ChatGPT

Coding Questions for Quant Entry-Level Interviews

The first stage of many quantitative finance interviews involves a timed coding test. In addition to analytical puzzles and finance questions, candidates are expected to implement algorithms efficiently. A recent guide for aspiring quants notes that **dynamic programming problems appear frequently in quant programming interviews** and highlights the "Best Time to Buy and Sell Stock" problem as a representative example. Another compilation of interview problems from Louisiana State University includes coding tasks such as writing a dynamic-programming solution to the **egg-dropping puzzle**, implementing a **stack using two queues**, computing the **intersection of two sorted arrays** and other algorithmic exercises [1] [2]. These sources illustrate that quant interviews blend classic software engineering questions with problems drawn from probability, statistics and financial modelling.

The list below collates more than 200 coding questions that are commonly asked in entry-level quant interviews. Questions are grouped by priority: high-priority items are fundamental algorithms and data-structures that appear regularly, medium-priority items are important but less ubiquitous, and low-priority items are specialised or advanced topics. The priority ordering is informed by the frequency with which these problems appear in curated interview lists (e.g., **Grind 75**, **NeetCode 150**) and quant-specific compilations [1]. Candidates should begin with the high-priority set, then move on to medium and low-priority questions as time permits.

High-priority questions (fundamentals)

These problems cover core data structures and algorithms that every quantitative developer should master. They dominate most curated interview lists and feature in many quant programming rounds. Where applicable, LeetCode titles are referenced for clarity.

1. **Two Sum** – find two numbers in an array that add up to a target.
2. **Valid Anagram** – determine whether two strings are anagrams of each other.
3. **Contains Duplicate** – check whether a list contains any duplicate elements.
4. **Product of Array Except Self** – compute an array where each element equals the product of all other elements.
5. **Maximum Subarray** – find the contiguous subarray with the largest sum.
6. **Maximum Product Subarray** – find the contiguous subarray with the largest product.
7. **Find Minimum in Rotated Sorted Array** – locate the minimum element in a rotated sorted list.
8. **Search in Rotated Sorted Array** – perform a binary search on a rotated sorted array.
9. **3Sum** – find all unique triplets that sum to zero.
10. **Container With Most Water** – compute the maximum area formed between vertical lines.
11. **Valid Palindrome** – determine whether a string is a palindrome, ignoring non-alphanumeric characters.
12. **Trapping Rain Water** – compute the amount of water that can be trapped between bars.
13. **Best Time to Buy and Sell Stock** – maximise profit by choosing a single buy/sell pair.
14. **Longest Substring Without Repeating Characters** – find the length of the longest substring with all unique characters.
15. **Palindromic Substrings** – count all palindromic substrings in a string.
16. **Longest Palindromic Substring** – find the longest palindromic substring.

