Data Structures — Exam (Hard)

Subject: Data Structures
Type: Exam
Difficulty: Hard
Total Questions: 10
Generated: 2025-11-28 09:46 UTC

Auto-generated exam (hard) for Data Structures

Question 1 [7 marks] (hard)

For a dynamic array that doubles its capacity upon overflow, explain why the amortized time complexity for an `append` operation is O(1) over a sequence of 'n' operations, despite occasional O(n) reallocations.

Question 2 [8 marks] (hard)

In a Red-Black Tree, after an insertion, if a red node's parent is also red, describe the primary structural operation (excluding simple recoloring) used to resolve this double red violation and briefly state its immediate effect on the tree's balance properties.

Question 3 [9 marks] (hard)

What is the combined amortized time complexity of `find` and `union` operations in a Disjoint Set Union data structure when both path compression and union by rank/size optimizations are applied, and what mathematical function describes this extremely efficient bound?

Question 4 [10 marks] (hard)

Fibonacci Heaps are theoretically efficient priority queues but are rarely used in practice due to their complexity. Provide a comprehensive explanation of the fundamental structure and key operations (Insert, Extract-Min, Decrease-Key, Delete, Union) of a Fibonacci Heap. Analyze the amortized time complexity of these operations, detailing the accounting method or potential function used. Compare and contrast Fibonacci Heaps with Binary Heaps and Binomial Heaps, specifically discussing the trade-offs in terms of theoretical performance,

implementation complexity, and practical applicability. Identify specific scenarios or algorithms where the unique performance characteristics of Fibonacci Heaps would be most beneficial. (600-900 words)

Question 5 [10 marks] (hard)

Designing robust and efficient concurrent data structures is a significant challenge in modern multi-core computing. Elaborate on the fundamental difficulties encountered when adapting traditional sequential data structures for concurrent access. Distinguish between lock-based, lock-free, and wait-free synchronization paradigms, providing a detailed explanation of the underlying mechanisms, guarantees (e.g., progress conditions), and inherent trade-offs (e.g., overhead, contention, starvation, deadlock). Choose a common data structure (e.g., a queue, stack, or hash table) and describe, at a high level, how it could be implemented using a lock-free approach, highlighting the atomic primitives involved (e.g., Compare-And-Swap, Fetch-And-Add) and addressing potential issues like the ABA problem and memory reclamation strategies (e.g., hazard pointers, RCU). (700-1000 words)

Question 6 [8 marks] (hard)

Consider a dynamic array implementation that starts with a capacity of 1. When the array becomes full, its capacity is doubled. If `N` elements are inserted into this array using `push_back` operations, what is the *amortized* time complexity of a single `push_back` operation?

A. O(N)
B. O(log N)
C. O(1)
D. O(N log N)

Question 7 [10 marks] (hard)

A Red-Black Tree with `n` internal nodes (excluding NIL leaves) has a black-height `h_b`. Which of the following statements correctly describes a fundamental property relating the height `h` of the tree to its black-height `h_b`?

A. h <= 2 * h_b
B. h >= 2 * h_b
C. h = h_b + 1
D. h_b <= log2(n+1)

Question 8 [10 marks] (hard)

Consider a network G=(V, E) with source s and sink t, and capacities `c(u,v)` for each edge `(u,v)`. If `(A, B)` is an `s-t` cut, where `s ∈ A` and `t ∈ B`, and `f` is a feasible flow, which of the following statements is always true according to the Max-Flow Min-Cut Theorem?

A. value(f) = capacity(A, B) for any s-t cut (A, B).
B. value(f) <= capacity(A, B) for any feasible flow f and any s-t cut (A, B).
C. value(f) = max_flow if and only if capacity(A, B) = min_cut.
D. max_flow < min_cut if the network has parallel edges.

Question 9 [8 marks] (hard)

In a Fibonacci Heap containing `N` elements, what is the *amortized* time complexity for a `DECREASE-KEY` operation?

A. O(log N)
B. O(N)
C. O(1)
D. O(log log N)

Question 10 [8 marks] (hard)

Cuckoo Hashing uses multiple hash functions (typically two) to resolve collisions. When inserting an element `x`, if `h1(x)` is occupied, `x` displaces the existing element `y`, which then attempts to move to its alternative location `h2(y)`. If `h2(y)` is also occupied, `y` displaces `z`, and the process continues. What is the primary advantage of Cuckoo Hashing over open addressing schemes like linear probing or double hashing, assuming a low load factor?

A. Guaranteed O(1) worst-case search time.
B. Eliminates the need for multiple hash functions.
C. Requires less memory due to no explicit pointers.
D. Provides O(1) amortized insertion time even with high load factors.