**Time and Space Complexity Assignment**

Question 1. 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; j++) {

sum++;

}

}

Solution: The time complexity of this code is O(n^2) as it uses nested loops, where the outer loop runs n times and the inner

loop runs i times where i varies from 1 to n.

The total number of operations performed can be calculated by summing the total number of operations in each

iteration of the outer loop. The inner loop will run i times on the i-th iteration of the outer loop. So the number of

operations is (1+2+3+...+n) which is n(n+1)/2, which is O(n^2).

One way to improve the time complexity of this code is to use a mathematical formula to find the sum instead of

using nested loops.

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

Solution: given T(n) = 3T(n-1) + 12n

Substituting the values in the relation:

T(1) = 3T(0) + 12

=> T(1) = 15 + 12 = 27

T(2) = 3T(1) + 12 \* 2

=>T(2) = 3 \* 27 + 24 = 81 + 24

Hence T(2) = 105.

Question 3: Given a recurrence relation, solve it using a substitution method. Relation : T(n) = T(n - 1) + c.

Solution: Let the solution be T(n) = O(n), now let’s prove this using the induction method.

For that to happen T(n) <= cn where c is some constant.

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

T(n - 1) = T(n - 2) + c

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

|

|

T(2) = T(1) + c

—------------------------ Adding all the above equations

T(n) = T(1) + cn

Let us assume T(1) to be a constant value.

T(n) = k + cn

Therefore, T(n) <= cn

Hence we can conclude T(n) = O(n).

Question 4: Given a recurrence relation:

T(n) = 16T(n/4) + n2logn

Find the time complexity of this relation using the master theorem.

Solution: From the given recurrence we can obtain the value of different parameters such as a,b,p and k

The relation:T(n)=16T(n/4)+n2logn

Here,a=16

b = 4

k = 2

p = 1

bk = 42 = 16

Here a=bk

Also p > -1

Hence T(n) = θ(nlogab\*logp+1n)

Therefore T(n)= θ(nlog164\*log1+1n) = θ(n1/2log2n)

Question 5: Solve the following recurrence relation using recursion tree method

T(n) = 2T(n/2) + n

Solution: To solve the given recurrence relation T(n) = 2T(n/2) + n using the recursion tree method, we will recursively break down the relation until we reach the base case.

Step 1: Draw the recursion tree: At each level, we have two subproblems, each with size n/2.

T(n)

/ \

T(n/2) T(n/2)

/ \ / \

T(n/4) T(n/4)

We can see that the depth of the recursion tree is log(n) since we divide the problem size by 2 at each level.

Step 2: Analyze the work done at each level: At each level i (starting from 0), the work done is n \* (2^i).

Level 0: n Level 1: n/2 + n/2 = n Level 2: (n/4 + n/4) + (n/4 + n/4) = n ... Level i: n \* (2^i)

Step 3: Calculate the number of levels in the recursion tree: The recursion tree stops when the problem size becomes 1. Therefore, the number of levels is log(n).

Step 4: Sum up the work done at each level: We sum up the work done at each level to find the total work.

Total work = n + n + n + ... (log(n) times) = log(n) \* n

Step 5: Analyze the additional work outside the recursion tree: In each level, we perform additional work of n units (as given by the recurrence relation).

Total additional work = n \* log(n)

Step 6: Calculate the total work: The total work is the sum of the work done inside the recursion tree and the additional work outside the tree.

Total work = log(n) \* n + n \* log(n) = 2 \* n \* log(n)

Therefore, the solution to the recurrence relation T(n) = 2T(n/2) + n using the recursion tree method is T(n) = 2 \* n \* log(n).

Question 6. T(n) = 2T(n/2) + K, Solve using Recurrence tree method.

Solution: To solve the recurrence relation T(n) = 2T(n/2) + K using the recursion tree method, we will recursively break down the relation until we reach the base case.

Step 1: Draw the recursion tree: At each level, we have two subproblems, each with size n/2.

T(n)

/ \

T(n/2) T(n/2)

/ \ / \

T(n/4) T(n/4)

We can see that the depth of the recursion tree is log(n) since we divide the problem size by 2 at each level.

Step 2: Analyze the work done at each level: At each level i (starting from 0), the work done is K \* (2^i).

Level 0: K Level 1: K + K = 2K Level 2: 2K + 2K = 4K ... Level i: K \* (2^i)

Step 3: Calculate the number of levels in the recursion tree: The recursion tree stops when the problem size becomes 1. Therefore, the number of levels is log(n).