17. **Median of Two Sorted Arrays** – find the median of two sorted arrays.
18. **Longest Repeating Character Replacement** – find the length of the longest substring that can be made of one character with limited replacements.
19. **Permutation in String** – check if a permutation of one string appears in another.
20. **Minimum Window Substring** – find the minimum window in a string which contains all characters of another string.
21. **Sliding Window Maximum** – find the maximum in each sliding window of size *k*.
22. **Valid Parentheses** – check whether brackets are balanced.
23. **Min Stack** – design a stack that supports retrieval of the minimum element.
24. **Generate Parentheses** – generate all combinations of well-formed parentheses.
25. **Daily Temperatures** – compute days until a warmer temperature for each day.
26. **Evaluate Reverse Polish Notation** – compute the value of an expression in postfix notation.
27. **Largest Rectangle in Histogram** – find the area of the largest rectangle in a histogram.
28. **Car Fleet** – determine how many car fleets will arrive at the destination.
29. **Binary Search Tree Iterator** – design an iterator over a binary search tree.
30. **Basic Calculator II** – evaluate an arithmetic expression with +, –, *, /.
31. **Binary Search** – implement binary search on a sorted array.
32. **Search 2D Matrix** – search for a target in a matrix sorted row- and column-wise.
33. **Koko Eating Bananas** – minimise the integer speed at which all bananas can be eaten before an hour limit.
34. **Reverse Linked List** – reverse a singly linked list.
35. **Merge Two Sorted Lists** – merge two sorted linked lists.
36. **Reorder List** – reorder a linked list in a specific interleaving fashion.
37. **Remove Nth Node From End of List** – delete the *n*th node from the end of a linked list.
38. **Copy List With Random Pointer** – clone a linked list where each node has an extra random pointer.
39. **Add Two Numbers** – add two numbers represented as linked lists.
40. **Linked List Cycle** – detect if a linked list has a cycle.
41. **Intersection of Two Linked Lists** – find the node where two linked lists intersect.
42. **LRU Cache** – design a least recently used cache.
43. **Maximum Depth of Binary Tree** – compute the maximum depth of a tree.
44. **Same Tree** – determine if two binary trees are identical.
45. **Invert Binary Tree** – invert a binary tree.
46. **Binary Tree Maximum Path Sum** – find the maximum path sum in a binary tree.
47. **Binary Tree Level Order Traversal** – traverse a tree level by level.
48. **Subtree of Another Tree** – check if one tree is a subtree of another.
49. **Construct Binary Tree from Preorder and Inorder Traversal** – build a tree from traversal orders.
50. **Validate Binary Search Tree** – check whether a tree is a valid BST.
51. **Kth Smallest Element in a BST** – find the *k*th smallest element in a BST.
52. **Lowest Common Ancestor of a Binary Tree** – find the lowest common ancestor of two nodes in a binary tree.
53. **Implement Trie (Prefix Tree)** – implement a trie supporting insert, search and prefix search.
54. **Design Add and Search Words Data Structure** – implement a word dictionary supporting wildcards.
55. **Word Search II** – find all words in a 2-D board.
56. **Kth Largest Element in an Array** – find the *k*th largest element.
57. **Task Scheduler** – schedule tasks with cooling time.
58. **Design Twitter** – implement a simplified Twitter with follow/unfollow and feed.
59. **Find Median from Data Stream** – continuously add numbers and obtain the median.
60. **Merge K Sorted Lists** – merge *k* sorted linked lists.

61. **Top K Frequent Elements** – find the *k* most frequent elements in an array.
62. **Subsets** – return all possible subsets of a set.
63. **Combination Sum** – find combinations of numbers that sum to a target.
64. **Permutations** – generate all possible permutations of a list.
65. **Word Search** – determine if a word exists in a board.
66. **Palindrome Partitioning** – partition a string into all palindromic substrings.
67. **N-Queens** – place *N* queens on an *N*×*N* chessboard without attacking each other.
68. **Number of Islands** – count the number of islands in a grid.
69. **Clone Graph** – clone an undirected graph.
70. **Max Area of Island** – compute the area of the largest island.
71. **Pacific Atlantic Water Flow** – find cells from which water can flow to both the Pacific and Atlantic oceans.
72. **Surrounded Regions** – capture surrounded regions on a board.
73. **Rotting Oranges** – determine the time for all oranges to rot.
74. **Course Schedule** – determine if all courses can be completed given prerequisites.
75. **Course Schedule II** – return a topological ordering of courses.
76. **Alien Dictionary** – determine the order of letters in an alien language.
77. **Graph Valid Tree** – determine if an undirected graph forms a tree.
78. **Number of Connected Components in an Undirected Graph** – count connected components.
79. **Climbing Stairs** – count distinct ways to climb a staircase (dynamic programming).
80. **House Robber** – maximise the sum of non-adjacent houses.
81. **House Robber II** – circular version of House Robber.
82. **Longest Palindromic Subsequence** – find the longest palindromic subsequence.
83. **Decode Ways** – count how many ways a numeric string can be decoded.
84. **Coin Change** – minimum number of coins to make a given amount.
85. **Longest Increasing Subsequence** – length of the longest increasing subsequence.
86. **Partition Equal Subset Sum** – determine if a set can be partitioned into two equal subsets.
87. **Unique Paths** – count distinct paths in a grid.
88. **Longest Common Subsequence** – length of longest common subsequence between two strings.
89. **Word Break** – determine if a string can be segmented into dictionary words.
90. **Combination Sum IV** – count the number of combinations that sum to a target.
91. **Minimum Path Sum** – minimal sum of numbers along a path in a grid.
92. **Edit Distance** – minimum number of operations to convert one string into another.
93. **Maximal Square** – largest square containing only 1s in a binary matrix.
94. **Burst Balloons** – maximise coins earned by bursting balloons.
95. **Jump Game** – determine if you can reach the last index.
96. **Jump Game II** – minimum number of jumps to reach the end.
97. **Gas Station** – find a starting gas station so that a circular trip can be completed.
98. **Hand of Straights** – rearrange cards into groups of consecutive cards.
99. **Merge Intervals** – merge overlapping intervals.
100. **Insert Interval** – insert a new interval into a list and merge if necessary.
101. **Non-overlapping Intervals** – erase the minimum number of intervals to eliminate overlap.
102. **Meeting Rooms** – determine if a person could attend all meetings.
103. **Meeting Rooms II** – find the minimum number of meeting rooms required.
104. **Rotate Image** – rotate a matrix by 90 degrees.
105. **Spiral Matrix** – return elements of a matrix in spiral order.
106. **Set Matrix Zeroes** – set an entire row/column to zero if an element is zero.
107. **Happy Number** – determine if a number is a happy number.
108. **Plus One** – add one to an integer represented as an array.
109. **Pow(x, n)** – implement power function.

