Objective:

At the end of this lecture, students will understand :

- Time Complexity
- Space Complexity
- Importance of them in Competitive Programming

Time and Space Complexity:

Sometimes, there are more than one way to solve a problem. We need to learn how to compare the performance of different algorithms and choose the best one to solve a particular problem. While analysing an algorithm, we mostly consider time complexity and space complexity.

Time and Space Complexity: (Contd.)

- Time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input.
- Similarly, Space complexity of an algorithm quantifies the amount of space or memory taken by an algorithm to run as a function of the length of the input.
- Time and space complexity depends on lots of things like hardware, operating system, processors, etc.

Order of growth:

- Order of growth of an algorithm is a way of saying /predicting how execution time of a program and the space/memory occupied by it changes with the input size.
- The most famous way is the Big-Oh notation. It gives the worst case possibility for an algorithm.

Order of growth: (Contd.)

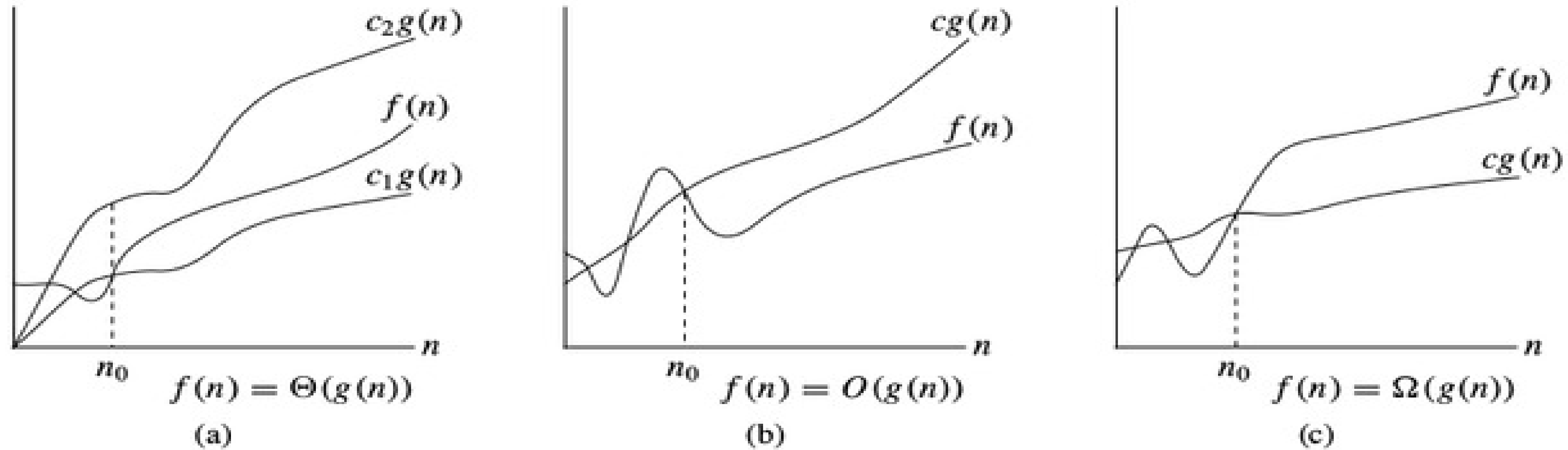
- E.g. a program takes $O(n)$ time means it takes at most n operations to compute the answer. Now if your input for n is 10^6 , it takes 10^6 operations, while a $O(n^2)$ algorithm takes 10^{12} operations for the same input.

Order of growth: (Contd.)

O-notation:

To denote asymptotic upper bound, we use O-notation. For a given function $g(n)$, we denote by $O(g(n))$ (pronounced "big-oh of g of n") the set of functions:

$O(g(n)) = \{ f(n)$: there exist positive constants c and n_0 such that $0 \leq f(n) \leq c * g(n)$ for all $n \geq n_0$ $\}$

