**3. What is the time complexity of**the **following code:**

|  |
| --- |
| **int** i, j, k = 0;  **for** (i = n / 2; i <= n; i++) {  **for** (j = 2; j <= n; j = j \* 2) {          k = k + n / 2;      }  } |

**Options:** 

1. O(n)
2. O(nLogn)
3. O(n^2)
4. O(n^2Logn)

**Output:**

2. O(nLogn)

**Explanation:** If you notice, j keeps doubling till it is less than or equal to n. Several times, we can double a number till it is less than n would be log(n).   
Let’s take the examples here.   
for n = 16, j = 2, 4, 8, 16   
for n = 32, j = 2, 4, 8, 16, 32   
So, j would run for O(log n) steps.   
i runs for n/2 steps.   
So, total steps = O(n/ 2 \* log (n)) = **O(n\*logn)**

**============================================================**

**4. What does it mean when we say that an algorithm X is asymptotically more efficient than Y?**   
**Options:** 

1. X will always be a better choice for small inputs
2. X will always be a better choice for large inputs
3. Y will always be a better choice for small inputs
4. X will always be a better choice for all inputs

2. X will always be a better choice for large inputs

**Explanation:** In asymptotic analysis, we consider the growth of the algorithm in terms of input size. An algorithm X is said to be asymptotically better than Y if X takes smaller time than y for all input sizes n larger than a value n0 where n0 > 0.

**5. What is the time complexity of**the **following code:**

|  |
| --- |
| **int** a = 0, i = N;  **while** (i > 0) {      a += i;      i /= 2;  } |

**Options:** 

1. O(N)
2. O(Sqrt(N))
3. O(N / 2)
4. O(log N)

Output: 

4. O(log N)

**Explanation:** We have to find the smallest x such that N / 2^x N   
x = log(N)

**4. What is the time complexity of the following code :**

* C++
* C

|  |
| --- |
| **void** fun(**int** n)  {  **for**(**int** i=0;i\*i<n;i++)      cout<<"GeeksforGeeks";  }    // This code is contributed by Shubham Singh |

**Solution –**  
Time complexity = O(√n).

**Explanation –**  
Let **k**be the no. of iteration of the loop.

|  |  |
| --- | --- |
| **i** | **i\*i** |
| 1 | 1 |
| 2 | 22 |
| 3 | 32 |
| 4 | 42 |
| … | … |
| k | k2 |

**⇒** The loop will stop when i \* i >=n       i.e.,  i\*i=n  
**⇒** i\*i=n ⇒ k2 = n  
**⇒**k =√n  
Hence, the time complexity is O(√n).