TIME AND SPACE

```
1  class IfElseDemo
2  {
3     public static void main(String args[])
4     {
5        int i = 10;
6        if (i < 15)
7           System.out.println("i is smaller than 15");
8        else
9           System.out.println("i is greater than 15");
10    }
11 }
```

output

```
i is smaller than 15
```

```
1  class NestedIfDemo
2  {
3     public static void main(String args[])
4     {
5        int i = 10;
6        if (i == 10)
7        {
8           if (i < 15)
9              System.out.println("i is smaller than 15");
10          if (i < 12)
11             System.out.println("i is smaller than 12 too");
12          else
13             System.out.println("i is greater than 15");
14       }
15    }
16 }
```

output

```
i is smaller than 15
i is smaller than 12 too
```

```
1   class ifelseifDemo
2   {
3      public static void main(String args[])
4      {
5         int i = 20;
6
7         if (i == 10)
8            System.out.println("i is 10");
9         else if (i == 15)
10           System.out.println("i is 15");
11        else if (i == 20)
12           System.out.println("i is 20");
13        else
14           System.out.println("i is not present");
15     }
16  }
```

output

```
i is 20
```

```
1   class SwitchCaseDemo
2   {
3      public static void main(String args[])
4      {
5         int i = 9;
6         switch (i)
7         {
8         case 0:
9            System.out.println("i is zero.");
10           break;
11        case 1:
12           System.out.println("i is one.");
13           break;
14        case 2:
15           System.out.println("i is two.");
16           break;
17        default:
18           System.out.println("i is greater than 2.");
19        }
20     }
21  }
```

output

```
i is greater than 2.
```

tell the output

```
// Predict the output
class Test {
public static void main(String[] args)
    {
        int x = 10;
        if (x) {
            System.out.println("HELLO");
        } else {
            System.out.println("BYE");
        }
    }
}
```

```
// Predict the output
class Test {
public static void main(String[] args)
    {
        int x = 10;
        if (x)
            System.out.println("HELLO");
        System.out.println("WELCOME");

        else
        {
            System.out.println("BYE");
        }
    }
}
```

```
// Predict the output
class Test {
public static void main(String[] args)
    {
        if (true)
            ;
    }
}
```

```
// Predict the output
class MainClass {
public static void main(String[] args)
    {
        int x = 10;
        switch (x + 1 + 1) {
        case 10:
            System.out.println("HELLO");
            break;
        case 10 + 1 + 1:
            System.out.println(“BYE");
            break;
        }
    }
}
```

```
// Predict the output
public class A {
    public static void main(String[] args)
    {
        if (true)
            break;
    }
}
```

time complexity

What is Time Complexity?

You can get the time complexity by "counting" the number of operations performed by your code.

This time complexity is defined as a function of the input size n using Big-O notation. n indicates the input size, while O is the worst-case scenario growth rate function.

We use the Big-O notation to classify algorithms based on their running time or space (memory used) as the input grows.

The O function is the growth rate in function of the input size n.

Big O Cheatsheet

Big O Notation	Name	Examples
O(1)	Constant	Odd or Even number
O(log n)	Logarithmic	Finding element on sorted array with binary search
O(n)	Linear	Find max element in unsorted array
O(n log n)	Linearithmic	Sorting elements in array with merge sort
$O(n^2)$	Quadratic	Sorting array with bubble sort
$O(n^3)$	Cubic	3 variables equation solver
$O(2^n)$	Exponential	Find all subsets
O(n!)	Factorial	Find all permutations of a given set/string

O(1) – Constant Time Complexity

```
int addition(int x, int y)
{
        int a = x ;
        int b = y ;
        return a + b ;
}
```

O(n) – Linear Time Complexity

```
void print_numbers(int x)
{
        int num = 1 ;
        while(num <= x)
        {
                cout << num << " " ;
                num += 1 ;
        }
}
```