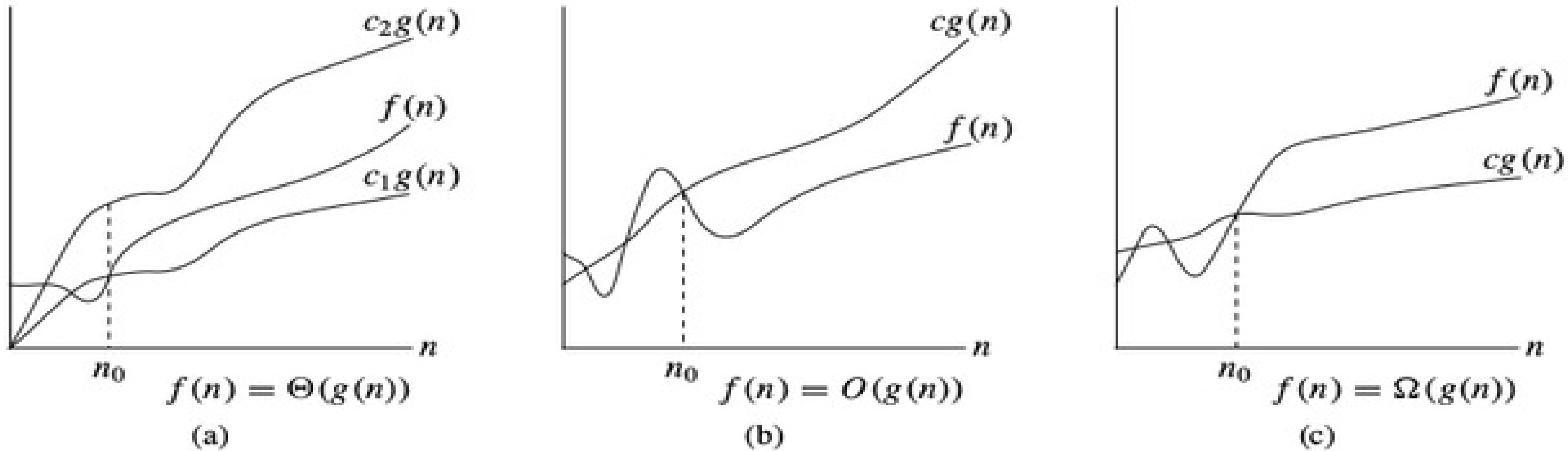


Order of growth: (Contd.)

Ω-notation:

To denote asymptotic lower bound, we use Ω-notation. For a given function $g(n)$, we denote by $\Omega(g(n))$ (pronounced "big-omega of g of n") the set of functions:

$\Omega(g(n)) = \{ f(n)$: there exist positive constants c and n_0 such that $0 \leq c * g(n) \leq f(n)$ for all $n \geq n_0$ }

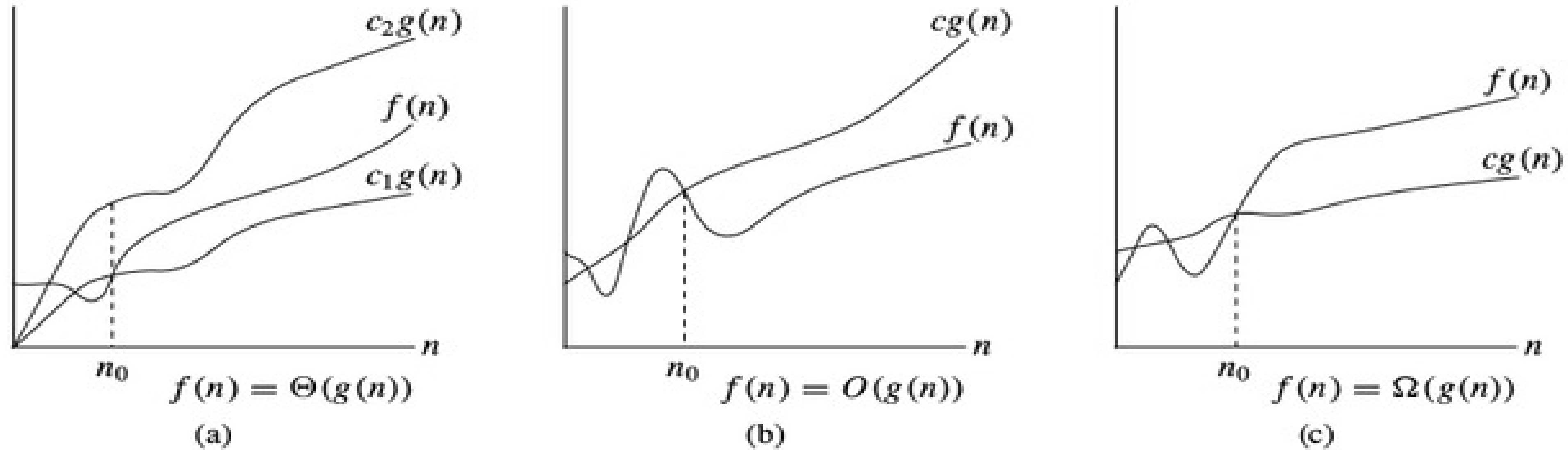


Order of growth: (Contd.)

