operations (T(n)): Within the nested loops, there are constant-time operations (multiplications and additions) performed3 times, where n represents the dimensions of the
matrices.

c. For Looping
Time Complexity (O(n)): In this code snippet, there are three separate loops, each of which iterates n times. Therefore, the time complexity is linear with respect to n, making it O(n). Total Operations (T(n)): Each loop consists of constant-time operations executed n times, resulting in a total of 3n operations.

d. While Looping
Time Complexity (O(log n)): In this algorithm, the loop decrements & by half each iteration until
it reaches zero. This behavior is characteristic of a logarithmic time complexity. Therefore, the time complexity is O(log n).
• Total Operations (T(n)): Within the loop, there are constant-time operations performed for each iteration. The total number of operations depends on the number of iterations, which is logarithmic in n.

4. Please provide the examples of some well-known algorithms and their time complexity in Big O notation. Find at least one for each degree complexity.

. Examples of Well-known Algorithms and Their Time Complexity:
• Constant Time (0(1)): Accessing an element in an array by index.
* Linear Time (O(n)): Linear search in an unsorted array.
* Logarithmic Time (O(log n)): Binary search in a sorted array.

• Linearithmic Time (O(n log n)): Merge sort.
• Quadratic Time (O(n^2)): Bubble sort.
* Cubic Time (O(n^3)): Matrix multiplication using the naive method.
* Exponential Time (O(2^n)): Brute-force solution for the subset sum problem.

5. Abstract Data Type (ADT) in Data Structure:
   An Abstract Data Type (ADT) is a mathematical model for data types. It specifies a set of operations and the semantics of these operations, without specifying their implementation.
   Examples include stacks, queues, lists, sets, and maps. Here's a simple example of implementing a stack ADT in Java:

```
import java.util.*;

public class StackADT<T> {
    private Deque<T> stack;

    public StackADT() {
        stack = new ArrayDeque<>();
    }

    public void push(T item) {
        stack.push(item);
    }

    public T pop() {
        return stack.pop();
    }

    public T peek() {
        return stack.peek();
    }

    public boolean isEmpty() {
        return stack.isEmpty();
    }
}
```

6. Please provide a summary of the difference between List and ArrayList in Java. Show it in the form of a table.

Feature	List	ArrayList
Implementation	Interface	Class
Resizable	Yes	Yes
Performance	Slower than ArrayList	Faster than List
Memory Usage	More memory overhead	Less memory overhead
Efficiency	Iterations are slower	Iterations are faster
Growth Strategy	Doubles in size when full	Increases by 50% when full

7. Write a program that does the following

• Create an ArrayList of integers, add the elements [12, 25, 34, 46] to it
• Remove the number 25 from the ArrayList
• Print the final ArrayList

```
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        // Create ArrayList of integers
        ArrayList<Integer> numbers = new ArrayList<>();

        // Add elements to ArrayList
        numbers.add(12);
        numbers.add(25);
        numbers.add(34);
        numbers.add(46);

        // Remove number 25
        numbers.remove(Integer.valueOf(25));
```

```
        // Print final ArrayList
        System.out.println("Final ArrayList: " + numbers);
    }
}
```