# 24TH JULY COMPLEXITY ANALYSIS AND ARRAY

**Q1: Program to print the sum of all the elements present at even indices in the given array:**

java

Copy code

```
public class SumOfEvenIndicesElements {
    public static void main(String[] args) {
        int[] arr = {3, 20, 4, 6, 9};
        int sum = 0;

        for (int i = 0; i < arr.length; i += 2) {
            sum += arr[i];
        }

        System.out.println("Output: " + sum);
    }
}
```

**Q2: Program to traverse the array using a for-each loop and print all even elements:**

java

Copy code

```
public class PrintEvenElements {
    public static void main(String[] args) {
        int[] arr = {34, 21, 54, 65, 43};

        System.out.print("Output: ");
        for (int num : arr) {
            if (num % 2 == 0) {
                System.out.print(num + " ");
            }
        }
```

```
    }
}
```

**Q3: Program to calculate the maximum element in the array:**

java

Copy code

```
public class MaxElement {
    public static void main(String[] args) {
        int[] arr = {34, 21, 54, 65, 43};
        int max = arr[0];

        for (int i = 1; i < arr.length; i++) {
            if (arr[i] > max) {
                max = arr[i];
            }
        }

        System.out.println("Output: " + max);
    }
}
```

**Q4: Program to find the second largest element in the given array:**

java

Copy code

```
public class SecondLargestElement {
    public static void main(String[] args) {
        int[] arr = {34, 21, 54, 65, 43};
        int largest = arr[0];
        int secondLargest = Integer.MIN_VALUE;

        for (int i = 1; i < arr.length; i++) {
            if (arr[i] > largest) {
                secondLargest = largest;
                largest = arr[i];
            } else if (arr[i] > secondLargest && arr[i] != largest) {
```

```
                secondLargest = arr[i];
            }
        }

        System.out.println("Output: " + secondLargest);
    }
}
```

**Q5: Program to find the first peak element in the array:**

```
java
Copy code
public class FirstPeakElement {
    public static void main(String[] args) {
        int[] arr = {1, 3, 2, 6, 5};

        for (int i = 1; i < arr.length - 1; i++) {
            if (arr[i] > arr[i - 1] && arr[i] > arr[i + 1]) {
                System.out.println("Output: " + arr[i]);
                break;
            }
        }
    }
}
```

**Q6: Java code to count positive, negative, odd, even, and zero numbers from user inputs:**

```
```java
import java.util.Scanner;

public class NumberCounter {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of rows (m): ");
        int m = scanner.nextInt();
```
```

```
System.out.print("Enter the number of columns (n): ");
int n = scanner.nextInt();

int positiveCount = 0;
int negativeCount = 0;
int oddCount = 0;
int evenCount = 0;
int zeroCount = 0;

System.out.println("Enter " + (m * n) + " integer inputs:");

for (int i = 0; i < m * n; i++) {
    int num = scanner.nextInt();
    if (num > 0) {
        positiveCount++;
    } else if (num < 0) {
        negativeCount++;
    }
    if (num % 2 == 0) {
        evenCount++;
    } else {
        oddCount++;
    }
    if (num == 0) {
        zeroCount++;
    }
}

System.out.println("Number of positive numbers = " + positiveCount);
System.out.println("Number of negative numbers = " + negativeCount);
System.out.println("Number of odd numbers = " + oddCount);
System.out.println("Number of even numbers = " + evenCount);
System.out.println("Number of 0 = " + zeroCount);
```

```
        scanner.close();
    }
}
```

**Q7: Java code to print elements above the secondary diagonal in a user-inputted square matrix:**

```java
import java.util.Scanner;

public class SecondaryDiagonalElements {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the size of the square matrix: ");
        int size = scanner.nextInt();

        int[][] matrix = new int[size][size];

        System.out.println("Enter the elements of the square matrix:");

        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                matrix[i][j] = scanner.nextInt();
            }
        }

        System.out.println("Elements above the secondary diagonal:");
        for (int i = 0; i < size; i++) {
            for (int j = i + 1; j < size; j++) {
                System.out.print(matrix[i][j] + " ");
            }
        }

        scanner.close();
```

```
    }
}
```

**Q8: Java code to print elements of both the diagonals in a user-inputted square matrix:**

```java
import java.util.Scanner;

public class DiagonalElements {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the size of the square matrix: ");
        int size = scanner.nextInt();

        int[][] matrix = new int[size][size];

        System.out.println("Enter the elements of the square matrix:");

        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                matrix[i][j] = scanner.nextInt();
            }
        }

        System.out.println("Elements of the secondary diagonal:");
        for (int i = 0; i < size; i++) {
            System.out.print(matrix[i][size - i - 1] + " ");
        }

        System.out.println("\nElements of the primary diagonal:");
        for (int i = 0; i < size; i++) {
            System.out.print(matrix[i][i] + " ");
        }
```

```
        scanner.close();
    }
}
```

**Q9: Java code to find the largest element of a given 2D array of integers:**

```java
import java.util.Scanner;

public class LargestElement {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of rows: ");
        int rows = scanner.nextInt();

        System.out.print("Enter the number of columns: ");
        int columns = scanner.nextInt();

        int[][] arr = new int[rows][columns];

        System.out.println("Enter the elements of the array:");

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                arr[i][j] = scanner.nextInt();
            }
        }

        int largest = arr[0][0];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
```

```java
                if (arr[i][j] > largest) {
                    largest = arr[i][j];
                }
            }
        }

        System.out.println("Largest element in the array: " + largest);

        scanner.close();
    }
}
```

