This document gives model solutions to the assignment problems. Note that alternative solutions may exist.

Question 1 *Bridge Support*

[20 marks] The NSW government has decided to build a new bridge. The bridge is to be n meters long, where $A[i]$ is the maximum load on the bridge between $(i-1)$ meters and i meters.

The government have contracted you to complete this project, and you've decided to build the bridge in spans of integer length. The cost of building a span is given by the length of the span squared, plus the maximum load that the span needs to withstand. The total cost of building the bridge is the sum of the cost of all spans. More explicitly:

$$\texttt{total\_cost} = \sum_{s \in \text{spans}} (s_R - s_L)^2 + \max(A[(s_L+1)..s_R])$$

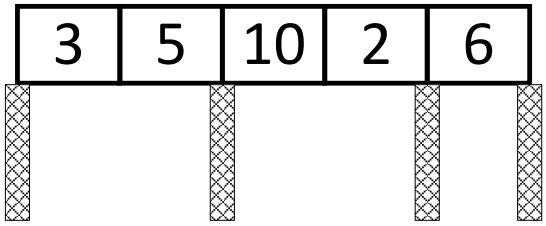
where s_L and s_R are the left and right endpoints respectively of the span s.

Your goal is to determine the **minimum** cost required to build a bridge that supports the load.

An example for the cost calculation: if $n = 5$ and $A = [3, 5, 10, 2, 6]$, and you decided to put pillars in at 2 meters and 4 meters, the total cost would be:

`span_[0,2]` $= (2-0)^2 + \max(A[1..2]) = \$9$
`span_[2,4]` $= (4-2)^2 + \max(A[3..4]) = \$14$
`span_[4,5]` $= (5-4)^2 + \max(A[5..5]) = \$7$
`total_cost` $= \$9 + \$14 + \$7 = \$30$

A depiction of this example is shown in image below.



1.1 **[14 marks]** Design an $O(n^3)$ algorithm which achieves your goal.

Subproblems: For $0 \le i \le n$, let $P(i)$ be the minimum cost of the bridge with span $[0, i]$.

Recurrence: To find the minimum cost of span $[0, i]$, we consider each possibility of $0 \le k < i$, where k is the position of the pillar that is the closest to i. The minimum cost of building the bridge of span $[0, k]$ is $P(k)$ and the extra cost to extend the bridge of span $[0, k]$ to span $0, i]$ is $(i-k)^2 + \max(A[(k+1)..i])$.

The minimum cost of span $[0, i]$ can be found by considering every possibility of k, which is given by

$$P(i) = \min_{0 \le k < i} (P(k) + (i-k)^2 + \max(A[(k+1)..i]))$$

Base cases: The minimum cost of building a bridge of length 0 is 0 and therefore $P(0) = 0$.

Overall answer: The final answer is given by $P(n)$.

Order of computation: Completing subproblem $P(i)$ requires subproblem $P(i-1)$. This is satisfied by computing them in increasing order of i.

Time complexity: There are precisely n subproblems, each subproblem takes $O(n^2)$ computation. This is because we iterate over n different position of k to find the minimum and for each position of k, we go through $A[(k+1)..i]$ to find the maximum value, which takes $O(n)$. Therefore the overall time complexity $O(n^3)$.

1.2 [6 marks] With an election looming, the NSW government has asked you to hurry up. Design an algorithm that achieves the goal in $O(n^2)$.

You may choose to skip Question 1.1, in which case your solution to Question 1.2 will also be submitted for Question 1.1.

To solve it in $O(n^2)$, we preprocess A and create an array $T[1..n][1..n]$ where $T[i][j]$ is the maximum value of $A[i..j]$. To do this, for each $1 \leq i \leq n$, we first set $T[i][i] = A[i]$, then we iterate from $i \leq j \leq n$ and for each ending j we update $T[i][j]$ if the element on position j is larger than $\max(A[i..j-1])$, that is $T[i][j] = \max(T[i][j-1], A[j])$. Therefore, the recurrence from 1.1 becomes

$$P(i) = \min_{0 \leq k < i} (P(k) + (i-k)^2 + T[k+1][i]))$$

Time complexity: Similar to 2.1, there are precisely n problems, while now each subproblem only takes $O(n)$ because we iterate through n different positions of k and the lookup operation in each iteration takes $O(1)$. Therefore, the overall time complexity is $O(n^2)$.

