General directions:

Algorithms may be described at "high level" without actual code. You may use any lower bound, algorithm or data structure from the text or in class, and their correctness and analysis, but be careful. For example, if you construct a graph with \( n^2 \) nodes and \( n^3 \) edges and then run Dijkstra’s algorithm on the resulting graph, the total run-time is \( O(n^3 \log n) \), because Dijkstra’s algorithm is \( O(E \log V) \). If you want to use the correctness of Dijkstra’s algorithm on the graph, you must check that edge weights are positive, since otherwise the proof that Dijkstra’s algorithm gives shortest paths does not go through. If a problem has multiple size parameters, you should express the run-time as a function of all relevant parameters; e.g., saying maximum bi-partite matching takes time \( O(|V||E|) \) is more accurate than to say it takes time \( O(V^3) \), although both are correct.

For each problem 1-4, you must prove correctness and give a time analysis. Each problem is worth 10 points. Grading may be based on any of the following that are relevant for the problem: the efficiency of your algorithm; the correctness and proof thereof; and the time analysis. The number of points depending on each part is given after the problem.

**Flight scheduling** You are devising a flight scheduler for a travel agency. The scheduler will get a list of available flights, and the customer’s origin and destination. For each flight, it is given the cities and times of departure and arrival. The scheduler should output a list of flights that will take the customer from her origin to her destination that arrives as early as possible, subject to giving her at least one hour for each connection.

a. Give a formal specification for this problem (Instance, Solution Space, Constraints, Objective).

b. Give as efficient as possible an algorithm to solve the problem.

(2 points specification, 4 points correct algorithm, 4 points efficiency. Efficient algorithms can be within log factors of linear time.)

**Non-negative path** (Story about protocols and state space omitted, because some were finding it confusing.) You are given a directed graph with edge weights 1, 0 or -1. A valid path is one where, for all \( k \), the sum of the weights of the first \( k \) edges is non-negative. The problem is, given nodes \( s \) and \( t \) and such a graph \( G \), to determine whether there is a valid path from \( s \) to \( t \).

(Hint: Let the imbalance of the path be the partial sum of the weights. Prove that, if there is a valid path from \( s \) to \( t \), there is one with an upper
bound on the imbalance (where the upper bound is some polynomial in \(n\) and \(m\)) Your running time may well depend on this maximum imbalance.)

**Energy Contracts:** UCSD needs long-term contracts for power to keep the lights on. It has a list of \(n\) bids \(B_I, 1 \leq I \leq n\) from power companies, each with a rate (cost per megawatt) \(r_I\) and a capacity \(C_I\) (the maximum number of megawatts the company can guarantee). The regents, to encourage low bids, has guaranteed that they will pay for all accepted bids at the highest rate of any accepted bid. The accepted bids’ capacities must sum to at least \(M\), the university’s demand for power, to ensure enough power. In addition to the money paid per megawatt, the regents expect each contract to cost a fixed amount \(F\) for lawyers, setting up connections to the grid, etc. So the total cost will be \(Fk + rM\), where \(k\) is the number of accepted bids, and \(r\) is the maximum rate of an accepted bid. Give an efficient algorithm, polynomial-time in \(n\), that, given \(F, M\) and the list of \(n\) bids \(B_I\), computes a subset of bids \(A\) that minimizes the cost to the university subject to ensuring enough power. (3 pts correct poly-time algorithm and correctness proof, 7 pts efficiency and time analysis. My best time is \(O(n \log n)\).)

**Summing triples (20 points)** Let \(A[1,...n]\) be an array of positive integers. A *summing triple* in \(A\) is a set of 3 distinct indices \(1 \leq i, j, k \leq n\) so that \(A[i] + A[j] = A[k]\).

Give and analyze an algorithm that, given \(A\), determines whether there is any summing triple in \(A\). Your algorithm must be faster than \(O(n^3)\).

(3 points correct algorithm and proof of correctness, 7 points efficiency and time analysis).

**Implementation (20 points)** Implement the algorithm you gave for the summing triples problem above. Try it on random arrays where each element \(A[i]\) is chosen in the range \(1...n\), for \(n = 2^6, 2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}\) and \(2^{20}\). Plot its performance on a \(\log_{2n}\) vs. \(\log_2\) of the time scale. Then try the same experiment on random arrays where each element is chosen in the range \(1...n^2\). Do you see a difference? If so, can you explain it?