Financial Report (Solution)

In this review, we explain a solution for each of 6 Subtasks. We first explain how to solve Subtask 1 by a brute-force search algorithm whose time complexity is $O(2^N \times N)$. Then we explain a solution to Subtask 2 by Dynamic Programming whose time complexity is $O(N^3)$. An improvement of it gives a solution to Subtask 3 whose time complexity is $O(N^2)$

Then we explain faster algorithm which is valid in some special circumstances such as Subtask 4 ($D = 1$) and Subtask 5 ($D = N$). Finally, we explain full score solution whose time complexity is $O(N \log N)$. It is an improvement of the solution of Subtask 3 by data structures. for certain how to solve the task faster for

If you hurry to read a full score solution, please see "Full Score Solution" section in p. 5.

Subtask 1 ($N \leq 20$)

First, we consider a brute-force search algorithm to choose the data for the presentation. There are two choices (Bitaro chooses it or does not choose it) for the data for each day. In total, there are $2 \times 2 \times \cdots \times 2 = 2^N$ choices to choose the data . The solution of this task is the maximum of the impression scores for these choices.

How can we list all of the 2^N possible choices by a brute-force algorithm? There are several methods of implementation.

1. Search all of the 2^N possible choices by a nested loop. (It is an unusual method because the implementation is hard.)
2. Use a recursive function.
3. Use a **brute-force bit search** using binary digit numbers.

Here we explain the third method because the implementation is easy. The idea of brute-force bit search is to associate an integer between 0 and $2^N - 1$, inclusive, to each of the 2^N choices. Precisely, we associate it in the following way.

> Consider the binary digits of an integer i ($0 \leq i \leq 2^N - 1$). If the digit in the 2^j-th place is 1, Bitaro chooses the j-th data. If the digit in the 2^j-th place is 0, Bitaro does not choose the j-th data.

We can accomplish a brute-force search of 2^N possible choices by writing a for loop from $i = 0$ to $2^N - 1$. For each i, we can check whether it satisfies the condition or not and calculate the impression scores in $O(N)$ time. The total time complexity is $O(2^N \times N)$. We can calculate the answer under the constraints of Subtask 1. Thus we can get 15 points. The time complexity will be the same if we use other implementation methods such as recursive functions.

Subtask 2 ($N \leq 400$)

A brute-force search algorithm is not efficient because if N is increased by 1, then the time complexity is multiplied by 2. Obviously it cannot give an answer when $N = 400$. We shall use Dynamic Programming (DP) to improve the algorithm. In the following, we consider an algorithm to choose the data in chronological order and update the values in the DP table sequentially

> $dp[pos][curmax]$: the maximum number of times of making record-high sales amounts under the assumption that the final chosen data is the data of the pos-th day and the maximum sales amount at that day is $curmax$ yen.

Let $(pos, curmax)$ be the state that $dp[pos][curmax]$ indicates. If Bitaro chooses the data of the nxt-th day when the state is $(pos, curmax)$, the state becomes $(nxt, \max(curmax, nxt))$. Thus we can update the values in the DP table in the following way.

1. In the beginning, all the values of $dp[pos][curmax]$ are the initial value $-\infty$.
2. Do the operations 2-3 from smaller values of $pos, curmax$.
3. If $curmax = A_{pos}$, the value of $dp[pos][curmax]$ becomes $\max(dp[pos][curmax], 1)$. It means the first data chosen by Bitaro is the data of the pos-th day.
4. For every nxt $(pos + 1 \leq nxt \leq N)$, we update the value of $dp[pos][\max(curmax, A_{nxt})]$ to be $\max(dp[pos][\max(curmax, A_{nxt})], dp[pos][curmax] + 1)$.
5. The answer is the maximum of the final values of $dp[pos][curmax]$.

However, we cannot calculate these values efficiently because the value of $curmax$ can be as large as 10^9. Here we shall use the **coordinate compression** technique because the answer depends only on the relative magnitude relationship between A_i. We may consider all values are compressed into the range $0 \leq A_i \leq N - 1$. Then, we have $0 \leq curmax \leq N - 1$, and we can finite the calculation of the DP table in $O(N^3)$ time.

Subtask 3 ($N \leq 7000$)

We shall speed up the DP algorithm as above. We are now calculating the values of the DP table $dp[pos][curmax]$ sequentially. In order to speed up the calculation, we need to do one of the following:

- We shall speed up the calculation of each $dp[pos][curmax]$.
- We shall reduce the total number of possible states of DP. Currently, the number of possible values of $(pos, curmax)$ is $O(N^2)$.

Here we explain how to reduce the calculation in the second part. We shall perform DP calculation without considering *curmax*. Now, the reason why we need to consider *curmax* is to decide whether a record-high sales amount is achieved or not. In order to eliminate this consideration, we shall **change the state of DP only when a record-high sales amount is achieved**. Let us calculate the following DP table sequentially.

> $dp[pos]$: the maximum number of times of making record-high sales amounts under the assumption that the final chosen data is the data of the pos-th day and the maximum sales amount is achieved at that day.

We can calculate the valued in the DP table as follows: if it is possible to change the state from $dp[pre]$ to $dp[pos]$ (i.e., it is possible to achieve a maximum sales amount on the pos-th day after achieving a maximum sales amount on the pre-th day), then we update the value of $dp[pos]$ to be $\max(dp[pos], dp[pre] + 1)$. It is possible to change the state as $dp[pre] \to dp[pos]$ if and only if the following condition is satisfied.