**Q10: Java function to display the elements of the middle row and middle column of a square matrix:**

```java
import java.util.Scanner;

public class MiddleRowAndColumn {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the size of the square matrix: ");
        int size = scanner.nextInt();

        int[][] matrix = new int[size][size];

        System.out.println("Enter the elements of the square matrix:");

        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                matrix[i][j] = scanner.nextInt();
            }
        }
```

```
        printMiddleRowAndColumn(matrix);

        scanner.close();
    }

    public static void printMiddleRowAndColumn(int[][] matrix) {
        int size = matrix.length;
        int middleRow = size / 2;
        int middleColumn = size / 2;

        System.out.println("Elements of the middle row:");
        for (int j = 0; j < size; j++) {
            System.out.print(matrix[middleRow][j] + " ");
        }

        System.out.println("\nElements of the middle column:");
        for (int i = 0; i < size; i++) {
            System.out.print(matrix[i][middleColumn] + " ");
        }
    }
}
```

**11. Analyze the time complexity of the following Java code and suggest a way to improve it: int sum = 0; for (int i = 1; i < n; i+) { for(int j = 1; j < i; jH) { sUm+; } }**

**Ans:** The time complexity of the given Java code is O(n^2). It consists of two nested loops, one iterating from 1 to n and the other from 1 to i. As a result, the total number of iterations will be the sum of the first n natural numbers, which is O(n^2).

To improve the time complexity, we can optimize the code by using a mathematical formula to directly calculate the sum of the first n natural numbers, which is n * (n + 1) / 2. This way, we can achieve a linear time complexity of O(n) for calculating the sum, rather than O(n^2) as in the original code.

Improved Java code:

java

Copy code

int n = ... // some value of n

```
int sum = n * (n + 1) / 2;
```

**12: Find the value of $T(2)$ for the recurrence relation $T(n) = 3T(n-1) + 12n$, given that $T(0) = 5$.**

**Ans:** To find the value of $T(2)$ for the given recurrence relation $T(n) = 3T(n-1) + 12n$ with $T(0) = 5$, we can recursively apply the relation:

$T(2) = 3T(1) + 12(2)$

$T(1) = 3T(0) + 12(1) = 3(5) + 12 = 27$

$T(2) = 3(27) + 12(2) = 81 + 24 = 105$

So, the value of $T(2)$ is 105.

**13: Given a recurrence relation, solve it using a substitution method. Relation: $T(n) = T(n - 1) + c$**

**Ans:** To solve the given recurrence relation $T(n) = T(n - 1) + c$ using the substitution method, we can repeatedly substitute the expression for $T(n - 1)$ into $T(n)$, until we reach the base case $T(0) = 5$.

$T(n) = T(n - 1) + c$

$= (T(n - 2) + c) + c$

$= T(n - 2) + 2c$

$= (T(n - 3) + c) + 2c$

$= T(n - 3) + 3c$

...

$= T(0) + nc$

$= 5 + nc$

Therefore, the solution to the recurrence relation is $T(n) = 5 + nc$.

**14: Given a recurrence relation: $T(n) = 16T(n/4) + n2logn$ Find the time complexity of this relation using the master theorem.**

**Ans:** To find the time complexity of the given recurrence relation using the master theorem, we need to first identify the values of a, b, and f(n).

$T(n) = 16T(n/4) + n^2 * \log(n)$

Here, $a = 16$ (the number of recursive calls),

$b = 4$ (the factor by which n is divided in each recursion), and

$f(n) = n^2 * \log(n)$.

Now, let's calculate the value of $\log_b(a)$:

$\log_4(16) = 2$, because $4^2 = 16$.

Comparing f(n) with n^log_b(a), we have:

n^2 * log(n) = n^(2 * log_4(16)) = n^2.

Since f(n) = n^2 and n^log_b(a) = n^2, we are in case 2 of the master theorem.

The time complexity of the given recurrence relation is O(n^2 * log(n)).

15: **Solve the following recurrence relation using recursion tree method T(n) = 27(n/2) + n**

**Ans:** To solve the recurrence relation T(n) = 27(n/2) + n using the recursion tree method, we'll construct a recursion tree and sum up the costs at each level.

Recursion tree:

scss

Copy code

```
          T(n)
           |
      T(n/2) + n
       /        \
 T(n/4) + n/2   T(n/4) + n/2
     ...          ...
     /              \
  T(1) + 1         T(1) + 1
```

The tree has a depth of log(n) (base 2), and at each level, the cost is n. Therefore, the total cost is the sum of costs at each level, which is:

Total cost = n + n/2 + n/4 + ... + 1

This is a geometric series with a common ratio of 1/2, and the sum of a geometric series is given by the formula:

Sum = (first term) * (1 - (common ratio)^(number of terms)) / (1 - common ratio)

Plugging in the values, we get:

Total cost = n * (1 - (1/2)^log(n)) / (1 - 1/2)

= n * (1 - 1/n) / (1/2)

= 2n * (1 - 1/n)

= 2n - 2

Therefore, the time complexity of the recurrence relation T(n) = 2T(n/2) + n using the recursion tree method is O(n).

16. **T(n) = 2T(n/2) + K, Solve using Recurrence tree method.**

**Ans:** To solve the recurrence relation T(n) = 2T(n/2) + K using the recursion tree method, we'll construct a recursion tree and sum up the costs at each level.

Recursion tree:

scss

Copy code

```
          T(n)
           |
      T(n/2) + K
       /       \
 T(n/4) + K    T(n/4) + K
     ...          ...
     /              \
  T(1) + K       T(1) + K
```

The recursion tree has a depth of log(n) (base 2), and at each level, the cost is K. Therefore, the total cost is the sum of costs at each level, which is:

Total cost = K + K + K + ... (log(n) times)

This can be expressed as:

Total cost = K * log(n)

Therefore, the time complexity of the recurrence relation T(n) = 2T(n/2) + K using the recursion tree method is O(log(n)).