110. **Multiply Strings** – multiply two large numbers represented as strings.
111. **Rotate Array** – rotate an array to the right by *k* steps.
112. **Single Number** – find the element that appears once.
113. **Number of 1 Bits** – count the number of 1 bits in an integer.
114. **Counting Bits** – count bits for all numbers from 0 to *n*.
115. **Reverse Bits** – reverse bits of an integer.
116. **Missing Number** – find the missing number in the range 0 … n.
117. **Sum of Two Integers** – add two integers without using the + operator.
118. **Reverse Nodes in *k*-Group** – reverse nodes of a linked list *k* at a time.
119. **Longest Valid Parentheses** – longest valid parentheses substring.
120. **Regular Expression Matching** – match a string against a pattern with '.' and '*'.
121. **Wildcard Matching** – match a string against a pattern with '?' and '*'.
122. **Word Ladder** – transform one word to another by changing one letter at a time.
123. **Word Ladder II** – return all shortest transformation sequences.
124. **Serialize and Deserialize Binary Tree** – convert a tree to a string and back.
125. **Palindrome Pairs** – find all pairs of indices such that concatenation forms a palindrome.
126. **Maximal Rectangle** – largest rectangle containing only 1s in a binary matrix.
127. **First Missing Positive** – find the smallest missing positive integer.
128. **N-Queens II** – count the total number of distinct N-Queens solutions.
129. **Letter Combinations of a Phone Number** – generate all letter combinations for a digit string.
130. **01 Matrix** – distance from each cell to the nearest zero.
131. **K Closest Points to Origin** – find the *k* closest points to the origin in a plane.
132. **Minimum Height Trees** – find all roots of minimum height trees in an undirected graph.
133. **Find All Anagrams in a String** – find all start indices of anagrams of a string.
134. **Binary Tree Right Side View** – right-side view of a binary tree.
135. **String to Integer (atoi)** – convert a string to an integer.
136. **Basic Calculator** – evaluate an arithmetic expression containing parentheses.
137. **Maximum Profit in Job Scheduling** – schedule jobs to maximise profit.
138. **Balanced Binary Tree** – determine if a binary tree is height-balanced.
139. **Flood Fill** – perform flood-fill on an image.
140. **Add Binary** – add two binary strings.
141. **Diameter of Binary Tree** – compute the longest path in a tree.
142. **Implement Queue using Stacks** – simulate a queue using two stacks.
143. **First Bad Version** – find the first bad version of a product.
144. **Ransom Note** – determine if a ransom note can be constructed from letters.
145. **Longest Palindrome** – find the longest palindromic substring (duplicate of 16, emphasised as a separate practise item).
146. **Majority Element** – find the element that appears more than n/2 times.
147. **Middle of the Linked List** – find the middle of a linked list.
148. **Balanced Parentheses Generation** – generate all balanced parentheses (duplicate of 24, but widely practised).
149. **Queue using Stacks** – alternative implementation emphasising two stacks (duplicate of 142, but again frequently asked).
150. **Anagram Detection** – determine if two strings are anagrams (duplicate of 2 but widely practised).

