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

Output 2: 11

Ans:

**package** Assignment\_1D\_Array;

**public** **class** Array\_Even\_Index\_sum\_Question1

{

**public** **static** **void** main(String[] args)

{

**int**[] arr = {3,20,4,6,9};

**int** n = arr.length;

**int** i = 0,sum = 0;

**while**(i<n)

{

sum +=arr[i];

//addition of even index element

i +=2;

}

System.***out***.println("The sum of even index elements is: "+sum);

}

}

// Output:The sum of even index elements is: 16

**Time Complexity is-: O (n)**

**Space Complexity is-: O (1)**

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

Ans:

**package** Assignment\_1D\_Array;

**public** **class** Array\_Even\_Element\_Print\_Question2

{

**public** **static** **void** main(String[] args)

{

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

**for**(**int** num : arr)

{

//check whether the number is even or odd

**if**(num % 2 == 0)

{

System.***out***.println(num);

}

}

}

}

//Output: 34

// 54

**Time Complexity is-: O (n)**

**Space Complexity is-: O (1)**

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

Ans:

**package** Assignment\_1D\_Array;

**public** **class** Array\_Maximum\_Element\_Print\_Question3

{

**public** **static** **void** main(String[] args)

{

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

**int** max = Integer.***MIN\_VALUE***;

**for**(**int** num : arr)

{

max = Math.*max*(max, num);

}

System.***out***.println("The maximum element in a given array is: "+max);

}

}

//Output:

// The maximum element in a given array is: 65

**Time Complexity is-: O (n)**

**Space Complexity is-: O (1)**

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

Ans:

**package** Assignment\_1D\_Array;

**public** **class** Array\_Second\_Largest\_Element\_Print\_Question4

{

**public** **static** **void** main(String[] args)

{

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

//max1=first largest element in an array

//max2=second largest element in an array

**int** max1 = Integer.***MIN\_VALUE***;

**int** max2 = Integer.***MIN\_VALUE***;

**for**(**int** num : arr)

{

max1 = Math.*max*(max1, num);

}

**for**(**int** num : arr)

//condition makes sure that we are getting the second largest element in an array

{

**if**(num != max1)

{

max2 = Math.*max*(max2, num);

}

}

**if**(max2 == Integer.***MIN\_VALUE***)

{

System.***out***.println("No second largest element found");

}

**else**

{

System.***out***.println("The second largest element in a given array is: "+max2);

}

}

}

//Output:

// The second largest element in a given array is: 54

**Time Complexity is-: O (n)**

**Space Complexity is-: O (1)**

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

Ans:

**package** Assignment\_1D\_Array;

**public** **class** Array\_Peak\_Element\_Question5

{

**public** **static** **void** main(String[] args)

{

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

**int** n = arr.length;

**for**(**int** i=1;i<n-1;i++)

{

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

{

System.***out***.println(arr[i]);

**break**;

}

}

}

}

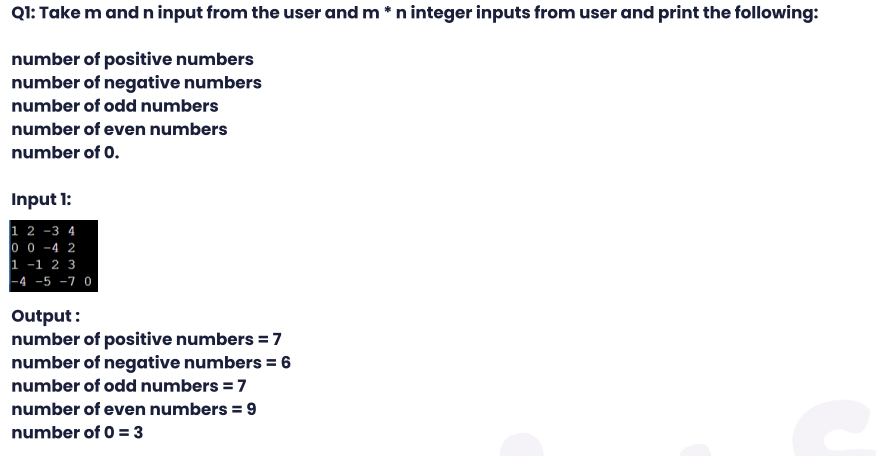
//Output: 7

**Time Complexity is-: O (n)**

**Space Complexity is-: O (1)**

**2D Array**

**Assignment**



Ans:

**package** Assignment\_2D\_Array;

**import** java.util.\*;

**import** java.sql.Array;

**public** **class** Arrays\_2D\_CountValues\_Question1

{

**public** **static** **void** main(String[] args)

{

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter the number of rows: ");

**int** m = sc.nextInt();

System.***out***.println("Enter the number of columns: ");

**int** n = sc.nextInt();

//2D array initialization

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

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

**for**(**int** i=0;i<m;i++)

{

**for**(**int** j=0;j<n;j++)

{

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

}

}

//logic of finding the total number of elements positive,negative,even,odd and zeros

**int** positive = 0, negative = 0, even = 0,odd = 0,zeros = 0;

//traverse array

**for**(**int** i=0;i<m;i++)

{

**for**(**int** j=0;j<n;j++)

{

//1. positive elements

**if**(arr[i][j] > 0 )

{

positive++;

}

//2.Negative elements

**if**(arr[i][j]<0)

{

negative++;

}

//3.even elements

**if**(arr[i][j]%2 == 0)

{

even++;

}

//4.Odd elements

**if**((arr[i][j]%2) != 0)

{

odd++;

}

//2.Zeros elements

**if**(arr[i][j] == 0)

{

zeros++;

}

}

}

System.***out***.println("The total number of positive elements= "+positive);

System.***out***.println("The total number of negative elements= "+negative);

System.***out***.println("The total number of even elements= "+even);

System.***out***.println("The total number of odd elements= "+odd);

System.***out***.println("The total number of zeros elements= "+zeros);

}

}

/\* Output:

Enter the number of rows:

4

Enter the number of columns:

4

Enter the array elements:

1 2 -3 4

0 0 -4 2

1 -1 2 3

-4 -5 -7 0

The total number of positive elements= 7

The total number of negative elements= 6

The total number of even elements= 9

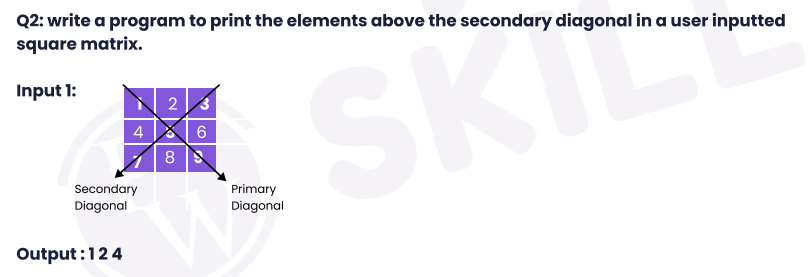
The total number of odd elements= 7

The total number of zeros elements= 3

\*/

**Time Complexity is-: O(m\*n)**

**Space Complexity is-:O(1)**



Ans:

**package** Assignment\_2D\_Array;

**public** **class** Arrays\_2D\_SecondaryDiagonal\_Question2

{

**public** **static** **void** main(String[] args)

{

**int**[][] arr = {{1,2,3},{4,5,6},{7,8,9}};

//m=Number of rows

//n=Number of coloumns

**int** m = arr.length;

**int** n = arr.length;

**for**(**int** i=0;i<m;i++)

{

**for**(**int** j=0;j<n;j++)

{

//condition to display elements above the secondary diagonal

**if**((i+j)<n-1)

{

System.***out***.print(arr[i][j]+" ");

}

}

}

System.***out***.println(" ");

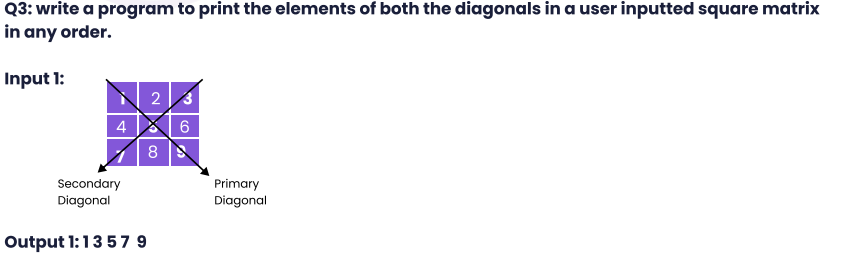
}

}

//Output:1 2 4

**Time Complexity is-: O (n2)**

**Space Complexity is-: O (1)**



Ans:

**package** Assignment\_2D\_Array;

//1. Brute Force Approach

//Time Complexity:O(**n2**)

//Space Complexity:O(1)

//2. Optimized Approach

//Time Complexity:O(**n**)

//Space Complexity:O(1)

**public** **class** Arrays\_2D\_Primary\_And\_Secondary\_Diagonal\_Element\_Print\_Question3

{

**public** **static** **void** main(String[] args)

{

**int**[][] arr = {{1,2,3},{4,5,6},{7,8,9}};

**int** m = arr.length;

/\*First Approach- Brute Force Approach

//approach to display the diagonal element

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

{

for(int j=0;j<arr[0].length;j++)

{

//check for the primary diagonal elements

if(i == j)

{

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

}

//check for the secondary diagonal elements

else if((i+j) == arr[0].length-1)

{

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

}

}

}

\*/

//Second Approach- Optimized Approach

**for**(**int** i=0;i<m;i++)

{

//primary element in the give 2D Matrix

System.***out***.print(arr[i][i]+" ");

//secondary element in the give 2D Matrix

//below check is to avoid the display of the middle element twice

**if**(i != m-1-i)

{

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

}

}

System.***out***.println(" ");

}

}

//First Approach Output : 1 3 5 7 9

//Second Approach Output: 1 3 5 9 7





Ans: **package** Assignment\_2D\_Array;

**public** **class** Arrays\_2D\_Largest\_Element\_Find\_Question4

{

**public** **static** **void** main(String[] args)

{

**int**[][] arr = {{1,2,4,0},{2,5,7,-1},{4,2,6,9}};

**int** m = arr.length;

**int** n = arr[0].length;

//logic to find the maximum element in a 2D array

**int** max = Integer.***MIN\_VALUE***;

**for**(**int** i=0;i<m;i++)

{

**for**(**int** j=0;j<n;j++)

{

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

}

}

System.***out***.println("The largest element in a given 2D array is="+max);

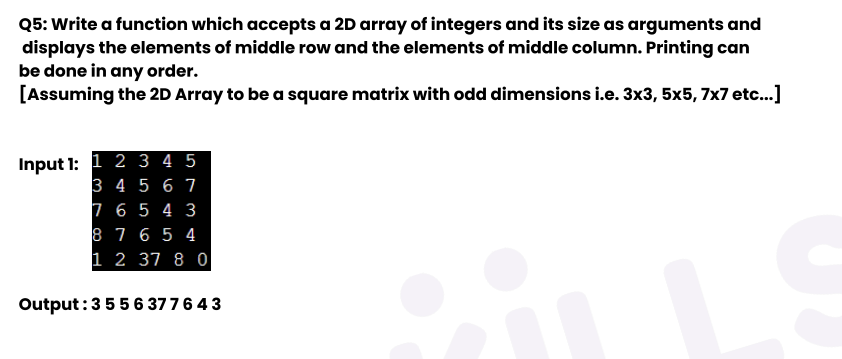
}

}

//Output:The largest element in a given 2D array is=9

**Time Complexity is-: O(m\*n)**

**Space Complexity is-:O(1)**

Ans:

**package** Assignment\_2D\_Array;

//Time Complexity is-: O(m) if n < m or O(n) if m < n

//Space Complexity is-:O(1)

**public** **class** Arrays\_2D\_Display\_Middle\_Element\_Question5

{

**public** **static** **void** main(String[] args)

{

**int**[][] arr = {{1,2,3,4,5},{3,4,5,6,7},{7,6,5,4,3},{8,7,6,5,4},{1,2,37,8,0}};

**int** m = arr.length;

**int** n = arr[0].length;

//display of middle column values

**for**(**int** i=0;i<m;i++)

{

System.***out***.print(arr[i][n/2]+" ");

}

//display of middle row values

**for**(**int** j=0;j<n;j++)

{

**if**(j == n/2)

{

**continue**;

}

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

}

System.***out***.println(" ");

}

}

//Output: 3 5 5 6 37 7 6 4 3

**Time and Space Complexity**

**Assignment**

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++;

}

}

Ans: Time complexity O(n2)

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.

