----------ASSIGNMENTS 1----------

Q1 ans-

- O(logn).
- Instead of using Nested loop, we can into a single loop and update a given array by using first loops which makes time complexity is O(n).

Q2 ans-

Given: T(n) = 3T(n-1) + 12n T(0) = 5

Let's calculate T(1) using the recurrence relation:

T(1) = 3T(1-1) + 12(1) T(1) = 3T(0) + 12 T(1) = 3(5) + 12 T(1) = 15 + 12 T(1) = 27

Now, let's calculate T(2) using the recurrence relation:

T(2) = 3T(2-1) + 12(2) T(2) = 3T(1) + 24 T(2) = 3(27) + 24 T(2) = 81 + 24 T(2) = 105

Therefore, the value of T(2) for the given recurrence relation is 105.

Q3 ans-

we'll assume a closed form solution of the form T(n) = a*n + b, where a and b are constants to be determined.

Let's substitute this assumed solution into the recurrence relation:

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

Substituting T(n) = a*n + b and T(n - 1) =* a(n - 1) + b:

a*n + b =* a(n - 1) + b + c

Simplifying the equation:

a*n + b =* an - a + b + c

a*n and b cancel out, resulting in:

0 = -a + c

Solving for a:

a = c

Now that we have found the value of a, we can substitute it back into the assumed solution to find b:

0 = -c + c

Since a = c, this equation simplifies to:

0 = 0

This equation doesn't provide any information about b. Therefore, we can choose any value for b.

Let's assume b = k, where k is a constant.

Finally, the solution to the recurrence relation T(n) = T(n - 1) + c is:

T(n) = c*n + k

where c is a constant, and k is an arbitrary constant.

Q4 ans-

To determine the time complexity of the recurrence relation T(n) = 16T(n/4) + n^2log(n) using the master theorem, we need to compare the equation to the standard form of the master theorem:

T(n) = aT(n/b) + f(n)

In this case, a = 16, b = 4, and f(n) = n^2log(n).

Now, we can compare the values of f(n) to n^log_b(a) and analyze the different cases:

1. If f(n) is polynomially smaller than n^log_b(a), i.e., f(n) = O(n^c) for some constant c < log_b(a), then the time complexity is dominated by the recursive term T(n/b).
2. If f(n) and n^log_b(a) have the same growth rate, i.e., f(n) = Θ(n^c log^k(n)), where c = log_b(a), then the time complexity is affected by both the recursive term T(n/b) and the additional term f(n).
3. If f(n) is polynomially larger than n^log_b(a), i.e., f(n) = Ω(n^c), where c > log_b(a), then the time complexity is dominated by the additional term f(n).

In our case, a = 16, b = 4, and f(n) = n^2log(n). Let's calculate n^log_b(a):

n^log_4(16) = n^2

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

n^2log(n) vs. n^2

f(n) is in case 2 since they have the same growth rate.

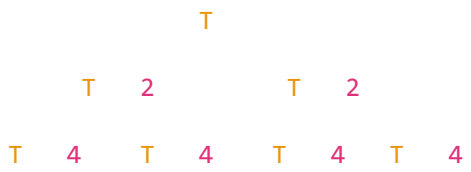
Therefore, using the master theorem, the time complexity of the recurrence relation T(n) = 16T(n/4) + n^2log(n) is Θ(n^2log(n)).

Q5 ans-

At each level of the tree, we have the recurrence relation T(n) = 2T(n/2) + K, and we keep dividing n by 2 until we reach the base case T(1) or T(0).

Let's draw the recursion tree:

```
scssCopy code
```



We can observe that the tree has a total of log n levels (base 2) since we keep dividing n by 2 at each level until we reach 1.

Now, let's calculate the cost at each level:

- At the root level, the cost is K.
- At level 1, there are two recursive calls of size n/2 each, resulting in a cost of K for each subtree, totaling 2K.
- At level 2, there are four recursive calls of size n/4 each, resulting in a cost of K for each subtree, totaling 4K.
- At level 3, there are eight recursive calls of size n/8 each, resulting in a cost of K for each subtree, totaling 8K.
- And so on...