Question 2 *Mountain Radios*

[30 marks] Alice and Bob are bored of the park, and have decided to go mountaineering! They will each be traversing the mountain and moving through a sequence of n camps. Both Alice and Bob will always spend the night at one of their camps. Alice's camps are given by a sequence of coordinates pairs $A_1, ..., A_n$, where $A_i = (A_{i_x}, A_{i_y})$, and Bob's camps are given as $B_1, ..., B_n$, where $B_j = (B_{j_x}, B_{j_y})$. On the first night, Alice will spend the night at camp A_1 and Bob will spend the night at camp B_1. Each day, both Alice and Bob have the choice of staying at their current camp or moving to the next camp in their sequence. They can never move back to a camp they have previously visited. At least one of them must move each day, meaning that both Alice and Bob will have reached their final camps no later than the $(2n-1)$th night.

Alice and Bob would like to be able to talk to each other at night via a radio, but all of the wireless radios available for purchase have a fixed range. Alice has noticed that if they carefully plan who moves to the next camp on each day, it will affect how far apart they get during their journey. The distance between two coordinates (x_1, y_1) and (x_2, y_2) is calculated as $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

For example, let $n = 5$ and $A_1 = (1, 1)$, $A_2 = (3, 1)$, $A_3 = (4, 1)$, $A_4 = (5, 1)$, $A_5 = (6, 1)$, $B_1 = (1, 2)$, $B_2 = (2, 2)$, $B_3 = (3, 4)$, $B_4 = (4, 2)$, $B_5 = (6, 2)$. In this example, the furthest apart Alice and Bob **must** get during their journey is 3 units. One way this could be achieved is with the sequence of stays $[(A_1, B_1), (A_2, B_2), (A_2, B_3), (A_2, B_4), (A_3, B_4), (A_4, B_5), (A_5, B_5)]$. With this sequence of moves, the furthest apart that Alice and Bob get is on night 3, when Alice is at A_2 and Bob is at B_3.

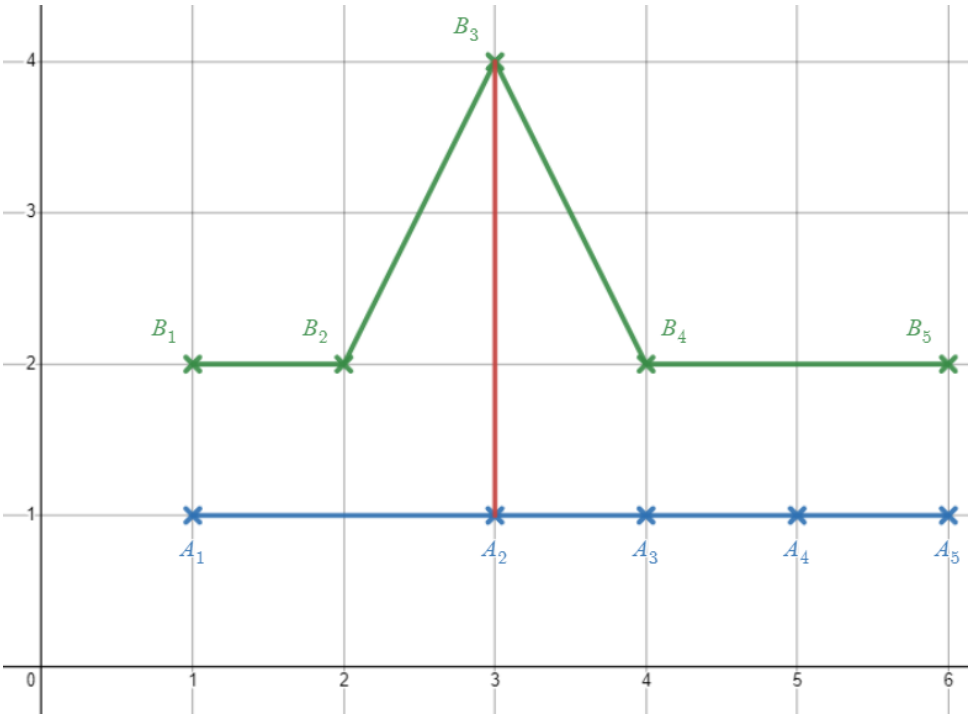


Figure 1: The simple example described above. The blue line represents the path Alice will walk and the green line represents the path Bob will walk. The red line shows the furthest apart Alice and Bob **must** get during their journey.

2.1 **[15 marks]** Design an $O(n^2)$ algorithm to determine whether a radio with a range of D will allow them to communicate every night.

In the provided example, your method should determine that it is impossible for any value of $D < 3$, and possible otherwise.

