1. (40 points) We consider the following scheduling problem:

**INPUT:** A collection of jobs \( J_1, \ldots, J_n \), where the \( i \)th job is a tuple \((r_i, x_i)\) of non-negative integers specifying the release time and size of the job.

**OUTPUT:** A preemptive feasible schedule for these jobs on one processor that minimizes the total completion time \( \sum_{i=1}^{n} C_i \).

A schedule specifies for each unit time interval, the unique job that that is run during that time interval. In a feasible schedule, every job \( J_i \) has to be run for exactly \( x_i \) time units after time \( r_i \). The completion time \( C_i \) for job \( J_i \) is the earliest time when \( J_i \) has been run for \( x_i \) time units. Examples of these basic definitions can be found below.

We consider two greedy algorithms for solving this problem that schedule times in an online fashion, that is the algorithms are of the following form:

\[
t = 0
\]

while there are jobs left not completely scheduled

Among those jobs \( J_i \) such that \( r_i \leq t \), and that have previously been scheduled for less than \( x_i \) time units, pick a job \( J_m \) to schedule at time \( t \) according to some rule;

increment \( t \)

One can get different greedy algorithms depending on the rule for selecting \( J_m \). For each of the following greedy algorithms, prove or disprove that the algorithm is correct. Proofs of correctness must use an exchange argument.

**SJF:** Pick \( J_m \) to be the job with minimal size \( x_i \). Ties may be broken arbitrarily.

**SRPT:** Let \( y_{i,t} \) be the total time that job \( J_i \) has been run before time \( t \). Pick \( J_m \) to be a job that has minimal remaining processing time, that it, that has minimal \( x_i - y_{i,t} \). Ties may be broken arbitrarily.

As an example of SJF and SRPT consider the following instance: \( J_1 = (0, 100) \), \( J_2 = (10, 10) \) and \( J_3 = (1, 4) \). Both SJF and SRPT schedule job \( J_1 \) between time 0 and time 1, and job \( J_3 \) between time 1 and time 5, when job \( J_3 \) completes, and job \( J_1 \) again between time 5 and time 10. At time 10, SJF schedules job \( J_2 \) because its original size 10 is less than job \( J_1 \)’s original size 100. At time 10, SRPT schedules job \( J_2 \) because its remaining processing time 10 is less than job \( J_1 \)’s remaining processing time 94. Both SJF and SRPT schedule job \( J_2 \) between time 10 and 20, when \( J_2 \) completes, and then job \( J_1 \) from time 20 until time 114, which job \( J_1 \) completes. Thus for both SJF and SRPT on this instance \( C_1 = 114 \), \( C_2 = 20 \) and \( C_3 = 5 \) and thus both SJF and SRPT have total completion time 139.
2. (20 points) Consider the problem of computing an optimal binary search tree. The input to this problem is \( n \) keys \( K_1, \ldots, K_n \) where \( K_i < K_{i+1} \), and associated probabilities \( p_1, \ldots, p_n \) for accessing these keys. So \( \sum_{i=1}^{n} p_i = 1 \). The output to this problem should minimize possible expected depth of a key in a binary search tree \( T \) on these keys, that is the output should be the minimum possible value of

\[
\sum_{i=1}^{n} p_i \text{ (depth of key } K_i \text{ in } T) \]

More generally let \( A(a,b) \) be the minimum possible value of

\[
\sum_{i=a}^{b} p_i \text{ (depth of key } K_i \text{ in } T) \]

over all binary search trees \( T \) on the keys \( K_a, \ldots, K_b \).

(a) Give a recursive formula for \( A(a,b) \).

(b) Prove that naively using this recursive formula to compute \( A(1,n) \) will require time at least \( 2^n \).

(c) Give iterative array-based polynomial-time pseudo-code to compute a table \( A \) storing all the possible values of \( A(a,b) \).

(d) Show the table for the inputs \( p_1 = \frac{1}{2}, p_2 = \frac{1}{3} \) and \( p_3 = \frac{1}{6} \).

3. (20 points) Consider the problem where the input is a collection of \( n \) trips on the Pennsylvania turnpike. For the \( i \)th trip \( T_i \) you are given the date \( d_i \) of that trip, and the non-discounted toll \( f_i \) for that trip. The turnpike sells an Easypass for \( E \) dollars that gives you a 25% reduction in tolls for the next \( L \) days after the pass is purchased (counting the day of purchase). The problem is to determine on what days to buy an Easypass so as to minimize total cost of your travel.

For example if the input was \( L = 4, E = 25, d_1 = 1 \) and \( f_1 = $20, d_2 = 3 \) and \( f_2 = $40, d_3 = 5 \) and \( f_3 = $30, d_4 = 8 \) and \( f_4 = $60, d_5 = 10 \) and \( f_4 = $160 \), then buying an Easypass on days 1 and 8 would result in a cost of $50 for two Easypass purchases, a fare cost of $15 for day 1, a fare cost of $30 for day 3, a fare cost of $30 for day 5, a fare cost of $45 for day 8, and a fare cost of $120 for day 10. This would give a total cost of $290. There is no claim that this is the optimal solution.

Give a dynamic programming algorithm for this problem whose running time is polynomial in \( n \). The running time of you algorithm should not depend on \( L \) or \( E \). Initially you can give an algorithm that just computes the minimum cost (that is you don’t need to compute the days on which to buy the Easypass). Don’t forget to say where to look in your table for the answer. Then just briefly sketch how you can use the table to reconstruct the actual days to purchase the Easypass.
4. (20 points) Consider the following problem. You have \( n \) containers. Each container \( C_i \) has an associated positive integer weight \( w_i \), positive integer content value \( v_i \), and an associated shape. There are three possible shapes: sphere, cylinder and cube. You want to load some of these containers onto a ship. You have a weight limit \( W \) on the aggregate weight of the containers that you can load onto the ship. Furthermore, you know that the ship has space for at most \( P \) spheres, and at most \( Y \) cylinders and at most \( U \) cubes. Note that these shape constraints are independent, so you can have \( P \) spheres and \( Y \) cylinders and \( U \) cubes. The weights, values, weight limit, \( P, Y \) and \( U \) are all part of the input. The goal in the problem is to find the most profitable containers to load on the ship subject to the above constraints. The profit is just the sum of the values of the loaded containers. Give a dynamic programming algorithm for this problem with running time that is bounded by a polynomial in \( n + W \).

Initially you can give an algorithm that just computes a table containing the maximum profit as one entry (that is you don’t have to compute the actual containers). Don’t forget to say where to look in your table for the answer. Then just briefly sketch how you can use the table to reconstruct the actual containers.