Medium-priority questions (important but less frequent)

These problems are still common but may appear less often in quant interviews. They often test deeper understanding of algorithms or introduce more complex data structures. Many dynamic-programming

puzzles fall into this category, reflecting the advice that quant interviews often focus on dynamic programming.

1. **Egg Dropping Puzzle** – generalise the two-egg problem: for *k* eggs and *n* floors, find the minimum number of trials needed to determine the highest safe floor using dynamic programming [1].
2. **Stack with Two Queues** – implement a stack using two queues; also implement using a single queue (inefficient method) [2].
3. **Intersection of Two Sorted Arrays** – given two sorted arrays with $m \ll n$, find their intersection in O(n log(m/n)) time [3].
4. **Generate All Valid Anagrams** – given a dictionary and a multiset of letters, generate all valid words (anagrams).
5. **Integer Partition** – write a recursive function to compute the number of partitions of a positive integer [4].
6. **Find Odd Ball with Three Weighings** – with 12 balls and a balance scale, find the ball with different weight and determine if it is lighter or heavier.
7. **Bayesian Coin Bias** – given two biased coins with unknown biases, compute the probability that one is more biased than the other.
8. **Expected Draws to Get an Ace** – expected number of draws from a standard deck until the first Ace appears [5].
9. **Ants Walking on a Stick** – given ants moving on a stick and turning around when they meet, determine when all ants fall off [6].
10. **Weighted Random Sampling** – design a data structure to sample items with probability proportional to their weights [7].
11. **Maximise Sharpe Ratio Hypothesis Test** – given a claimed Sharpe ratio and observed performance, determine at what sample size the hypothesis is rejected [8].
12. **Prevent Overfitting Methods** – discuss techniques to prevent overfitting: data augmentation, regularization, model complexity, cross-validation, etc. [9].
13. **Uniform Sampling from a Disk** – devise methods to sample uniformly from a disk and do it without square roots [10].
14. **Eigenvalues of Special Matrix** – find eigenvalues of an n×n matrix with *n* on the diagonal and 1 elsewhere [11].
15. **Conditional Distribution of Normals** – given i.i.d. normals X and Y, find the distribution of X given X + Y > 0 [12].
16. **Expected Cycles in Permutation** – find expected number of cycles longer than n/2 in a random permutation [13].
17. **Sample Uniformly from Unit Vectors** – given n unit vectors in $\mathbb{R}^n$, find a vector that makes the same angle to all of them (Gram–Schmidt variant) [14].
18. **Uniform Sampling of a Disk Without Square Root** – generate a random point in a unit disk without using square roots [10].
19. **Data Structure for Weighted Distribution** – build a data structure to store an integer-weighted probability distribution and discuss runtime trade-offs [7].
20. **Generate a Number Using 1–9 Once Each** – find a nine-digit number using digits 1–9 exactly once such that the number formed by the first *k* digits is divisible by *k* [15].
21. **Implement Queue Using Two Stacks** – design a queue using two stacks.
22. **Binary Tree Level Order Traversal II** – traverse a tree from bottom up.
23. **Binary Tree Zigzag Level Order Traversal** – traverse a tree in zigzag (alternating) order.
24. **Reconstruct Itinerary** – given a list of airline tickets, reconstruct the itinerary in lexical order.
25. **Course Schedule III** – schedule courses to maximise the number taken subject to deadlines.
26. **Shortest Path in a Weighted Graph (Dijkstra)** – implement Dijkstra's algorithm for shortest paths.

