Category 1 — Containers (10 problems)

1. Implement a **thread-safe bounded queue** using std::deque + std::mutex + condition_variable.
2. Build an **LRU cache** using std::list and std::unordered_map.
3. Implement a **multi-index container** supporting lookups by two different keys using std::map + std::unordered_map.
4. Design a **sparse matrix** using std::map<std::pair<int,int>, double>.
5. Create a **fixed-size stack** using std::array with overflow handling.
6. Implement a **ring buffer** using std::vector with wrap-around indexing.
7. Build a **history buffer** for last N operations using std::deque.
8. Implement a **priority queue with dynamic priorities** using std::vector + heap algorithms.
9. Create a **multi-set maintaining insertion order** using std::list + std::multiset.
10. Implement a **circular doubly-linked list** using std::list but with custom iterators.

Category 2 — Iterators (10 problems)

11. Implement a **custom reverse iterator** for std::vector without rbegin().
12. Create a **filter iterator** to lazily skip elements based on a predicate.
13. Implement a **transforming iterator** applying a function lazily.
14. Build a **zip iterator** that iterates two containers simultaneously.
15. Implement a **stride iterator** skipping every N elements.
16. Create a **flatten iterator** for std::vector<std::vector<int>>.
17. Build a **cycle iterator** looping infinitely over a container.
18. Implement a **pairwise iterator** yielding consecutive element pairs.
19. Create a **reverse filter iterator** traversing backward.
20. Implement a **tuple of iterators** that iterates multiple containers in parallel.

Category 3 — Algorithms (10 problems)

21. Implement **custom sort with multiple criteria** using std::sort and lambda.
22. Build a **top-K elements extractor** using std::partial_sort.
23. Implement **stable sort on std::list** with custom comparator.
24. Write a **binary search for closest element** using std::lower_bound.
25. Rotate elements N steps in std::vector using std::rotate.

26. Merge two sorted vectors using std::merge.
27. Remove duplicates from a vector using std::sort + std::unique.
28. Partition a vector based on a predicate using std::partition.
29. Compute polynomial evaluation using std::accumulate.
30. Find median using std::nth_element.

Category 4 — Tuples, Variants, Optionals (10 problems)

31. Zip two vectors into a vector of tuples.
32. Write a **variant visitor** for std::variant<int,string,double>.
33. Flatten nested tuples into a single tuple.
34. Concatenate multiple tuples using std::tuple_cat.
35. Transform each tuple element using std::apply.
36. Implement **safe map get** returning std::optional.
37. Compute sum of std::vector<std::optional<int>> ignoring empty values.
38. Convert tuple to std::vector<string>.
39. Variant-based event dispatcher supporting multiple callbacks.
40. Flatten nested std::optional<std::optional<T>> into single optional.

Category 5 — Ranges and Views (10 problems)

41. Use std::views::filter and std::views::transform to process a vector lazily.
42. Implement a **chunked view** that yields fixed-size subranges.
43. Create a **sliding window max** using ranges and deque.
44. Filter out duplicates using std::ranges::unique (or mimic with transform+filter).
45. Compose **multi-step transformations** over a range (filter -> transform -> take).
46. Implement a **take_while view** manually.
47. Implement a **drop_while view** manually.
48. Zip two ranges using custom iterator adaptor.
49. Convert a range of strings to uppercase lazily using views.
50. Implement **flattened range of ranges** (vector<vector>) as a single range.

Category 6 — Custom Comparators and Hashing (10 problems)

51. Custom comparator for std::set sorting strings by length then lexicographically.
52. Thread-safe unordered map wrapper using std::shared_mutex.
53. Priority queue supporting decrease-key using std::set/multiset.
54. Memory-efficient flat_map using std::vector<pair<K,V>> + binary search.
55. Hash container for custom struct with perfect hash.
56. Comparator for multimap grouping elements by first char, secondary by last char.
57. Implement custom comparator for std::sort handling NaN floats.
58. Create custom ordering for std::priority_queue handling complex numbers.
59. Case-insensitive string comparator for std::map.
60. Stable ordering of std::unordered_map insertion using auxiliary vector.

Category 7 — Concurrency with STL (10 problems)

61. Thread-safe LRU cache using std::mutex + std::list + std::unordered_map.
62. Concurrent queue with multiple producers/consumers using std::deque + condition_variable.
63. Implement thread-safe counter with std::atomic + STL containers.
64. Parallel vector transform using std::for_each with execution policies.
65. Thread-safe map update for multiple threads.
66. Implement concurrent histogram using std::vector<std::atomic<int>>.
67. Parallel accumulation of large vector using std::reduce and execution policies.
68. Concurrent bucketed map using vector of maps + per-bucket mutex.
69. Use std::scoped_lock with multiple STL containers.
70. Thread-safe priority queue using STL heap algorithms + mutex.

Category 8 — Performance Optimizations (10 problems)

71. Optimize std::vector resizing patterns for large data insertion.
72. Minimize cache misses in std::map<int, vector<int>> traversal.
73. Compare std::list vs std::deque performance for large insertions.
74. Optimize STL container of unique_ptr for minimal allocations.
75. Compare std::vector vs std::unordered_map for frequent key lookups.
76. Implement **memory pool** allocator for std::vector<Node*>.
77. Implement cache-friendly sorting for std::vector<struct> with large struct size.

78. Optimize multi-stage pipeline using std::transform + std::ranges.
79. Use std::span to reduce copies in a function operating on vector slices.
80. Use move semantics to optimize vector of tuples insertion.

Category 9 — Advanced STL Algorithms (10 problems)

81. Implement **nth element with custom comparator** for complex struct.
82. Implement **partial_sort_copy** for top-K elements to another container.
83. Build **merge without duplicates** using STL algorithms.
84. Implement **stable partition** manually and benchmark.
85. Compute **matrix multiplication** using std::transform and zip iterators.
86. Flatten vector<vector> and sum all elements using STL algorithms.
87. Find longest increasing subsequence using std::lower_bound efficiently.
88. Implement **search_n** for repeated elements and return iterator.
89. Use std::adjacent_find to detect duplicate consecutive elements in vector.
90. Implement **rotate_copy** to shift vector by N steps into a new container.

Category 10 — Miscellaneous STL Challenges (10 problems)

91. Implement **graph BFS/DFS traversal** using std::queue and std::vector.
92. Implement Dijkstra's algorithm using std::priority_queue.
93. Simulate **event-driven scheduler** using std::multiset of timestamps.
94. Implement **undo-redo stack** using std::stack and std::deque.
95. Build **text auto-complete** suggestion system using std::map<string, int>.
96. Implement **versioned vector** storing previous states using STL containers.
97. Implement **merge intervals** using std::vector<pair<int,int>> and sort+merge.
98. Implement **topological sort** using std::vector<vector<int>> adjacency list.
99. Implement **circular buffer with overwrite** policy using std::deque.
100. Implement **median of running stream** using two std::priority_queues.