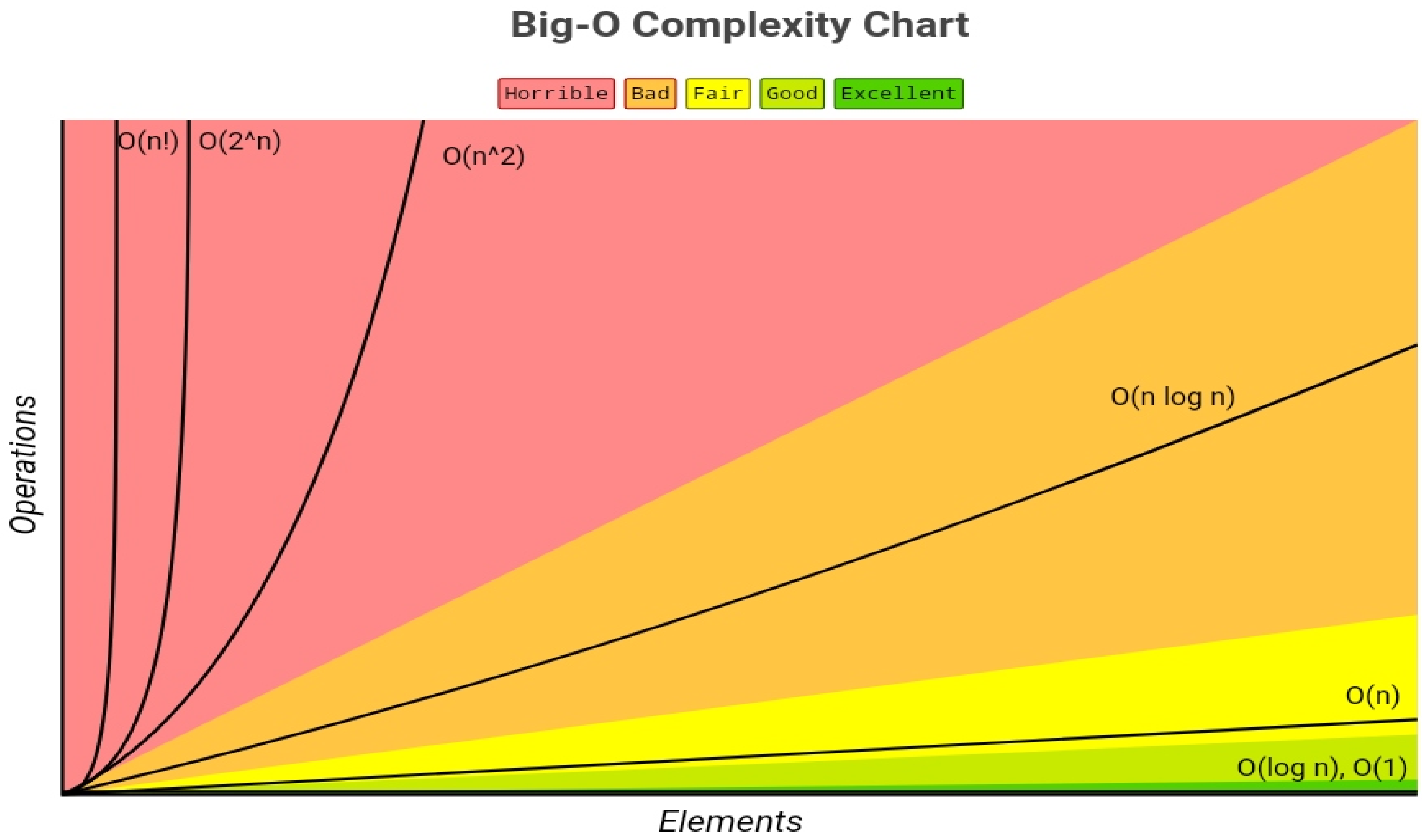
Θ-notation:

To denote asymptotic tight bound, we use Θ-notation. For a given function $g(n)$, we denote by $\Theta(g(n))$ (pronounced "big-theta of g of n") the set of functions:

$\Theta(g(n)) = \{ f(n)$: there exist positive constants $c_1,\ c_2$ and n_0 such that $0 \le c_1 * g(n) \le f(n) \le c_2 * g(n)$ for all $n > n_0$ $\}$



N vs. Growth of Complexity:



Complexity O(1):

Time complexity of a function (or set of statements) is considered as O(1) if it doesn’t contain loop, recursion and call to any other non-constant time function.

