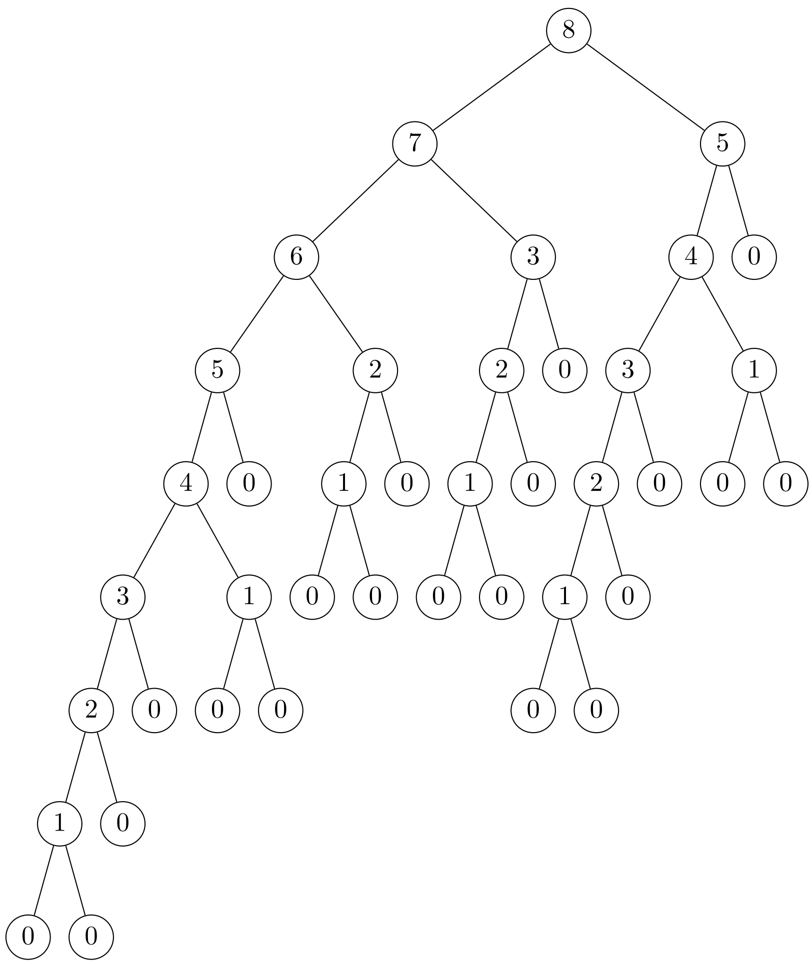
Problem 1

Job Number	$\mathbf{s_i}$	$\mathbf{f_i}$	$\mathbf{w_i}$	$\mathbf{p(i)}$
1	1	4	5	0
2	3	5	1	0
3	0	6	8	0
4	4	7	4	1
5	3	8	6	0
6	5	9	3	2
7	6	10	2	3
8	8	11	4	5

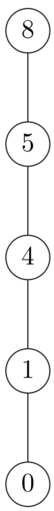
(a)



(b)

(c)

i	Values in M
1	[0,5]
2	[0,5,5]
3	[0,5,5,8]
4	[0,5,5,8,9]
5	[0,5,5,8,9,9]
6	[0,5,5,8,9,9,9]
7	[0,5,5,8,9,9,9,10]
8	[0,5,5,8,9,9,9,10,13]

(d) 

Problem 2

(a) The dynamic programming state of this problem is $OPT(i, w, k)$ where i is the index of the item being considered, w is the remaining weight of the knapsack, and k is the number of items we can still select. The Bellman equation can be written as follows:

$$OPT(i, w, k) = \begin{cases} 0 & \text{if } i = 0 \text{ or } w = 0 \text{ or } k = 0, \\ OPT(i-1, w, k) & \text{if } w_i > w \\ max(OPT(i-1, w, k), v_i + OPT(i-1, w - w_i, k-1)) & \text{O.W.} \end{cases}$$

(b) The pseudocode can be written as below:

```
foreach i = 0 to n do
    foreach w = 0 to W do
        foreach k = 0 to K do
            if i = 0 or w = 0 or k = 0 then
                dp[i][w][k] ← 0
                visited[i][w][k] ← True
            else
                visited[i][w][k] ← False
            end if
```

        end foreach
      end foreach
    end foreach
    Function Top-down $(i, w, k, weights, values, dp, visited)$
    if $visited[i][w][k]$ = True **then**
      return $dp[i][w][k]$
    end if
    if $weights[i] > w$ **then**
      $dp[i][w][k] \leftarrow$ Top-down $(i-1, w, k, weights, values, dp, visited)$
    else
      $exclude \leftarrow$ Top-down $(i-1, w, k, weights, values, dp, visited)$
      $include \leftarrow values[i]$+Top-down $(i-1, w-weights[i], k-1, weights, values, dp, visited)$
      $dp[i][w][k] \leftarrow max(exclude, include)$
    end if
    $visited[i][w][k] \leftarrow$ True
    return $dp[i][w][k]$