Step 4: Sum up the work done at each level: We sum up the work done at each level to find the total work.

Total work = K + 2K + 4K + ... (log(n) times) = K \* (1 + 2 + 4 + ... + 2^(log(n))) = K \* (2^(log(n)+1) - 1) = K \* (2n - 1)

Step 5: Analyze the additional work outside the recursion tree: In each level, we perform additional work of K units (as given by the recurrence relation).

Total additional work = K \* log(n)

Step 6: Calculate the total work: The total work is the sum of the work done inside the recursion tree and the additional work outside the tree.

Total work = K \* (2n - 1) + K \* log(n) = K \* (2n - 1 + log(n))

Therefore, the solution to the recurrence relation T(n) = 2T(n/2) + K using the recursion tree method is T(n) = K \* (2n - 1 + log(n)).

**2D Array Assignment**

Q1: Take m and n input from the user and m \* n integer inputs from user and print the following:

number of positive numbers

number of negative numbers

number of odd numbers

number of even numbers

number of 0.

Solution: import java.io.\*;

import java.util.\*;

public class Main{

public static void main(String args[]){

int m,n;

Scanner sc=new Scanner(System.in);

System.out.print("enter the number of rows=");

m=sc.nextInt();

System.out.print("enter the number of column=");

n=sc.nextInt();

int arr[][]=new int[m][n];

int i,j;

System.out.println("enter the matrix element=\n");

for(i=0;i<m;i++){

for(j=0;j<n;j++){

arr[i][j]=sc.nextInt();

}

}

int positive = 0 , negative = 0 , zero = 0 , odd = 0 , even = 0;

for( i = 0 ; i < m ; i++){

for( j = 0 ; j < n ; j++){

if(arr[i][j] > 0)positive++;

if(arr[i][j] < 0)negative++;

if(arr[i][j] == 0)zero++;

if((arr[i][j] % 2) == 0)even++;

if((arr[i][j] % 2) != 0)odd++;

}

}

System.out.println("Number of positives = " + positive);

System.out.println("Number of negatives = " + negative);

System.out.println("Number of odds = " + odd);

System.out.println("Number of evens = " + even);

System.out.println("Number of zeroes = " + zero);

}

}

Q2: write a program to print the elements above the secondary diagonal in a user inputted square matrix.

Solution: import java.io.\*;

import java.util.\*;

public class Main{

public static void main(String args[]){

int m,n;

Scanner sc=new Scanner(System.in);

System.out.print("enter the number of rows : ");

m=sc.nextInt();

System.out.print("enter the number of column : ");

n=sc.nextInt();

int arr[][]=new int[m][n];

int i,j;

System.out.println("enter the matrix element : ");

for(i = 0 ; i < m ; i++){

for(j = 0 ; j < n ; j++){

arr[i][j]=sc.nextInt();

}

}

System.out.println("Elements above secondary diagonal are as follows : ");

for(i = 0 ; i < m ; i++){

for(j = 0 ; j < n ; j++){

if(i + j < m - 1)System.out.print(arr[i][j] + " ");

}

}

}

}

Q3: write a program to print the elements of both the diagonals in a user inputted square matrix in any order.

Solution:

import java.io.\*;

import java.util.\*;

public class Main{

public static void main(String args[]){

int m,n;

Scanner sc=new Scanner(System.in);

System.out.print("enter the number of rows : ");

m=sc.nextInt();

System.out.print("enter the number of column : ");

n=sc.nextInt();

int arr[][]=new int[m][n];

int i,j;

System.out.println("enter the matrix element : ");

for(i = 0 ; i < m ; i++){

for(j = 0 ; j < n ; j++){

arr[i][j]=sc.nextInt();

}

}

System.out.println("Elements of both the diagonals are as follows : ");

for(i = 0 ; i < m ; i++){

for(j = 0 ; j < n ; j++){

if(i + j == m - 1)System.out.print(arr[i][j] + " ");

else if(i == j)System.out.print(arr[i][j] + " ");

}

}

}

}

Q4: Write a program to find the largest element of a given 2D array of integers.

