Directions: For each of the first four problems, a "high level" greedy strategy is given. For some of the problems, the strategies give a correct (optimal) solution, and for others, it sometimes gives incorrect (suboptimal) solutions. For each, decide whether the greedy strategy produces optimal solutions. If it is, give a proof that it is correct, then describe what data structures and preprocessing you would use to give an efficient version, and give a time analysis. If it is not correct, give a counter-example showing the strategy is incorrect. Then give a back-tracking algorithm for the problem, and give an upper bound on the time your algorithm would take.

**Caravan stops** You are organizing a caravan crossing a desert. Your path is fixed. The caravan can only travel \( m \) miles between stops at oases. You are given a list of oases \( \text{Oasis}[1..n] \), each with its distance \( d_i \) in miles from the starting point. The last oasis, \( \text{O_n} \), is the destination. You wish to choose the minimum size set of stops subject to the constraint that there are no more than \( m \) miles between consecutive stops.

Candidate greedy strategy: Treat the start as an oasis with \( d_i = 0 \). At each stop, at oasis \( \text{Oasis}[i] \), if the destination \( d_n \leq d_i + m \), go there directly. Otherwise, find the oasis \( j \) of maximum \( d_j \) subject to \( d_j \leq d_i + m \). Make \( j \) your next stop, and repeat.

**Maximum matching** A matching in an undirected graph is a set of edges \( M \subseteq E \), so that no two edges in \( M \) share a common endpoint, i.e., we cannot have \( \{x, y\} \) and \( \{x, z\} \) both in \( M \) for any three nodes \( x, y \neq z \). The maximum matching problem is to find a largest matching in a given graph.

Candidate greedy strategy: For each edge, sum the degrees of its endpoints, \( s(\{x, y\}) = \deg(x) + \deg(y) \). Put the edge with the minimum such sum in \( M \) and remove both the endpoints from \( G \). Repeat until no edges are left.

**3. Spectrum** You want to create a scientific laboratory capable of monitoring any frequency in the electromagnetic spectrum between \( L \) and \( H \). You have a list of possible monitoring technologies, \( T_i, i = 1..n \), each with an interval \( [l_i, h_i] \) of frequencies that it can be used to monitor. You want to pick as few as possible technologies that together cover the interval \( [L, H] \). Candidate Greedy Strategy: Of intervals that contain \( L \), pick the one \( (l_i, h_i) \) with largest \( h_i \). Set \( L \) to \( h_i \) and recurse.

**Minimum Average Completion Time** You are given a list of \( n \) jobs \( j_1...j_n \), each with a time \( t(j) \) to perform the job, and a weight \( w(j) \). You
wish to order the jobs as \( j_{\sigma(1)}, \ldots, j_{\sigma(n)} \) in such a way as to minimize:
\[
\sum_i w(j_{\sigma(i)}) (\sum_{1 \leq k \leq i} t(j_{\sigma(k)}));
\]
the weighted sum of the time each job has to wait before being performed.

Candidate strategy: First, perform the job \( j \) that maximizes the ratio
\[
r(j) = \frac{w(j)}{t(j)}.
\]
Then order the other jobs recursively.

**Sudoku** The *sudoku* problem of size \( n \) is as follows. The input is an \( n^2 \times n^2 \) matrix \( M \) whose entries are either “blank” or an integer between 1 and \( n^2 \). A solution fills in the blank spaces with integers between 1 and \( n^2 \). The following constraints must be met: Each integer from 1 to \( n^2 \) appears exactly once in each row, in each column, and in each \( n \times n \) sub-matrix of the form \( M[jn + 1..(j + 1)n][in + 1..(i + 1)n] \) for each \( 0 \leq i, j \leq n - 1 \). The problem is to find any solution meeting the constraints, or return “no solution possible” if there is no such solution.

Write a back-tracking algorithm to solve the *sudoku* algorithm, and implement it. Describe your algorithm, and relevant programming information such as language, machine characteristics, etc. Test your algorithm on both 9 \( \times \) 9 and 16 \( \times \) 16 *sudoku*. The San Diego Union Tribune publishes *sudoku* problems in their Marketplace (classified ads) section. Monday through Saturday they publish problems in increasing order of supposed difficulty. Sunday they publish a 16 \( \times \) 16 *sudoku* problem. Report the times for your algorithm to solve many of these puzzles. Does the time for your algorithm increase with the difficulty ratings? (Note: Be careful not to use up too much computer time. Don’t leave programs running unsupervised too long. Depending on your algorithm, you may find even very small sizes take huge amounts of time.)