- For example swap() function has O(1) time complexity.
- A loop or recursion that runs a constant number of times is also considered as O(1). E.g. the following loop is O(1).

```
for (int i = 1; i <= c; i++) {          // Here c is a constant
    // some O(1) expressions
}
```

Complexity O(n):

Time Complexity of a loop is considered as O(n) if the loop variable is incremented / decremented by a constant amount. For example following functions have O(n) time complexity.

```
for (int i = 1; i <= n; i += c) {    // Here c is a +ve integer constant
    // some O(1) expressions
}
for (int i = n; i > 0; i -= c) {
    // some O(1) expressions
}
```

Complexity $O(n^c)$:

Time complexity of nested loops is equal to the number of times the innermost statement is executed. For example the following sample loops have $O(n^2)$ time complexity

```
for (int i = 1; i <=n; i += c) {
    for (int j = 1; j <=n; j += c) {
      // some O(1) expressions
    }
}
```

```
for (int i = n; i > 0; i -= c) {
    for (int j = i+1; j <=n; j += c) {
      // some O(1) expressions
    }
}
```

For example Selection sort and Insertion Sort have $O(n^2)$ time complexity.

Complexity O(logn):

Time Complexity of a loop is considered as $O(Log_c n)$ if the loop variables is divided / multiplied by a constant amount.

```
for (int i = 1; i <=n; i *= c) {
    // some O(1) expressions
}
for (int i = n; i > 0; i /= c) {
    // some O(1) expressions
}
```

For example Binary Search has $O(Log_2 n)$ time complexity.

Complexity O(LogLogn):

Time Complexity of a loop is considered as $O(LogLog_c n)$ if the loop variables is reduced / increased exponentially by a const amount.

```
for (int i = 2; i <=n; i = pow(i, c)) {    // c is a const greater than 1
    // some O(1) expressions
}
// Here fun is sqrt or cuberoot or any other constant root
for (int i = n; i > 1; i = fun(i)) {
    // some O(1) expressions
}
```

Time complexities of consecutive loops:

Time Complexity When there are consecutive loops, we calculate time complexity as sum of time complexities of individual loops.

```
for (int i = 1; i <=m; i += c) {
    // some O(1) expressions
}
for (int i = 1; i <=n; i += c) {
    // some O(1) expressions
}
```

Time complexity of the code is O(m) + O(n) which is O(m + n)

If m == n, the time complexity becomes O(2n) which is O(n).

Quiz!

1. What is the complexity of the following snippet ? (n can be any integer)

```
for (int i = 0; i < n; i += 1) {
        for (int j = 0; j < m; j +=1) {
            print “Hello”
        }
    }
```

A. O(n)
B. O(m)
C. O(n+m)
D. O(nm)

1. What is the complexity of the following snippet ? (n can be any integer)

```
for (int i = 0; i < n; i += 1) {
        for (int j = 0; j < m; j +=1) {
            print “Hello”
        }
    }
```

A. O(n)
B. O(m)
C. O(n+m)
D. O(nm)

2. What is the complexity of the following snippet ? (n can be any integer)

```
for (int j = 0; j < n; j += 1) {
    for (int i = 0; i < n; i += 1)
        print "First loop"
}
for (int i = 0; i < n; i +=1) {
    print "Second loop"
}
for (int i = 0; i < n; i += 1) {
    print "Third loop"
}
```

A. O(n)
B. O(n^2)
C. Can’t be determined
D. None of these

2. What is the complexity of the following snippet ? (n can be any integer)

```
for (int j = 0; j < n; j += 1) {
    for (int i = 0; i < n; i += 1)
        print "First loop"
}
for (int i = 0; i < n; i +=1) {
    print "Second loop"
}
for (int i = 0; i < n; i += 1) {
    print "Third loop"
}
```

A. O(n)
B. O(n^2)
C. Can’t be determined
D. None of these

3. What is the complexity of the following snippet ? (n can be any integer)

```
while (var < n){
    cout << pie << endl;
    for (int j = 1; j < n; j+=2){
        sum += 1
    }
    var *= 3
  }
```

A. $O(n)$
B. $O(\log n)$
C. $O(n\log_3 n)$
D. O(n^2)

3. What is the complexity of the following snippet ? (n can be any integer)

```
while (var < n){
    cout << pie << endl;
    for (int j = 1; j < n; j+=2){
      sum += 1
    }
    var *= 3
  }
```

A. O(n)
B. O(logn)
C. $O(n\log_3 n)$
D. O(n^2)

In the outer loop we have var *= 3, therefore outer loop will have $\log_3(n)$ complexity but the inner loop proceeds 1 to n, therefore ultimately the whole code will have complexity $n\log_3 n$.