Solution: import java.io.\*;

import java.util.\*;

public class Main{

public static void main(String args[]){

int m,n;

Scanner sc=new Scanner(System.in);

System.out.print("enter the number of rows : ");

m=sc.nextInt();

System.out.print("enter the number of column : ");

n=sc.nextInt();

int arr[][]=new int[m][n];

int i,j;

System.out.println("enter the matrix element : ");

for(i = 0 ; i < m ; i++){

for(j = 0 ; j < n ; j++){

arr[i][j]=sc.nextInt();

}

}

int maximum = -100000008;

for(i = 0 ; i < m ; i++){

for(j = 0 ; j < n ; j++){

if(maximum < arr[i][j])maximum = arr[i][j];

}

}

System.out.println("The maximum element in this matrix is : " + maximum);

}

}

Q5: Write a function which accepts a 2D array of integers and its size as arguments and

displays the elements of middle row and the elements of middle column. Printing can

be done in any order.

[Assuming the 2D Array to be a square matrix with odd dimensions i.e. 3x3, 5x5, 7x7 etc...]

Solution: import java.io.\*;

import java.util.\*;

public class Main{

public static void main(String args[]){

int m,n;

Scanner sc=new Scanner(System.in);

System.out.print("enter the number of rows : ");

m=sc.nextInt();

int arr[][]=new int[m][m];

int i,j;

System.out.println("enter the matrix element : ");

for(i = 0 ; i < m ; i++){

for(j = 0 ; j < m ; j++){

arr[i][j]=sc.nextInt();

}

}

System.out.println("The elements of the middle row and middle column are as follows : ");

for(i = 0 ; i < m ; i++)System.out.print(arr[i][m/2] + " ");

for(j = 0 ; j < m ; j++){

if(j == m/2)continue;

System.out.print(arr[m/2][j] + " ");

}

}

}

**1D Array Assignment**

Q1: Write a program to print the sum of all the elements present on even indices in the given array.

Input 1: arr[] = {3,20,4,6,9}

Output 1: 16

Input 1: arr[] = {4,3,6,7,1}

Output 1: 11

Solution: public class CODE10 {

public static void main(String[] args) {

int[] arr = {1,2,3,4,5,6,7,8};

int i = 0, sum = 0;

while (i < arr.length) {

sum += arr[i];

i += 2;

}

System.out.println(sum);

}

}

Q2: Write a program to traverse over the elements of the array using for each loop and print all even

elements.

Input 1: arr[] = {34,21,54,65,43}

Output 1: 34 54

Input 1: arr[] = {4,3,6,7,1}

Output 1: 4 6

Solution: public class CODE11 {

public static void main(String[] args) {

int[] arr = { 1, 2, 3, 4, 5, 6, 7, 8 };

for (int elem : arr) {

if (elem % 2 == 0)

System.out.println(elem);

}

}

}

Q3: Write a program to calculate the maximum element in the array.

Input 1: arr[] = {34,21,54,65,43}

Output 1: 65

Input 1: arr[] = {4,3,6,7,1}

Output 1: 7

Solution: public class CODE12 {

public static void main(String[] args) {

int[] arr = { 10, 7, -5, 8, 9, 0, -4 };

int max = Integer.MIN\_VALUE;

for (int val : arr) {

max = Math.max(max, val);

}

System.out.print("Largest in given array is " + max);

}

}

Q4: Write a program to find out the second largest element in a given array.

Input 1: arr[] = {34,21,54,65,43}

Output 1: 54

Input 1: arr[] = {4,3,6,7,1}

Output 1: 6

Solution: public class CODE13 {

public static void main(String[] args) {

int[] arr = { 34, 21, 54, 65, 43 };

int arr\_size = arr.length;

int i, first, second;

// There should be at least two elements

if (arr\_size < 2) {

System.out.printf(" Invalid Input ");

return;

}

int largest = second = Integer.MIN\_VALUE;

// Find the largest element

for (i = 0; i < arr\_size; i++)

largest = Math.max(largest, arr[i]);

// Find the second largest element

for (i = 0; i < arr\_size; i++) {

if (arr[i] != largest)

second = Math.max(second, arr[i]);

}

if (second == Integer.MIN\_VALUE)

System.out.printf("There is no second " +

"largest element\n");

else

System.out.printf("%d\n", second);

}

}

Q5: Given an array. Find the first peak element in the array. A peak element is an element that is greater than

its just left and just right neighbor.

Input 1: arr[] = {1,3,2,6,5}

Output 1: 6

Input 2: arr[] = {1 4,7,3,2,6,5}

Output 1: 7

Solution: public class CODE14 {

public static void main(String[] args) {

int[] arr = { 1, 1, 3, 4, 2, 3, 5, 7, 1 };

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

if (arr[i - 1] < arr[i] && arr[i] > arr[i + 1]) {

System.out.print(arr[i]);

break;

}

}

}

}