Sum=n(n+1)/2

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

Ans Given, T(0) = 5

T(n) = 3T(n-1) + 12n

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

=3\*5+12

=15+12

=27

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

=3\*27+24

=81+24

=105

1. Given a recurrence relation, solve it using a substitution method.

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

Ans:

T(n) = T(n - 1) + c …………Standard Equation(1)

T(n-1) = T(n-2) + c …………..(put this value in eq.1)

So,here

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

T(n)= T(n-2) +2c ……………………….Equation(2)

T(n-2)= T(n-2) +c …………..(put this value in eq.1)

So,here

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

T(n)= T(n-3) +3c ……………………….Equation(3)

K times

T(n) = T(n-k) + kc

n- k =1

n=k

T(n) = T(1) + kc

T(1)=Constant

So,here

T(n) = kc

Time Complexity is T (n) =O (n)

1. Given a recurrence relation:

T(n) = 16T(n/4) + n2 log n

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

Ans: Master’s Theorem Standard Equation:-

a=16 b=4 k=2 p=1

a≥1

b>1

These are all satisfied

k≥0

p is a real number

a>bk

16>42

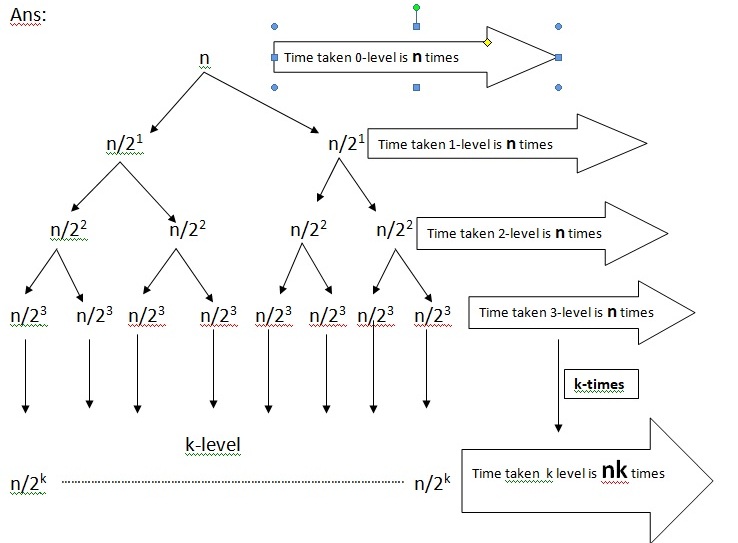
16>16

1St case not satisfied

2nd case- 1st part satisfied=p>-1

1>-1

1. Solve the following recurrence relation using recursion tree method T(n) = 2T(n/2) + n



So here assume,

n/2k =1

n=2k

Taking log both sides, we get

**log**2n= **klog**22 [∴**log**22=1]

k= **log**2n[Height of the tree]

Time Complexity

T(n)=nk

**T(n)=n log2n**

**T(n)=O (n log2n)**

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

Ans: T(n) = 2T(n/2) + K

Cost 0-leve is K

T(n)

Cost 1-leve is 2K

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

Cost 2-leve is 4K

T(n/22) T(n/22) T(n/22) T(n/22)

………………………………………………………………………………………..

………………………………………………………………………………………..

………………………………………………………………………………………..

…………………………………y-level……………………………………………

Cost level and so on

T(n/2y)

So, here

n/2y =1

n=2y

Taking Log both sides, we get

log2n=y log22 [∴log22=1]

y= log2n [Height of the tree]

These follows Sum of G.P. series

…………Eq(1)

Sum of G.P. Series=

a=1 r = =

Sum of G.P. Series=

=2 ……………………….These value put in 1st eq.

So, n(2)

2 is constant

Time Complexity:-O(n)