
Category 1 — Lock-Free Data Structures (10 problems)

1. Implement a **lock-free SPSC queue** using a ring buffer.
2. Implement a **lock-free MPSC queue** with minimal memory fences.
3. Build a **lock-free stack** using Treiber's algorithm and safe memory reclamation.
4. Create a **lock-free freelist allocator** for fixed-size objects.
5. Implement a **wait-free counter** for multiple threads incrementing atomically.
6. Build a **lock-free priority queue** using a skiplist.
7. Implement a **lock-free circular buffer** with wrap-around indexing.
8. Build a **lock-free reference-counted pointer**.
9. Implement a **lock-free hash map** with insert/delete support.
10. Build a **MPMC queue** with hazard pointers for safe reclamation.

Category 2 — Cache Optimization & Memory Layout (10 problems)

11. Implement a **cache-aligned struct** to reduce false sharing.
12. Benchmark **struct-of-arrays vs array-of-structs** for cache locality.
13. Implement a **prefetching iterator** for large vectors.
14. Build a **memory-pool allocator** reducing heap fragmentation.
15. Implement a **packed struct** to fit multiple objects per cache line.
16. Create a **ring buffer with cache-line padding**.
17. Implement a **cache-aware vector traversal**.
18. Build a **concurrent hash map** with per-bucket cache-aligned storage.
19. Implement a **lock-free circular buffer** minimizing cache-line bouncing.
20. Measure latency for **aligned vs unaligned memory allocations**.

Category 3 — Threading & Concurrency (10 problems)

21. Implement a **low-latency thread pool** with work-stealing queues.
22. Coordinator for M producer threads and N consumer threads with minimal context switches.
23. Implement a **barrier synchronization primitive** optimized for low contention.
24. Build a **scalable latch** using atomics and spin-wait.
25. Implement a **timed waitable event** with spinning fallback.

26. Build a **scalable read-write lock** optimized for read-heavy workloads.
27. Implement a **thread-local object pool**.
28. Build a **lock-free logging queue** for multiple threads.
29. Implement **priority-aware scheduling** of tasks across worker threads.
30. Build a **cooperative multitasking system** using fibers.

Category 4 — Network & IO (10 problems)

31. Implement a **non-blocking TCP server** using epoll/kqueue.
32. Build a **high-throughput UDP packet processor**.
33. Implement a **zero-copy ring buffer** for inter-thread messaging.
34. Build a **high-performance async logger** writing to disk.
35. Implement **batched socket send/receive** to reduce syscalls.
36. Implement a **latency-sensitive message queue** between threads.
37. Build a **lock-free event dispatcher** for network events.
38. Implement a **connection pool** with low-latency handoff.
39. Implement a **polling-based timer wheel** for many timers.
40. Build a **backpressure-aware network pipeline** minimizing allocations.

Category 5 — Real-Time Algorithmic Challenges (10 problems)

41. Implement **sliding window maximum** using deque.
42. Build a **real-time priority queue** with fast insertion/removal.
43. Implement **fixed-size rolling aggregation** (sum, min, max).
44. Build a **median-of-stream calculator** using two heaps.
45. Implement **high-throughput histogram computation** with per-thread buckets.
46. Build a **cache-efficient bloom filter**.
47. Implement a **lock-free token bucket rate limiter**.
48. Build a **high-performance periodic task scheduler**.
49. Implement **parallel reduction** on large arrays using execution policies.
50. Build a **microsecond-level time-series aggregator**.

Category 6 — Low-Latency Data Structures (10 problems)

51. Implement a **fixed-size circular deque** optimized for cache lines.
52. Build a **skiplist for low-latency search and insert**.
53. Implement a **pre-allocated linked list** to minimize allocations.
54. Build a **heap-based event scheduler** optimized for high throughput.
55. Implement a **priority vector** for top-K element retrieval.
56. Build a **fast bitset container** for real-time flag manipulation.
57. Implement a **low-latency string pool** with contiguous allocation.
58. Build a **memory-mapped array** for high-speed access.
59. Implement a **circular buffer of fixed-size packets**.
60. Build a **queue with batched dequeue** for high-performance consumers.

Category 7 — Message Passing & Inter-Thread Communication (10 problems)

61. Implement **lock-free mailbox per thread** for task dispatch.
62. Build a **single-producer multi-consumer message queue**.
63. Implement **multi-producer single-consumer message queue** with atomic pointers.
64. Build a **batched message queue** for low-latency data aggregation.
65. Implement a **multi-level queue** for prioritizing messages.
66. Build a **shared memory message passing mechanism**.
67. Implement **ring buffer for event-driven communication**.
68. Build a **double-buffering system** for inter-thread communication.
69. Implement **priority-aware messaging** between threads.
70. Build a **high-throughput signal/event dispatcher**.

Category 8 — Low-Latency Logging & Persistence (10 problems)

71. Implement **asynchronous log writer** with minimal allocations.
72. Build a **lock-free log buffer** for multi-threaded applications.
73. Implement **batched disk logging** to minimize syscall latency.
74. Build **zero-copy logging** using memory-mapped files.
75. Implement **per-thread log buffers** to avoid contention.
76. Build **log rotation system** without blocking producers.
77. Implement **timestamped event logger** for sub-microsecond precision.

78. Build **in-memory transaction log** with lock-free writes.
79. Implement **rolling log buffer** with overwrite policy.
80. Build a **high-performance metrics collector** writing asynchronously to disk.

Category 9 — Low-Latency Collections & Containers (10 problems)

81. Implement **fixed-capacity vector** that avoids heap allocations.
82. Build **pre-allocated pool of hash maps** for repeated usage.
83. Implement **low-latency set** using contiguous memory.
84. Build **batch-updating priority queue** for high-frequency inserts.
85. Implement **low-latency map** using sorted vectors.
86. Build **lock-free deque** for high-throughput producer-consumer.
87. Implement **fast circular buffer for network packets**.
88. Build **memory-efficient string container** for repeated allocations.
89. Implement **chunked vector container** for reducing allocation overhead.
90. Build **pre-allocated graph adjacency list** for low-latency traversal.

Category 10 — Micro-Optimizations & Profiling (10 problems)

91. Measure **cache-miss impact** for different container layouts.
92. Optimize **vector resize patterns** for low-latency insertion.
93. Benchmark **atomic vs mutex-based counter** under high contention.
94. Implement **branchless algorithms** for low-latency numeric computation.
95. Measure **false sharing impact** in multithreaded containers.
96. Optimize **ring buffer for pointer vs index access**.
97. Implement **fast memory copy using SIMD instructions**.
98. Profile and reduce **allocation latency in container of objects**.
99. Optimize **multi-threaded histogram update** for minimal locking.
100. Benchmark **execution policies (par, par_unseq) vs single-threaded** for vector transforms.