27. **A* Search Algorithm** – implement A* path-finding algorithm with heuristics.
28. **Implement Kruskal's Minimum Spanning Tree** – build minimum spanning tree using a union-find data structure.
29. **Bellman–Ford Algorithm** – compute shortest paths in graphs with negative weights.
30. **Topological Sort** – produce a topological ordering of a directed acyclic graph.
31. **Longest Increasing Path in a Matrix** – find the longest path in a matrix where each step must go to a strictly higher number.
32. **Number of Provinces** – count connected components in a matrix (graph) representation.
33. **Accounts Merge** – merge accounts with overlapping email addresses.
34. **Time Based Key-Value Store** – implement key-value store supporting retrieval of values by timestamp.
35. **Evaluate Boolean Expression** – evaluate a boolean expression with parentheses and operators.
36. **Design File System** – implement a simple file system with create and read operations.
37. **Maximum Subarray Sum with K Concatenations** – compute the maximum subarray sum in an array concatenated *k* times.
38. **Permutation in Array (Heap's algorithm)** – generate all permutations of an array using Heap's algorithm.
39. **Randomised Set** – design a set that supports insert, delete and random retrieval in O(1).
40. **Sudoku Solver** – solve a 9×9 Sudoku puzzle using backtracking.
41. **String Compression** – compress a string in place (LeetCode 443).
42. **Random Pick With Weight** – pick an index at random with weights.
43. **Kth Smallest Element in a Sorted Matrix** – find the *k*th smallest element in a sorted matrix.
44. **Permutation Sequence** – find the *k*th permutation sequence of numbers 1–n.
45. **Basic Calculator III** – evaluate an expression with +, –, *, / and parentheses.
46. **Snakes and Ladders** – find minimum number of dice throws to reach the final square.
47. **Game of Life** – implement Conway's Game of Life.
48. **Counting Prime Numbers (Sieve)** – count primes less than a non-negative integer.
49. **Random Point in Non-Overlapping Rectangles** – pick a random integer point inside non-overlapping rectangles.
50. **Maximal Network Rank** – compute the maximal network rank in a graph.

Low-priority questions (advanced or quant-specific)

These questions tend to be specialised or are asked less frequently. They often integrate financial mathematics and numerical methods or explore unusual puzzles. Candidates aiming for quantitative developer roles should attempt these after mastering the high- and medium-priority lists.

1. **Implement Quicksort** – in-place quicksort implementation.
2. **Implement Mergesort** – stable O(n log n) sorting algorithm.
3. **Implement Heapsort** – in-place O(n log n) heap sort.
4. **Implement Insertion Sort** – straightforward $O(n^2)$ sorting algorithm.
5. **Implement Bubble Sort** – demonstrate simple sorting; understand time complexity.
6. **Implement Selection Sort** – selection sort algorithm.
7. **Breadth-First Search (BFS) and Depth-First Search (DFS)** – traverse graphs using BFS and DFS; compute connected components.
8. **Thread-Safe Queue** – implement a concurrent queue (e.g., using locks or atomics).
9. **Compute Option Greeks** – write functions to compute delta, gamma, vega and theta of a European option using closed-form Black–Scholes formulas [16].
10. **Sharpe Ratio Calculation** – calculate the Sharpe ratio of a return series and test its statistical significance [8].