We want to identify if there exists a path through the camps that will allow Alice and Bob to stay within a range of D the entire time. The optimal substructure arises from an observation: for Alice and Bob to finish their journey at A_n and B_n without exceeding a distance of D, we must have $d(A_n, B_n) < D$, and they must have been able to reach either camps (A_n, B_{n-1}), (A_{n-1}, B_n), or (A_{n-1}, B_{n-1}) without exceeding this range. This motivates the following solution.

Subproblems: For $0 \leq i, j \leq n$, let $P(i, j)$ be true if it is possible for Alice and Bob to travel through camps $A_1, ..., A_i$ and $B_1, ..., B_j$ respectively without ever exceeding a distance of D, and finishing their journey at camps A_i and B_j.

Recurrence: The recurrence arises from the observation made above. This is summarised as follows:

$$P(i, j) = (P(i-1, j-1) \textbf{ or } P(i-1, j) \textbf{ or } P(i, j-1)) \textbf{ and } (d(A_i, B_j) < D)$$

Base cases: $P(1, 1)$ is true as long as $d(A_1, B_1) < D$. We also define $P(i, j)$ to be false whenever $i = 0$ or $j = 0$, since Alice and Bob must always start at A_1 and B_1.

Alternatively, you can separate out the cases at $i = 1$ and $j = 1$ as follows.

$$P(i, 1) = P(i-1, 1) \textbf{ and } (d(A_i, B_1) < D)$$

$$P(1, j) = P(1, j-1) \textbf{ and } (d(A_1, B_j) < D)$$

Overall answer: The final answer is given by $P(n, n)$.

Order of computation: Completing subproblem (i, j) requires subproblems $(i-1, j-1)$,$(i-1, j)$ and $(i, j-1)$ to already be completed. This is satisfied by computing them in increasing order of j nested in increasing order of i.

Time complexity: There are precisely n^2 subproblems, and each subproblem takes $O(1)$ computation. This results in a total complexity of $O(n^2)$ as required by the question.

2.2 **[15 marks]** Design an $O(n^2)$ algorithm that calculates the minimum range required for Alice and Bob to be able to communicate every night.

In the provided example, your method should calculate a minimum required range of 3.

You can receive up to 7 marks for an $O(n^2 \log n)$ solution.

You may choose to skip Question 2.1, in which case your solution to Question 2.2 will also be submitted for Question 2.1.

We wish to choose a path through the camps that minimises the maximum distance between Alice and Bob at any given night. This is extremely similar to question 2.1, and will use a very similar subproblem structure by slightly tweaking our initial observation. If we assume that Alice and Bob are in camps i, j on a particular night, we can see that the minimum range required on their journey so far will be the maximum of their current distance apart,

and the minimum range required by any journey that could possibly have lead them to these camps.

Subproblems: For $0 \leq i, j \leq n$, let $P(i, j)$ be the minimum range radio required by Alice and Bob when Alice travels through camps $A_1, ..., A_i$ and Bob travels through camps $B_1, ..., B_j$, finishing their journey at camps A_i and B_j.

Recurrence: The recurrence arises from the observation made above. This is summarised as follows:

$$P(i,j) = \min \begin{cases} \max(d(A_i, B_j), P(i-1, j)) \\ \max(d(A_i, B_j), P(i, j-1)) \\ \max(d(A_i, B_j), P(i-1, j-1)) \end{cases}$$

Base cases: Trivially, we have $P(1, 1) = d(A_1, B_1)$. We also define $P(i, j) = \infty$ whenever $i = 0$ or $j = 0$, since Alice and Bob must always start at A_1 and B_1.

Alternatively, you can separate out the cases at $i = 1$ and $j = 1$ as follows:

$$P(i, 1) = \max(d(A_i, B_j), P(i-1, j))$$

$$P(1, j) = \max(d(A_i, B_j), P(i, j-1))$$

Overall answer: The final answer is given by $P(n, n)$.

Order of computation: Same as in question 2.1.

Time complexity: Same as in question 2.1.

Question 3 *KFC Now!*

[20 marks] There are $n \geq 0$ people queuing for KFC. The i^{th} person has an hunger of h_i, and everyone in line has a distinct hunger. A person's 'annoyance' is defined as the number of people in front of them who are less hungry, and the annoyance of the entire queue is the sum of every person's annoyance.

Given the hunger of every person, your goal is to determine how many arrangements of the queue will result in a total annoyance of exactly k.

As always, you can assume that all arithmetic operations $(+ - \times /)$ are done in constant time.

