For each of the algorithm problems, design as asymptotically efficient an algorithm as possible. Give a correctness argument (explanation, if it is relatively simple, or proof if not) and time analysis. You may use any well-known algorithm or data structure, or algorithm from the text or from class, as a sub-routine without needing to provide details.

**Merging lists** You want to collect sorted lists from different databases and merge them into a single sorted list. The cost to merge a list of size $l_1$ with one of size $l_2$ is $l_1 + l_2$, and creates a new list of size $l_1 + l_2$ replacing the old ones. You are given as input the sizes of $n$ lists $l_1, \ldots, l_n$ and need to schedule merges in order to unite them into a single list. You wish to minimize the total cost for all merges. Give an efficient algorithm that, given $l_1, \ldots, l_n$, finds the lowest cost schedule to merge lists of those sizes. (12 pts. correct poly-time alg. and correctness proof, 8 pts. efficiency)

**Homework grade maximization** In a class, there are $n$ assignments. You have $H$ hours to spend on all assignments, and you cannot divide an hour between assignments, but must spend each hour entirely on a single assignment. The $I$th hour you spend on assignment $J$ will improve your grade on assignment $J$ by $B[I, J]$, where for each $J$, $B[1, J] \geq B[2, J] \geq \ldots \geq B[H, J] \geq 0$. In other words, if you spend $h$ hours on assignment $J$, your grade will be $\sum_{i=1}^{h} B[i, J]$ and time spent on each project has diminishing returns, the next hour being worth less than the previous one. You want to divide your $H$ hours between the assignments to maximize your total grade on all the assignments. Give an efficient algorithm for this problem. (6 pts. correct poly-time algorithm, 6 points correctness proof, 8 points efficiency. My best time is $O(n + H \log n)$.)

**Preemptive scheduling** Consider the following preemptive scheduling problem. You are trying to schedule jobs on a machine that are arriving at different times, and require different numbers of steps to finish. Your schedule can be preemptive, in that you can start one job, then switch to another, then finish the first job. You are trying to minimize the sum over all jobs of the time they finish. More precisely, the input is a sequence of $n$ jobs, $Job_i = (a_i, d_i)$, where $a_i$ is an integer giving the arrival time of the job (first time step when we could start the job), and $d_i$ is a positive integer giving the duration of the job, the number of steps required to finish the job. A schedule specifies
for each time step, which job we are working on. At time step, \( t \), we can only work on \( Job_i \) if \( a_i \leq t \); and there must be at least \( d_i \) steps where we are working on \( Job_i \). The finish time for \( Job_i \) is the last time when \( Job_i \) is scheduled. The objective is to find a schedule that minimizes the sum of all the finish times.

Example: Job 1: Arrives at 8 AM: Practice piano. Duration: 3 hours.
Job 2: Arrives at 9 AM. Answer morning email. Duration 1 hour.
Job 3: Arrives at 11 AM. Do CSE homework. Duration 4 hours.
Finish times: email: 10; piano: 12; homework: 16. Total: 38.

Give an efficient greedy algorithm for this problem. (4 points correct algorithm, 10 points correctness proof, 6 points efficiency)

**Approximation for bin filling** In the bin filling problem, you have \( n \) items of positive integral sizes \( a_1, \ldots, a_n \) and \( m < n \) bins, where bin \( j \) is of size \( B_j \). You need to assign each item \( i \) to a bin \( A[i] \), in a way to fill the maximum number of bins, where a bin \( j \) is full if \( \sum_{i \mid A[i] = j} a_i \geq B_j \).

Give an efficient approximation algorithm for this problem. Most of the points will be based on your approximation ratio and the proof that it achieves this ratio. The best possible ratio is to fill at least \( 1/2 \) as many bins as the optimal solution.

**Implementation of Independent Set Heuristic** The independent set problem is to find as large as possible a set of nodes of an undirected graph so that the set does not contain both endpoints of any edge. Consider a greedy heuristic for independent set that selects the lowest degree node, puts it in the set, and deletes it and its neighbors and repeats. Implement this greedy heuristic, and test it on random graphs where every pair of nodes has an edge between them independently with probability \( 1/2 \). Try it on graphs with as wide a range of number of nodes as you can get reasonable times for. Plot the size of the independent set found as a function of the graph size. What do you conjecture about how the average size of the independent set grows as a function of \( n = |V| \)?

When describing an algorithm, don’t write out an entire pseudo-code; just describe it at a high level. Be sure to specify completely all data structures used in the algorithm. Include correctness proofs and time analysis for all algorithms, except for the implementation problem.