Problem 3

Suppose the set of chocolates is $M = 1, 2, ..., n$ where each i is of type $t \in$ {1, 2, ..., K}

Suppose that M_a and M_b are the set of chocolates for Alice and Bob respectively. We have $S_a = \sum_{i \in M_a} t_i$, $S_b = \sum_{i \in M_b} t_i$

(a) The goal of the problem is to optimize the difference between the two sets:

min $|S_a - S_b|$

such that $S_a + S_b = \sum_{i=1}^{n} t_i$

and $M_a \cup M_b = M$,

$M_a \cap M_b = \emptyset$

(b) $S_{min} = min(S_a, S_b), S_{max} = max(S_a, S_b)$

Therefore, $S_{min} \leq S_{max}$.

Using $S_{min} + S_{max} = \sum t_i$, we have

$S_{min} + S_{max} \leq S_{min} + S_{max} = \sum t_i =>$

$S_{min} \leq \frac{1}{2} \sum_{i=1}^{n} t_i$

(c) This is an alternative to the knapsack problem:

- We have a bag with capacity $\frac{1}{2} \sum_{i=1}^{n} t_i$.
- We have item i, which has weight t_i and value t_i. Notice that weights and values are the same.

Step 1: $OPT(n, W)$ = maximum value, using the first n items with capacity equal to W.

Step 2:

$$OPT(n, W) = \begin{cases} 0 & \text{if } n = 0 \\ OPT(n-1, W) & \text{if } n \geq 1 \text{ and } W < t_n \\ \max\left(OPT(n-1, W-t_n) + t_n, OPT(n-1, W)\right) & \text{if } n \geq 1 \text{ and } W \geq t_n \end{cases}$$

(d) The time and space complexity of the algorithm is O(n$\sum_{i=1}^{n} t_i$), as it is in the knapsack problem.

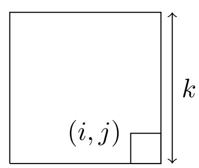
Problem 4

(a) Let A denote the fertility matrix such that $A[i,j]$ shows the fertility of land (i,j). Let $S_1 = A[:,1]$(first column of A) and $S_j = S_{j-1} + A[:,j]$:

foreach $j = 1$ to n **do**
  foreach $j' = j$ to n **do**
    $x \leftarrow S_j - S_{j'}$
    Run Kadane's algorithm to find the max sum subarray
    Update the coordinates of the maximum rectangle seen so far
  end foreach
end foreach

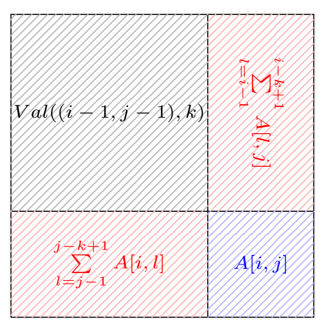
(b) The time complexity of the algorithm is $O(n^3)$ as the time it takes for the body of the double-loop to complete is $O(n)$. The space complexity is $O(n^2)$ as we only need to store the values of S_j for $j \in \{1, 2, ..., n\}$.

(c) Now that we have picked a wheat farm, we can change the fertility of the covered segments to $-\infty$, so that our algorithm won't pick those segments for the carrot farm. Notice that a square is identified by its corner coordinates and its size:



Assuming $Val((i,j),k)$ is the sum of the values in such a square, we have:

$$Val((i,j),k+1) = A[i,j] + \sum_{l=j-1}^{j-k+1} A[i,l] + \sum_{l=i-1}^{i-k+1} A[l,j] + Val((i-1,j-1),k)$$



Now let $R_{i,j} = \sum_{l=1}^{j} A[i,l]$ denote a row sum, and $C_{i,j} = \sum_{l=1}^{i} A[l,j]$ denote a column sum. Hence we have:

$$Val((i-1,j-1),k) = \begin{cases} A[i,j] & k = 1 \\ A[i,j] + R_{i,j-1} - R_{i,j-k+1} + V_{i-1,j} - V_{i-k+1,j} + Val((i-1,j-1),k-1) & k \geq 1 \\ -\infty & i < 0 \text{ or } j < 0 \end{cases}$$

Problem 4 continued on next page...

(d) Notice that calculating $R_{i,j}$ and $C_{i,j}$ takes $O(n^2)$ time and space:

$$R_{i,j} = \begin{cases} A[i,1] & j = 1 \\ R_{i,j-1} + A[i,j] & \end{cases}, C_{i,j} = \begin{cases} A[1,j] & i = 1 \\ C_{i-1,j} + A[i,j] & \end{cases}$$

and calculating $Val((i,j),k)$ takes $O(n^3)$ time and $O(n^2)$ space. Notice that we can reuse the space to ensure $O(n^2)$ space complexity, while carrying the best square in $O(1)$ variable.
Important note: You can also use an approach similar to part (a) to solve the problem. The main difference is that you do not need Kadane's algorithm, as given j and j', there are only $O(|j' - j|)$ possible squares.