1 Problem

Given $x_1, x_2, \ldots, x_n$ TB of available data for the next n days and given the amount of data a server can process $s_1, s_2, \ldots, s_n$ for n days after a fresh reboot (in TB).

Goal Choose the days on which you are going to reboot so as to maximize the total amount of data you process.

2 Dynamic Programming Algorithm

2.1 Main Idea (20 pts)

Breaking Problem Into Sub-problems For days 0 to n, choose to restart on day d such that $max(f(d))$ where $f(d) = P(0, d-1) + P(d+1, n)$. Find the amount of data proceesed $P(i, j)$ for a range of days $[i, j]$ provided you start with a fresh server (s_1) on day i. Now for the left partition (days 0 to $d-1$), repeat the same process of choosing the ideal day to reboot. This process will end when the length of X, the number of days of data left to process is 1 or 0 (when rebooting will not increase the amount of data processed).

Calculating $P(i, j)$, the Amount of Data Processed without rebooting from days i to j On each day, decide whether the amount of data the server processes is limited by the amount of available data or the processing capability of the server. Return the sum of the limiting factors (available data or power) across days i through j as $P(i, j)$.

Don't Repeat Calculations, The Essence of Dynamic Programming As we process the values of $P(i, j)$ for various $[i, j]$, we will populate a results matrix (two-dimensional array) to avoid repetitive calulations of both $P(i, j)$ and $P(a, b)$ where $i = a$ and $j = b$. This matrix will need to be of size $n + 1$ rows by n columns and will be initialized with a row of zero values which represent the amount of data processed on day d if we reboot on day d.

2.2 Pseudocode (10 pts)

```
# X represents the sequence of x_i values indexed from 1.
# S represents the sequence of s_j values indexed from 1.
def main():
    # read input and populate X and S
    row_init()  # initialize zeroth row of P (results matrix) with 0s
    GetMaximumProcessed(X, S)

def GetMaximumProcessed(X, S):
    for i in range(length of X):
        for j in range(length of S):
            # ensure do not calculate lower diagonals (impossible)
            P[i][j] = Min(X[i], S[j])
```

```
            if P[j - 1][i - 1] is a valid cell,
                then P[i][j] += P[j - 1][i - 1]
            if there are columns remaining in the left portion of the table P,
                then P[i][j] += the maximum of the i - j - 1 column.
    return max value of last column in P
```

2.3 Traceback Algorithm (10 pts)

Find the goal cell (maximum value in the results table P). From the right-most column in P, get the row index of the max value in that column. Using the indices of the max value, the day that will have caused that reboot is the index of the column - index of the row. Add the number of the day that will have caused that reboot to a set tracking the days we will reboot the server. Repeat this process for the columns on the left of the column of the last reboot until we do not have any more days on which to try and reboot.

SHOULD WE ADD PSEUDOCODE FOR TRACEBACK?

2.4 Time Complexity (10 pts)

$O(n^2)$

3 Implementation

3.1 Code (15 pts)

3.2 Small Example (10 pts)