Q1. \( f(x) = O(\log^2 n) \), \( g(x) = O(n \log n) \) and \( h(x) = O(n) \). Which of the following statements is true?

(A) \( g(x) = O(f(x)) \)
(B) \( h(x) = O(f(x)) \)
(C) \( g(x) = O(h(x)) \)
(D) \( f(x) = O(g(x)) \)

Q2. The time complexity of the \textit{merge} method in the \textit{Merge Sort} algorithm is:

(A) \( O(\log n) \)
(B) \( O(1) \)
(C) \( O(n) \)
(D) none of the above.

Q3. The time complexity of the \textit{Partition} method in the \textit{Quick Sort} algorithm is:

(A) \( O(n) \)
(B) \( O(n^2) \)
(C) \( O(\log n) \)
(D) constant time.
Q4. The following is the code for the dequeue() method for the recursive or linked implementation of a Queue:

```java
public Object dequeue () throws Underflow{
    if (!isEmpty()){
        Object o = first.item;
        <missing line 1.>
        if (isEmpty())
            <missing line 2.>
            return o;
    }
    else throw new Underflow("dequeuing from empty queue");
}
```

The missing lines are:

(A) 1.first = first.successor; 2.last = last.successor;
(B) 1.first = first.successor; 2.last = null;
(C) 1.first = null; 2.last = null;
(D) 1.first.successor = first; 2.last = null;
Q5. The following is the code for previous method in a singly linked list. It shifts the position of the window one position to the left, i.e., previous to the current position:

```java
public void previous (WindowLinked w) throws OutOfBounds {
    if (!isBeforeFirst(w)) {
        Link current = before.successor;
        Link previous = before;
        while (current != w.link) {
            <missing line 1.>
            current = current.successor;
        }
        <missing line 2.>
    } else throw new OutOfBounds ("Calling previous before start of list.");
}
```

The missing lines are:

(A) 1. current = previous; 2. w.link = previous;
(B) 1. previous = current; 2. w.link = current;
(C) 1. previous = current; 2. w.link = previous;
(D) 1. current = previous; 2. w.link = current;

Q6. We want to add an extra method called multiDequeue to the implementation of a Queue. multiDequeue(n) removes n elements from the front of a Queue. If we perform n multiDequeue operations, the amortized cost is:

(A) $O(n)$;
(B) $O(n^2)$;
(C) $O(n^3)$;
(D) $O(1)$;
Q7. The number of edges in a tree with $n$ nodes is:

(A) $n - 1$;  
(B) $n$;  
(C) $n \log n$;  
(D) $\frac{n}{2}$.

Q8. We have a tree with $n$ nodes. Which of the following statements about its height cannot be true?

(A) The height is $O(\log n)$;  
(B) The height is $\frac{n}{2}$;  
(C) The height is $O(n \log n)$;  
(D) The height is 6;

Q9. Which of the following statements is incorrect?

(A) The worst-case complexity of insertion sort is $O(n^2)$;  
(B) The worst-case complexity of merge sort is $O(n \log n)$;  
(C) The worst-case complexity of quick sort is $O(n \log n)$;  
(D) The worst-case complexity of quick sort is $O(n^2)$;
Q10. We want to implement a method called \texttt{popeye(i)} for the \texttt{stack} data structure. Given a stack \texttt{stack1} and an integer \texttt{i}, \texttt{popeye(i)} pops the \texttt{i}-th item from the top of \texttt{stack1} and keeps \texttt{stack1} otherwise as it was before. In other words, the only difference in the state of \texttt{stack1} before and after the \texttt{popeye(i)} operation is that the \texttt{i}th item from the top will be missing. We will assume that underflow will not occur during a \texttt{popeye(i)} operation. We propose two strategies for implementing \texttt{popeye(i)}.

1. We use an additional queue called \texttt{queue1}. We pop the first \texttt{i} − 1 items from the top of \texttt{stack1} one by one and enqueue those items in \texttt{queue1} as we pop each item from \texttt{stack1}. We then pop the \texttt{i}th item from \texttt{stack1} and return it as a result of the execution of the \texttt{popeye(i)} method. We next dequeue each item from \texttt{queue1} and push it onto \texttt{stack1} until \texttt{queue1} is empty.

2. We use an additional stack called \texttt{stack2}. We pop the first \texttt{i} − 1 items from the top of \texttt{stack1} one by one and push those items into \texttt{stack2} as we pop each item from \texttt{stack1}. We then pop the \texttt{i}th item from \texttt{stack1} and return it as a result of the execution of the \texttt{popeye(i)} method. We next pop each item from \texttt{stack2} and push it onto \texttt{stack1}, until \texttt{stack2} is empty.

Which of the following statements is true?

(A) Both strategies correctly implement the \texttt{popeye(i)} method.
(B) Only the first strategy correctly implements the \texttt{popeye(i)} method.
(C) None of the strategies correctly implements the \texttt{popeye(i)} method.
(D) Only the second strategy correctly implements the \texttt{popeye(i)} method.