4. What is the complexity of the following snippet ? (n can be any integer)

```
for (int i = 0; i < n; i += 3) {
    for (int j = 0; j < m; j += 2) {
        print ”Hello”
    }
}
```

A. O(n)
B. O(m)
C. O(n+m)
D. O(nm)

4. What is the complexity of the following snippet ? (n can be any integer)

```
for (int i = 0; i < n; i += 3) {
    for (int j = 0; j < m; j += 2) {
        print "Hello"
    }
}
```

A. O(n)
B. O(m)
C. O(n+m)
D. O(nm)

5. What is the complexity of the following snippet (n can be any integer)?

```
for (int i = n/2; i <= n; i += 1) {
    for (int j = 1; j < m; j *= 2) {
        print "Hello"
    }
}
```

A. $O(n\log_2 m)$
B. $O(nm)$
C. $O(\log_2 n * \log_2 m)$
D. None of these

5. What is the complexity of the following snippet (n can be any integer)?

```
for (int i = n/2; i <= n; i += 1) {
    for (int j = 1; j < m; j *= 2) {
        print ”Hello”
    }
}
```

A. $O(n\log_2 m)$
B. $O(nm)$
C. $O(\log_2 n * \log_2 m)$
D. None of these

6. What is the complexity of the following snippet ? (n can be any integer)

```
int i = 1, j = 1
for (; i < n; i += 1) {
    while(j < i) {
        print "Hello"
        j *= 2
    }
    j = 1
}
```

A. $O(\log_2 n)$
B. $O(n)$
C. $O(n\log_2 n)$
D. None of the above

6. What is the complexity of the following snippet ? (n can be any integer)

```
int i = 1, j = 1
for (; i < n; i += 1) {
    while(j < i) {
        print “Hello”
        j *= 2
    }
    j = 1
}
```

A. $O(\log_2 n)$
B. $O(n)$
C. $O(n\log_2 n)$
D. None of the above

7. What is the complexity of the following snippet ? (n can be any integer)

```
for (int i = 1; i < n; i += i) {
        for (int j = 0; j < n; j += 1) {
            print "Hello"
        }
    }
```

A. O(logn)
B. O(nlogn)
C. O(n)
D. O(n^2)

7. What is the complexity of the following snippet ? (n can be any integer)

```
for (int i = 1; i < n; i += i) {
        for (int j = 0; j < n; j += 1) {
            print “Hello”
        }
    }
```

A. O(logn)
B. O(nlogn)
C. O(n)
D. O(n^2)

8. What is the complexity of the following snippet ? (n can be any integer)

```
for (int i = 1; i < n; i *= 3) {
        for (int j = 1; j < n; j += 2) {
            print "Hello"
        }
    }
```

A. $O(\log_3 n)$
B. $O(n\log_3 n)$
C. $O(n)$
D. O(n^2)

8. What is the complexity of the following snippet ? (n can be any integer)

```
for (int i = 1; i < n; i *= 3) {
        for (int j = 1; j < n; j += 2) {
            print "Hello"
        }
    }
```

A. $O(\log_3 n)$
B. $O(n\log_3 n)$
C. $O(n)$
D. $O(n^2)$

9. What is the complexity of the following snippet ? (n can be any integer)

```
int i = 1 , j = 1
for (; i < n; i +=1) {
    while(j < i) {
        print “Hello”
        j *= 2
    }
}
```

A. O(logn)
B. O(n)
C. O(nlogn)
D. None of the above

9. What is the complexity of the following snippet ? (n can be any integer)

```
int i = 1 , j = 1
for (; i < n; i +=1) {
    while(j < i) {
        print “Hello”
        j *= 2
    }
}
```

A. O(logn)
B. O(n)
C. O(nlogn)
D. None of the above

10. What is the complexity of the following snippet ? (n can be any integer)

```
for (int i = 1; i < n; i += 2) {
        for (int j = 1; j < n; j *= 3) {
            print “Hello”
        }
    }
```

A. $O(\log_3 n)$
B. $O(n\log_3 n)$
C. $O(n)$
D. $O(n^2)$

10. What is the complexity of the following snippet ? (n can be any integer)

```
for (int i = 1; i < n; i += 2) {
        for (int j = 1; j < n; j *= 3) {
            print "Hello"
        }
    }
```

A. $O(\log_3 n)$
B. $O(n\log_3 n)$
C. $O(n)$
D. $O(n^2)$