3.1 **[12 marks]** Design an $O(n^2k)$ algorithm that achieves the goal.

If $K < 0$ or $K > \binom{N}{2}$, there are no possible arrangements. Otherwise, we shall solve this problem using dynamic programming.

First note that h_i is irrelevant - a persons annoyance is dependant only on their hunger compared to other people and their position. So from now on we will consider arrangements of any N distinct numbers.

Subproblems: For $0 \leq n \leq N$ and $0 \leq k \leq \binom{n}{2}$, let $P(n, k)$ be the number of permutations of n distinct numbers, which have exactly k inversion pairs.

Recurrence: When $n \geq 1$, there is a unique most important person, lets call them Max. We will count the number of queues with annoyance k for each position of Max.

Consider the case where there are r people in front of Max:

Firstly, note that Max has an annoyance of r, because everyone in front of them are less hungry.

Secondly, note that all other people would have the same annoyance as the queue with Max removed, because the people in front of Max are unaffected and the people behind Max are indifferent to a hungry person in front of them.

From these 2 observations, it follows that we are simply counting arrangements with 1 less person and r less annoyance. Hence there are $P(n-1, k-r)$ possible queues with r people in front of Max.

The number of people in front of Max is between 0 and $n-1$ inclusive.

$$\begin{aligned} P(n,k) &= \sum_{r=0}^{n-1} P(n-1, k-r) \\ &= P(n-1,k) + P(n-1,k-1) + \cdots + P(n-1,k-(n-1)) \end{aligned}$$

Base cases: When $n = 0$, the only possibility is having 0 people in line with a total annoyance of 0. Hence,

$$P(0,k) = \begin{cases} 1 & k = 0 \\ 0 & k \neq 0 \end{cases}$$

Overall answer: $P(N, K)$

Order of computation: Each subproblem $P(n, k)$ depends on subproblems $P(n-1, k)$ through to $P(n-1, k-(n-1))$. We can solve subproblems in any order of k, then increasing order of n.

Time complexity: There are $O(NK)$ subproblems, each solved in $O(N)$ time. Therefore the overall time complexity is $O(N^2K)$

3.2 **[8 marks]** Design an $O(nk)$ algorithm which achieves the goal.

You may choose to skip Question 3.1, in which case your solution to Question 3.2 will also be submitted for Question 3.1.

Notice that our recurrence relation is a sum of consecutive dp results. Hence, we would like to do a prefix sum so that we can calculate this in constant time.

Subproblems: Let

$$F(n,k) = P(n,0) + P(n,1) + \cdots + P(n,k).$$

In other words, for $0 \leq n \leq N$ and $0 \leq k \leq \binom{n}{2}$, let $F(n,k)$ be the number of permutations of n numbers, which have **at most** k inversion pairs.

A helpful observation is that $P(n,k) = F(n,k) - F(n,k-1)$, because a permutation has k inversion pairs when it has at most k inversion pairs but not at most $k-1$ inversion pairs.

Recurrence: When $n \geq 1$,

$$P(n,k) = P(n-1,k) + P(n-1,k-1) + \cdots + P(n-1,k-(n-1))$$

$$P(n,k) = \sum_{r=k-(n-1)}^{k} P(n-1,r)$$

$$F(n,k) - F(n,k-1) = F(n-1,k) - F(n-1,k-n)$$

$$F(n,k) = F(n-1,k) - F(n-1,k-n) + F(n,k-1)$$

Base cases: When $n = 0$, the only possibility is having 0 people in line with a total annoyance of 0. Hence,

$$F(0,k) = \begin{cases} 1 & k \geq 0 \\ 0 & k < 0 \end{cases}$$

Overall answer: $P(N,K) = F(N,K) - F(N,K-1)$

Order of computation: Each subproblem $F(n,k)$ depends on subproblems $F(n-1,k)$ through to $F(n-1,k-(n-1))$ and $F(n,k-1)$. We can solve subproblems in increasing order of k, then increasing order of n.

Time complexity: There are $O(NK)$ subproblems, each solved in $O(1)$ time. Therefore the overall time complexity is $O(NK)$

Question 4 *Antoni the Merchant*

[30 marks] Antoni wants to earn as much money as possible over the summer holidays as a travelling merchant in the DynaProg continent. He will begin his journey with \$0 wealth, and he only has D days before term starts and he is pulled back into answering forum questions, so he needs to plan very carefully. After thorough research, he has found n countries where he can turn an easy profit; in fact, he knows that if he is in country $i \in [1..n]$ on day $d \in [1..D]$, he is guaranteed to earn $P[i][d] > 0$ dollars.

