ALGORITHM IsSorted(arr, n)

// arr is the array of size n

// Returns true if the array is sorted in ascending order, otherwise false

// Empty or single-element arrays are considered sorted

if n <= 1

return true

// Iterate through the array to check for ascending order

for i = 0 to n - 2

// If the current element is greater than the next element, the array is not sorted

if arr[i] > arr[i + 1]

return false

// If the loop completes without returning false, the array is sorted

return true

Analyze the time complexity of the following code snippet ALGORITHM (A,n) // A is an array of size n fori=1 to n for j=2\*ito n A[i]=A[j]+1

In this code snippet, there are two nested loops. The outer loop runs from **i = 1** to **n**, and the inner loop runs from **j = 2 \* i** to **n**.

1. **Outer Loop (i):**
   * The outer loop runs **n** times.
2. **Inner Loop (j):**
   * The inner loop starts at **j = 2 \* i** and runs up to **n**.
   * In the first iteration of the outer loop (**i = 1**), the inner loop runs from **j = 2 \* 1** to **n**.
   * In the second iteration of the outer loop (**i = 2**), the inner loop runs from **j = 2 \* 2** to **n**.
   * And so on...

The number of iterations of the inner loop can be approximated as follows:

* + For **i = 1**, inner loop runs **n - 2 + 1** times.
  + For **i = 2**, inner loop runs **n - 4 + 1** times.
  + For **i = 3**, inner loop runs **n - 6 + 1** times.
  + ...

The total number of iterations is approximately: (�−2+1)+(�−4+1)+(�−6+1)+…(*n*−2+1)+(*n*−4+1)+(*n*−6+1)+…

This can be simplified to: �+(�−2)+(�−4)+…*n*+(*n*−2)+(*n*−4)+…

The number of terms is approximately �/2*n*/2, so the time complexity of the inner loop is proportional to �(�)*O*(*n*).

1. **Overall Time Complexity:**
   * The outer loop runs �*n* times.
   * The inner loop runs in �(�)*O*(*n*) time.

Therefore, the overall time complexity of the given code snippet is �(�2)*O*(*n*2) since we have nested loops, and the inner loop's time complexity dominates.

What are the advantages of implementing a queue as a circular array rather than a linear array?

Implementing a queue as a circular array has several advantages over using a linear array:

1. **Efficient Space Utilization:**
   * In a circular array implementation, the elements wrap around, allowing for better space utilization. This avoids wasted space at the beginning of the array when dequeues cause the front of the queue to move forward.
2. **Reduced Memory Overhead:**
   * A circular array implementation can reduce the need for dynamic resizing operations, as it can efficiently reuse the space that becomes available at the beginning of the array. This reduces the memory overhead associated with resizing operations.
3. **Faster Enqueue and Dequeue Operations:**
   * Enqueue and dequeue operations in a circular array implementation can be more efficient compared to a linear array. With a circular array, the front and rear pointers can move in a circular fashion without the need to reset them when they reach the end of the array.
4. **Simplified Implementation:**
   * The circular array implementation often simplifies the code for managing the front and rear pointers. In a linear array, when the rear pointer reaches the end, it needs to be reset to the beginning. Circular arrays naturally handle this condition without additional checks.
5. **Improved Cache Locality:**
   * Circular arrays can exhibit better cache locality, especially when there is a need to iterate over the elements of the queue. Elements are more likely to be stored contiguously in memory, improving cache efficiency.
6. **No Physical Shifts:**
   * In a circular array, the logical circular shift of elements is achieved without physically moving the elements in memory. This is in contrast to a linear array where dequeues often involve shifting elements to fill the vacant space.
7. **Smoother Implementation of a Ring Buffer:**
   * Circular arrays are a natural fit for implementing ring buffers, where data is continuously overwritten in a loop. This can be useful in scenarios like streaming data or in systems where the buffer should have a fixed size.

Despite these advantages, it's essential to note that circular arrays may introduce complexity in managing the wrap-around conditions, and care must be taken to avoid off-by-one errors. However, the benefits in terms of efficiency and reduced memory overhead often make circular arrays a preferred choice for implementing queues in certain scenarios.

Top of Form