1. Arrays in JavaScript are examples of:
   1. Primitive data structures
   2. Non-primitive data structures
   3. Both
   4. Neither

**Explanation:** Arrays and objects in JavaScript are considered primitive data structures in the sense that they are basic and built-in, allowing you to directly use them without needing to create your own data structures

1. Space complexity in algorithms refers to:
   1. The amount of memory used
   2. The time taken to execute the algorithm
   3. The size of the input data
   4. The number of iterations

**Explanation:** Space complexity measures how the memory requirements of an algorithm grow as the input size increases.

1. Binary search is more efficient than linear search when:
   1. The array is small.
   2. The array is unsorted.
   3. The array is sorted.
   4. The array contains duplicate elements.

**Explanation:** Binary search is more efficient when the array is sorted, allowing for a divide-and-conquer approach.

1. How is a stack implemented in JavaScript?
   1. Using a queue
   2. Using an array
   3. Using recursion
   4. None of the above

**Explanation:** A simple way to implement a stack in JavaScript is by using an array and utilizing push and pop operations.

1. In algorithm analysis, what does the term "best-case time complexity" refer to?
   1. The minimum time required for any input
   2. The average time required for random inputs
   3. The maximum time required for any input
   4. The time required for the worst-case input

**Explanation:** Best-case time complexity refers to the minimum time required for a given algorithm, typically occurring when the algorithm encounters the most favorable input.

1. What is the time complexity of a linear search algorithm in the worst case?
   1. O(1)
   2. O(log n)
   3. O(n)
   4. O(n^2)

**Explanation:** In the worst case, a linear search algorithm may need to iterate through the entire array, resulting in a time complexity of O(n).

1. When would you prefer using a queue over a stack in a specific algorithm?
   1. When the order of processing is critical
   2. When you need to track function calls
   3. When you want to implement undo functionality
   4. When you need to efficiently retrieve the most recently added element

**Explanation:** Queues are suitable when the order of processing elements is essential, as they follow the First In, First Out (FIFO) principle.

1. When analyzing the time complexity of an algorithm, which scenario should you consider?
   1. The average case
   2. The best case
   3. The worst case
   4. All of the above

**Explanation:** When analyzing time complexity, it's essential to consider the average case, best case, and worst case scenarios to have a comprehensive understanding of the algorithm's behavior across different inputs.

1. What is the time complexity of an algorithm that iterates through half of the input elements in each step?
   1. O(log n)
   2. O(n)
   3. O(n^2)
   4. O(sqrt(n))

**Explanation:** If an algorithm halves the input size in each step, it has a logarithmic time complexity, specifically O(log n).

1. What is the primary difference between a stack and a queue?
   1. A stack follows LIFO, while a queue follows FIFO.
   2. A stack follows FIFO, while a queue follows LIFO.
   3. Both follow LIFO.
   4. Both follow FIFO.

**Explanation:** The primary difference between a stack and a queue is the order in which elements are added and removed. A stack follows the Last In, First Out (LIFO) principle, while a queue follows the First In, First Out (FIFO) principle.