$O(n^2)$ – Quadratic Time Complexity

```
void square_pattern(int x)
{
      for(int i = 1 ; i <= x ; i++)
      {
            for(int j = 1 ; j <= x ; j++)
            {
                  cout << j << " " ;
            }
            cout << endl ;
      }
}
```

$O(n^3)$ – Cubic Time Complexity

```
void list_of_square_pattern(int arr[], int x)
{
      for(int k = 0 ; k < size ; k++)
      {
            for(int i = 1 ; i <= arr[k] ; i++)
            {
                  for(int j = 1 ; j <= arr[k] ; j++)
                  {
                        cout << j << " " ;
                  }
                  cout << endl ;
            }     cout << endl ;
}     }
```

$O(2^n)$ – Exponential Time Complexity

```
long int fibonacci(int num)
{
      if (num <= 1)
            return num ;
      return fibonacci(num - 2) + fibonacci(num - 1) ;
}
```

O(n!) – Factorial Time Complexity

```
void permute(string a, int l, int r)
{
        if (l == r)
                cout << a << endl ;
        else
        {
                for (int i = l; i <= r; i++)
                {
                swap(a[l], a[i]);

                permute(a, l+1, r);

                swap(a[l], a[i]);
                }
        }
}
```

O(log n) – Logarithmic Time Complexity

```
int sum_of_digits(int x)
{
        int sod = 0 ;
        while(x > 0)
        {
                sod = sod + x%10 ;
                x = x / 10 ;
        }
        return sod ;
}
```

O(n log n) – Linearithmic Time Complexity

```
void sum_of_digits(int arr[], int size)
{
      for(int i = 0 ; i < size ; i++)
      {
            int sod = 0 , x = arr[i] ;
            while(x > 0)
            {
                  sod = sod + x%10 ;
                  x = x / 10 ;
            }
            cout << sod << " " ;
}     }
```

SPACE COMPLEXITY

What is Space Complexity?

Space Complexity of an algorithm is the total space taken by the algorithm with respect to the input size.

Space complexity includes both Auxiliary space and space used by input.

Auxiliary Space is the extra space or temporary space used by an algorithm.

Space complexity is a parallel concept to time complexity. If we need to create an array of size n, this will require O(n) space. If we create a two-dimensional array of size n*n, this will require $O(n^2)$ space.

Predict the space complexity for the given below code snippet

```
int add (int n)
{
 if (n <= 0)
{
return 0;
 }
 return n + add (n-1);
 }
```

A) O(1) **B)** O(n) ✓ **C)** O (n * log n) **D)** O(n * n)

Predict the space complexity for the given below code snippet

```
int addSequence (int n)
{
      int sum = 0;
      for(int i = 0; i < n; i++){
            sum += pairSum(i, i+1);
      }
      return sum;
}
int pairSum(int x, int y){
      return x + y;
}
```

A) O(1) ✓ **B)** O(n) **C)** O (n * log n) **D)** O(n * n)

Predict the space complexity for the given below code snippet

```
int foo(int n) {
    int* arr = new int[n];
    for (int i = 0; i < n; i++) {
        arr[i] = i;
    }
    return arr[n-1];
}
```

A) O(1) **B)** O(n) ✓ **C)** O (n * log n) **D)** O(n * n)

Predict the space complexity for the given below code snippet

```
int sum = 0;
for (int i = 0; i < n; i++) {
    sum += i;
}
```

A) O(1) ✓ **B)** O(n) **C)** O (n * log n) **D)** O(n * n)

Predict the space complexity for the given below code snippet

```
int fibonacci(int n) {
    if (n <= 1) {
        return n;
    }
    return fibonacci(n-1) +
fibonacci(n-2);
}
```

A) O(1) **B)** O(n) ✓ **C)** O (n * log n) **D)** O(n * n)

Predict the space complexity for the given below code snippet

```
struct Node {
    int data;
    Node* next;
};

Node* head = nullptr;

void insert(int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = head;
    head = newNode;
}
```

A) O(1) **B)** O(n) ✓ **C)** O (n * log n) **D)** O(n * n)