All n countries have airports, and a direct flight to every other country. Each day Antoni can either travel to another country, or stay where he is. There is no restriction on how many times he can visit a certain country, nor on the number of days he can stay per visit.

However, the tax laws in DynaProg are vicious - every day he is taxed a *percentage of his current wealth*. Specifically, you are given a Tax Table $T[1..n][1..n]$ where $0 \leq T[j][i] \leq 100$ is the percentage Antoni will be taxed if he is currently in country j and was in country i the previous day. He is not taxed anything on the first day as he has \$0 wealth.

Help Antoni prepare a full itinerary (which country to be in on every day) that maximises his wealth by the end of D days.

Tax is deducted before the earnings are added for the day.

On Day 1, Antoni will not be taxed since his wealth is \$0 and there is nothing to deduct. He will then earn $\$P[i][1]$ for whichever country i he has chosen to start at.

On Day 2, he can either stay in country i, or travel to another country j. If he chooses to stay, he will be taxed $T[i][i]\%$ of his current wealth, and then earn $\$P[i][2]$. If he chooses to travel, he will be taxed $T[j][i]\%$ of his current wealth, and then earn $\$P[j][2]$.

4.1 [12 marks, 4 marks per part] Antoni has prepared a few Greedy approaches to solving this problem, but is not convinced they will always earn him the most money.

Provide a counter example for each of the following approaches. Your counter example must state the values for n, D, P, T, the wealth Greedy obtains and its itinerary, and finally, a more optimal wealth and the itinerary used to attain it. The more optimal solution does not have to be the best, it just has to beat Greedy.

[A] On each day d, travel to the country where Antoni will earn the most money (i.e. travel to the country i such that $P[i][d]$ is maximised, for all days $1 \leq d \leq D$). If there is a tie, choose the country with the lower tax rate.

[B] First, travel to the country with the highest Day 1 profit (i.e. travel to the country i such that $P[i][1]$ is maximised). Then, on each subsequent day d, travel to the country that minimises the amount of tax (in dollars) that Antoni will be charged (i.e. travel to the country j such that $T[j][i]\% \times$ `current_net_wealth` is minimised, for all days $2 \leq d \leq D$). If there is a tie, choose the country with the higher profit.

[C] First, travel to the country with the highest Day 1 profit. Then, on each subsequent day d, travel to the country that maximises Antoni's wealth at the end of the day (i.e. travel to the country j such that `current_net_wealth` $\times (100 - T[j][i])\% + P[j][d]$ is maximised, for all days $2 \leq d \leq D$). Break ties arbitrarily.

[A] Suppose we have two countries and two days to plan (i.e. $n = 2$ and $D = 2$). On country 1, $P[1][d] = 2$ for each $1 \leq d \leq 2$ and $T[j][1] = 0$ for each $1 \leq j \leq 2$. On country 2, $P[2][d] = 3$ for each $1 \leq d \leq 2$, $T[1][2] = 0$ and $T[2][2] = 100$.

Since the greedy method maximises $P[i][j]$, the greedy method will choose country 2

for each day. The wealth can be obtained as follows.

On day 1, Antoni earns $P[2][1] = \$3$ by travelling to country 2. On day 2, Antoni stays in country 2. Since staying at country 2 has a tax rate of 100%, the amount that is earned is $(100 - 100)\% \times P[2][1] + P[2][2] = \$3$.

However, a more optimal solution is for Antoni to stay in country 1 for the two days. On day 1, Antoni earns $P[1][1] = \$2$ by travelling to country 1. On day 2, Antoni stays in country 1 which does not affect the overall wealth since the tax rate is 0% each day. Therefore, the wealth after the second day is $(100-0)\% \times P[1][1] + P[1][2] = \$2 + \$2 = \$4$.

[B] Suppose we have two countries and two days to plan (i.e. $n = 2$ and $D = 2$). On country 1, $P[1][d] = 1$ for each $1 \leq d \leq 2$, and $T[1][1] = 0$ and $T[2][1] = 100$. On country 2, $P[2][1] = 0$ while $P[2][2] = 3$, and $T[j][2] = 100$ for each $1 \leq j \leq 2$.

Note that the proposed greedy method will stay in country 1 on both days since country 1 has the highest profit on day 1 and the lowest tax rate on day 2. The wealth can be obtained as follows.