We can see that at each level, the total cost is K. Since we have log n levels, the total cost of the recursion tree is:

Total cost = K * log n = K log n

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

----------ASSIGNMENT 2----------

Q1 ans -

```
public class Assignment_Question {
 public static void main(String[] args) {
 int[] arr = {3,20,4,6,9};
 int sum = 0;
 for(int i = 0; i< arr.length;i++)
 {
 if(i%2==0)
 {
 sum+=arr[i];
 }
 }
 System.out.println(sum);
 }
}
```

Q2 ans -

```
public class Assignment_Question {
 public static void main(String[] args) {
 int[] arr = {34, 21, 54, 65, 43};
 for (int i = 0; i < arr.length; i++) {
 if (arr[i] % 2 == 0) {
 System.out.print(arr[i] + " ");
 }
 }
 }
}
```

Q3 ans-

```
public class Assignment_Question
{
 public static void main(String[] args)
 {
 int[] arr = {34,21,54,65,43};
 int max = Integer.MIN_VALUE;
 for(int i = 0 ; i< arr.length;i++)
 {
 if(arr[i] > max)
 {
 max = arr[i];
 }
 }
 System.out.println(max);
 }
}
```

Q4 ans-

```
public class Assignment_Question
{
 public static void main(String[] args)
 {
 int[] arr = {34,21,54,65,43};
 Arrays.sort(arr);
 System.out.println(arr[arr.length-2]);
```

```
 }
}
```

Q5 ans -

```
public class Assignment_Question
{
 public static void main(String[] args)
 {
 int[] arr = {34,21,54,65,43};
 int max = Integer.MIN_VALUE;
 for(int i = 0 ; i< arr.length;i++)
 {
 if(arr[i] > max)
 {
 max = arr[i];
 }
 }
 System.out.println(max);
 }
}
```

----------2D Arrays Assignment----------

Q1 ans -

```
 import java.util.*;

public class check {
 public static void main(String[] args) {
 Scanner sc = new Scanner(System.in);
 System.out.print("Enter the number of rows : ");
 int m = sc.nextInt();
 System.out.print("Enter the number of Columns : ");
 int n = sc.nextInt();
```

```
int[][] arr = new int[m][n];
int number_of_positive_numbers = 0;
int number_of_negative_numbers = 0;
int number_of_odd_numbers = 0;
int number_of_even_numbers = 0;
int number_of_0 = 0;
System.out.println("Enter the elements ");
for(int i = 0; i<m;i++)
{
for(int j=0;j<n;j++)
{
arr[i][j] = sc.nextInt();
if(arr[i][j]>0)
{
number_of_positive_numbers++;
}
else if (arr[i][j] < 0)
{
number_of_negative_numbers++;
}
if (arr[i][j]%2== 1 || arr[i][j] %2 == -1 )
{
number_of_odd_numbers++;
}
if(arr[i][j] %2 == 0 || arr[i][j] == 0)
{
number_of_even_numbers++;
}
if (arr[i][j] == 0)
{
number_of_0++;
}

}
}

System.out.println("number_of_positive_numbers : "+ number_of_positive_numbers);
System.out.println("number_of_negative_numbers : "+number_of_negative_numbers);
System.out.println("number_of_odd_numbers : "+number_of_odd_numbers);
System.out.println("number_of_even_numbers : "+number_of_even_numbers);
System.out.println( "number_of_0 : " +number_of_0);
```

```
 }
}
```

Q2 ans-

```
public class Diagnoal {
 public static void main(String[] args) {
 int[][] arr = {
 {1,2,3},
 {4,5,6},
 {7,8,9}
 };
 int n = arr.length;
 for(int i = 0; i<n;i++)
 {
 for(int j =0; j< n-i-1;j++)
 {
 System.out.print(arr[i][j]+" ");
 }
 }
 }
}
```

Q3 ans -

```
 public class Diagnoal {
 public static void main(String[] args) {
 int[][] arr = {
 {1,2,3},
 {4,5,6},
 {7,8,9}
 };
 int n = arr.length;

 for(int i = 0; i<n;i++)
 {
 for(int j = 0; j<n;j++)
 {
 if(i==j)
 {
 System.out.print(arr[i][j]+" ");
 }
```

```
 else if(j == n-i-1)
 {
 System.out.print(arr[i][j]+" ");
 }
 }
 }
 }
}
```

Q4 ans-

```
public class Largest_2D {
 public static void main(String[] args) {
 int[][] arr1 =
 {
 {1,2,4,0},
 {2,5,7,-1},
 {4,2,6,9}
 };
 int n = arr1.length;
 int m = arr1[0].length;
 int max = Integer.MIN_VALUE;
 for(int i = 0; i<n; i++)
 {
 for(int j = 0;j<m;j++)
 {
 if(arr1[i][j] > max)
 {
 max = arr1[i][j];
 }
 }
 }
 System.out.println(max);
 }
}
```

Q5 ans-

```
public class Middle_2D {
 public static void main(String[] args) {
 int [][] arr2 = {
 {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 n = arr2.length;
 for(int i = 0; i<n;i++)
 {
 System.out.print(arr2[i][n/2]+" ");
 }

 for(int j = 0; j<n;j++)
 {
 if(j == n/2)continue;
 System.out.print(arr2[n/2][j]+" ");
 }
 }
}
```