11. **Monte Carlo Simulation for Option Pricing** – simulate geometric Brownian motion to price European options and estimate standard error [17].
12. **Binomial Tree for American Options** – build a binomial or trinomial tree to price American options and compare with Black–Scholes [18].
13. **Newton–Raphson Implied Volatility** – implement the Newton–Raphson method to compute implied volatility from option prices.
14. **Brownian Motion Simulator** – simulate paths of Brownian motion and verify properties (zero drift, variance proportional to time).
15. **Geometric Brownian Motion Simulator** – simulate stock price paths under the GBM model.
16. **Risk Parity Portfolio Algorithm** – implement an algorithm to allocate capital based on risk parity.
17. **Calculate Correlation Matrix** – compute correlation matrix of asset returns and cluster stocks using k-nearest-neighbours [19].
18. **Principal Component Analysis** – implement PCA to reduce dimensionality of a data set.
19. **Ordinary Least Squares Regression** – implement OLS regression and compute coefficients and residuals [20].
20. **Maximum Likelihood and MAP Estimation** – write functions to compute MLE and MAP estimates for simple distributions [20].
21. **Logistic Regression** – implement logistic regression using gradient descent.
22. **k-Nearest Neighbours** – implement k-NN classification or regression and discuss curse of dimensionality.
23. **Support Vector Machine Classifier** – implement a simple SVM using a library or from scratch; the LSU compilation references support-vector machines [21].
24. **Design Connect 4 AI** – program an AI to play Connect 4 using minimax with pruning [22].
25. **Knight's Tour via Neural Network or HMM** – design a neural network or hidden-Markov model to solve the knight's tour problem [23].
26. **Anagram Finder with Dictionary** – given a dictionary and a string, generate all valid anagrams [24].
27. **Population Markov Chain** – given a population that dies, stays, or doubles with certain probabilities, determine long-term behaviour using Markov chains [25].
28. **Lions and Sheep Puzzle** – determine when it is safe for any lion to eat the sheep without being eaten [26].
29. **Skittles Hypothesis Test** – evaluate whether someone can distinguish Skittles flavours by taste [27].
30. **Estimations and Mental Maths** – quick mental calculations (e.g., 0.5382 – 0.332, 23×21, 0.99^{100}, approximate ln 314) [28].
31. **Acquaintance Graph Modelling** – model acquaintance networks and estimate average degree [29].
32. **Uncorrelated but Dependent Variables** – give examples of random variables that are uncorrelated yet dependent [30].
33. **Probability an N-Segment Broken Stick Forms an N-gon** – choose $n-1$ random breakpoints on a stick and find the probability that the resulting segments form an n-gon [31].
34. **Estimation Questions** – weight of a giraffe, number of bankruptcies in a year, largest daily temperature change, etc. Evaluate by making reasonable assumptions [32].
35. **Kelly Criterion Optimisation** – given betting odds and probability, compute the optimal fraction of bankroll to wager [33].
36. **Fourier Filtering of Noisy Signals** – use the Fourier transform to reduce noise in regularly sampled data [34].
37. **Uniform Disk to Normal Distribution (Box–Muller)** – use uniform sampling in a disk to generate normally distributed random variables [35].

38. **Hypothesis Test for Coin Fairness** – evaluate fairness of a coin after 10 000 flips where 5200 heads appear [36].
39. **Derive OLS Solution** – derive closed-form solution for linear regression coefficients [37].
40. **Estimator Properties** – compare unbiasedness and minimum mean-squared-error estimators for mean and variance [38].
41. **Correlation Bounds** – find bounds on correlation of three variables given pairwise correlations [39].
42. **Product of Covariance Matrices** – determine whether the product of two covariance matrices is also a covariance matrix [40].
43. **Coefficient of Determination and Model Complexity** – discuss how R^2 changes as more independent variables are added and why high R^2 may indicate overfitting [41].

Conclusion

Quantitative finance interviews require mastery of classic coding problems and awareness of probabilistic and financial concepts. This extensive catalogue of more than 200 problems offers a structured way to prepare. High-priority questions ensure solid foundations in arrays, hashing, dynamic programming and tree/graph traversal. Medium-priority problems deepen understanding through more intricate algorithms and puzzles drawn from quant interview compilations [1]. Low-priority questions introduce financial models, simulations and statistical estimation, linking coding ability to quantitative finance. By working through these categories in order of priority, candidates can develop the breadth and depth needed to excel in entry-level quant interviews.

---

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] Quant_Interview_Prep.pdf
https://www.math.lsu.edu/~smolinsk/Quant_Interview_Prep.pdf

[16] [17] [18] Quant Interview Questions: Common Topics and Resources | Quant Finance Institute (QFI) posted on the topic | LinkedIn
https://www.linkedin.com/posts/quantitative-finance-institute_quantfinance-financialengineering-quantlife-activity-7395855169680437248-Xnh9