On day 1, Antoni earns $P[1][1] = \$1$ by travelling to country 1. On day 2, Antoni stays in country 1 since $T[1][1] < T[2][1]$. Since the tax rate is 0, the amount that is earned is $(100 - 0)\% \times P[2][1] + P[2][2] = \$1 + \$1 = \$2$.

However, a more optimal solution is for Antoni to stay in country 2 for the two days. On day 1, Antoni earns $P[2][1] = \$3$. On day 2, Antoni stays in country 2. In doing so, the tax rate is 100%, meaning that Antoni obtains a total of $P[2][2] = \$3$, which exceeds the solution given by the proposed greedy algorithm.

[C] Suppose we have two countries and two days to plan (i.e. $n = 2$ and $D = 2$). On country 1, $P[1][1] = 100$, $P[1][2] = 1$, and $T[j][1] = 100$ for each $1 \leq j \leq 2$. On country 2, $P[2][1] = 1$, $P[2][2] = 50$, and $T[j][2] = 0$ for each $1 \leq j \leq 2$. Under this instance, the wealth can be obtained as follows.

On day 1, Antoni earns $P[1][1] = \$100$ by travelling to country 1. On day 2, Antoni will have to either travel to country 1 or 2. Since both countries have a tax rate of 100%, the greedy method will choose the country that maximises $P[j][2]$, which is country 2. This gives a total wealth of $P[2][2] = \$50$.

However, a more optimal solution is for Antoni to stay in country 2 for both days. On day 1, Antoni earns $P[2][1] = \$1$. Since the tax rate is 0, the amount earned on day 2 by staying in country 2 is $(100 - 0)\% \times P[2][1] + P[2][2] = \$1 + \$50 = \$51$.

4.2 **[18 marks]** Design an $O(n^2D)$ algorithm that finds the maximum possible wealth Antoni can attain, and also provides the itinerary he needs to follow.

Subproblem $P(i, d)$: Let $\text{opt}(i, d)$ denote the *maximum wealth* Antoni can have after selling his goods in country i on day d. Let $\text{prev}(i, d)$ be the country that Antoni must have visited on day $d - 1$ in order to maximise $\text{opt}(i, d)$.

Base cases: Note that Antoni can start at any of the n countries. Therefore, for each $1 \leq i \leq n$, $\text{opt}(i, 1) = P[i][1]$ and $\text{prev}(i, 1) = -1$.

Recurrence: To understand how to arrive at the recurrence, we can consider all of the ways Antoni can arrive at country i on day d. Since Antoni can travel from *any* of the n countries on a single day, he could have spent the previous day $d - 1$ in any of the n countries. For each particular country j on day $d - 1$, $T[j][i]$ is the amount being taxed when travelling from country j to country i; therefore, for each country j the most money he can have after

being taxed on arrival in country i is $\text{opt}(j, d-1) \times (100 - T[i][j])\%$. Whichever country j maximises this quantity is the best place for Antoni to have travelled from.

Therefore, the recurrence becomes

$$\text{opt}(i, d) = P[i][d] + \max_{1 \le j \le n} \{\text{opt}(j, d-1) \times (100 - T[i][j])\%\}.$$

$$\text{prev}(i, d) = \arg\max_{1 \le j \le n} \{\text{opt}(j, d-1) \times (100 - T[i][j])\%\}.$$

Order of computation: Each subproblem $\text{opt}(i, d)$ only relies on subproblems from earlier days. Therefore, we solve these subproblems in ascending order of d, then in any order of i.

Final solution: Since we can end at any of the n countries, the final solution is obtained by maximising the quantity on day D. This gives us the final solution being $\max_{1 \le i \le n} \{\text{opt}(i, D)\}$.

To obtain the itinerary itself, we can backtrack through the predecessors that we stored through the recurrence. Let $\text{country}(d)$ denote the country in the itinerary that Antoni visited on day d. For the last day D, we have

$$\text{country}(D) = \arg\max_{1 \le i \le n} \{\text{opt}(i, D)\}.$$

For each $1 \le d \le D-1$, we have

$$\text{country}(d) = \text{prev}(\text{country}(d+1), d+1).$$

The final itinerary is given by

$$\{\text{country}(1), \text{country}(2), \ldots, \text{country}(D)\}.$$

Time complexity: To argue the time complexity, we observe that, for each country i, there are D subproblems (one subproblem for each day). Therefore, there are nD many subproblems. In each subproblem, we need to consider all n countries. Therefore, each subproblem takes $O(n)$ time to compute, giving us a total time complexity of $O(n^2 D)$.