**Quiz Questions on Asymptotic Notation**

**Question 1:**

What does Big-O notation (O(f(n))) describe in terms of algorithm performance?

**Answer:** Big-O notation describes the *upper bound* or worst-case performance of an algorithm. It provides a function that defines the maximum growth rate of an algorithm’s time or space complexity as the input size increases. For example, O(n) means that the algorithm's performance will grow linearly with the size of the input.

**Question 2:**

Which asymptotic notation describes the best-case lower bound of a function?

**Answer:** **Omega notation (Ω(f(n)))** describes the *lower bound* or best-case performance of a function. It indicates that the function will take at least this amount of time or resources in the best possible scenario.

**Question 3:**

What is the primary difference between Big-O (O(f(n))) and Theta (Θ(f(n))) notation?

**Answer:**

* **Big-O (O(f(n)))** describes the *upper bound* of a function, meaning the maximum time/resources an algorithm will take as the input size increases.
* **Theta (Θ(f(n)))** describes both the *upper and lower bounds*, meaning the algorithm’s performance is tightly bounded and grows at the same rate from both directions. In other words, if an algorithm is Θ(f(n)), it grows as f(n) both in the worst and best cases.

**Question 4:**

Which of the following statements is true for a function T(n) that is O(f(n)) and Ω(f(n))?

A) T(n) is o(f(n))  
B) T(n) is Θ(f(n))  
C) T(n) is not Θ(f(n))  
D) T(n) is always smaller than f(n)

**Answer:** **B) T(n) is Θ(f(n))**  
If a function T(n) is both O(f(n)) and Ω(f(n)), it means T(n) is bound by both the upper and lower limits of f(n), which means it is Θ(f(n)).

**Question 5:**

What is the time complexity of binary search using Big-O notation?

**Answer:** The time complexity of binary search is **O(log n)**, where n is the size of the input. Binary search repeatedly divides the search interval in half, which results in logarithmic time complexity.

**Question 6:**

What is the purpose of the little-o notation, o(f(n))?

**Answer:** **Little-o notation (o(f(n)))** describes an upper bound that is not tight. It means that the function T(n) grows strictly slower than f(n) and will never reach the rate of f(n). In other words, T(n) grows asymptotically smaller than f(n), but never equals or exceeds it.

**Question 7:**

Given that an algorithm has a time complexity of O(n^2), which of the following asymptotic notations can also be used to describe its performance?

A) O(n)  
B) Θ(n^2)  
C) Ω(n)  
D) o(n^3)

**Answer:** **B) Θ(n^2)**  
**D) o(n^3)**  
Explanation:

* **Θ(n^2)** means that the algorithm grows at the same rate as n^2.
* **o(n^3)** means the algorithm grows strictly slower than n^3.

**Question 8:**

How does the time complexity of an algorithm with **O(n log n)** compare to an algorithm with **O(n^2)** as n increases?

**Answer:** **O(n log n)** grows much more slowly than **O(n^2)** as n increases. For large inputs, an algorithm with O(n log n) will perform significantly better than one with O(n^2), because n log n grows slower than n^2.

**Question 9:**

Which of the following is a valid example of a function that is both O(n) and Ω(n)?

A) T(n) = n log n  
B) T(n) = 2n  
C) T(n) = n^2  
D) T(n) = 3n^2 + 5n

**Answer:** **B) T(n) = 2n**  
Explanation: T(n) = 2n is linear, so it is both O(n) and Ω(n), as its growth rate is tightly bounded by n in both upper and lower limits.

**Question 10:**

If an algorithm’s time complexity is **O(n^3)**, which of the following is a possible lower bound (Ω) for the algorithm?

A) Ω(n^2)  
B) Ω(n^4)  
C) Ω(1)  
D) Ω(n^3)

**Answer:** **D) Ω(n^3)**  
Explanation: The lower bound of an algorithm with time complexity O(n^3) can be Ω(n^3), meaning the algorithm will take at least n^3 time in the worst case.

**Question 11:**

What is the best-case time complexity for an algorithm that is Θ(n^2)?

**Answer:** **Θ(n^2)** notation represents both the upper and lower bounds of the function. The best-case time complexity for an algorithm with Θ(n^2) is **O(n^2)**, as the algorithm will take quadratic time in the best case as well as in the worst case.

**Question 12:**

What is the relationship between O(1) and O(n^2)?

A) O(1) is better than O(n^2) for large inputs.  
B) O(1) is worse than O(n^2) for large inputs.  
C) O(1) and O(n^2) are equally good for large inputs.  
D) O(1) and O(n^2) are the same.

**Answer:** **A) O(1) is better than O(n^2) for large inputs.**  
Explanation: O(1) denotes constant time complexity, meaning the algorithm will take the same amount of time regardless of input size, while O(n^2) grows quadratically with input size. Therefore, O(1) will always perform better for large inputs.

**Question 13:**

Which asymptotic notation would you use to describe an algorithm that grows faster than **O(n log n)** but slower than **O(n^2)**?

**Answer:** The best notation for this would be **O(n^1.5)** (or **O(n^3/2)**), as it describes a growth rate faster than O(n log n) but slower than O(n^2).

**Question 14:**

What does it mean if an algorithm’s time complexity is **Θ(n log n)**?

A) The algorithm has a linear time complexity.  
B) The algorithm takes time proportional to the input size and log of input size.  
C) The algorithm grows exponentially with input size.  
D) The algorithm is slower than an O(n^2) algorithm.

**Answer:** **B) The algorithm takes time proportional to the input size and log of input size.**  
Explanation: **Θ(n log n)** is typically seen in algorithms like merge sort, where the time grows based on the product of the input size and the logarithm of the input size.

These questions cover various aspects of **asymptotic notation** and help in understanding the performance characteristics of algorithms.