> (Condition) $A_{pre} < A_{pos}$, and, for each of the $pre + 1, pre + 2, \cdots, pos - 1$-th day, we can choose a data whose sales amount is at most A_{pre} yen so that the difference of any two consecutive dates is at most D days.

In order to satisfy the above condition, the optimum strategy is to choose **all** of the data whose sales amount is at most A_{pre} yen. Hence we can rewrite this condition as follows, and can determine it easily.

> (Condition) $A_{pre} < A_{pos}$, and, for each of the $pre+1, pre+2, \cdots, pos-1$-th day, there are no consecutive D days when the sales amount exceeds A_{pre} yen.

It takes $O(N^3)$ time to obtain the answer if, when we calculate the value of $dp[pos]$, we individually determine whether it is possible to change the state as $dp[pre] \to dp[pos]$ for each pre. We can calculate whether it is possible to change the state for every pair (pre, pos) in advance. Then, we can obtain the answer in $O(N^2)$ time in total. The answer of this task is the maximum of $dp[1], dp[2], \ldots, dp[N]$.

Method of precalculation (Part 1)

There are several methods of precalculation. Here is a method. First, we note that the condition "for each of the $pre + 1, pre + 2, \cdots, pos - 1$-th day, there are no consecutive D days when the sales amount exceeds A_{pre} yen" becomes stronger if the value of pos increases. Therefore, the condition is satisfied for a range of the form $pre + 1 \leq pos \leq R_{pre}$.

For each i, the value of R_i is "the first position among $A_{i+1}, A_{i+2}, \ldots, A_N$ where $D + 1$ consecutive values exceed A_i." (If such a position does not exists, then $R_i = N$.) Thus, it can be calculated in $O(N)$ time for each i. After we calculate the values of $R_1, R_2, \ldots, R_N$, we can determine whether it is possible to change the states easily; the change of states is possible if and only if "$A_{pre} < A_{pos}$ and $pre + 1 \leq pos \leq R_{pre}$" are satisfied. We

can calculate it for every pair (pre, pos) in advance in $O(N^2)$ time.

The method of calculating $R_1, R_2, \ldots, R_N$ is related to the full score solution.

Subtask 4 ($D = 1$)

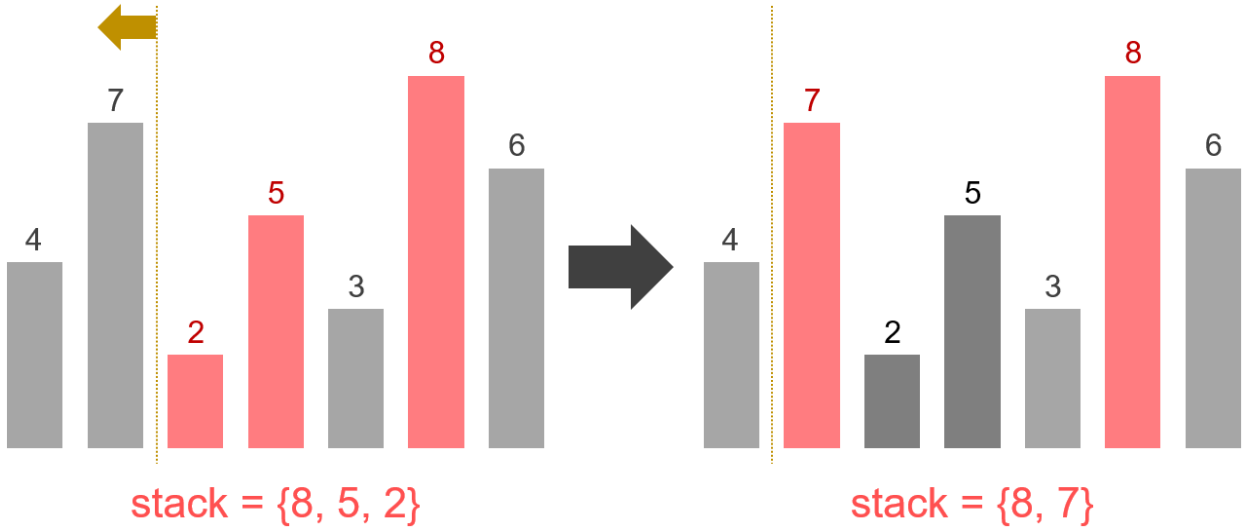
Let us recall the conditions. When $D = 1$, we need to choose the data so that the following two conditions are satisfied.

- Bitaro will show the latest sales amount.
- For any two consecutive data, the difference of the dates is at most 1 days. In other words, **Bitaro has to choose the data of consecutive days**.

Thus Bitaro has to choose the data of the $x, x + 1, \ldots, N$-th days ($1 \leq x \leq N$). There are N possible choices. But, if we calculate the impression score for each choice, the total time complexity becomes $O(N^2)$, and we cannot solve this Subtask.

In order to solve it, we shall calculate the answer for $x = N, N - 1, \cdots, 1$ in this order. We consider an algorithm using the data structure called "stack" to store the position where the maximum sales amount is achieved.

- Perform the operations 1-3 for $i = N, N - 1, \ldots, 1$, in this order.
  1. Delete (pop) the last element until the stack becomes empty or the last element of the stack becomes at least $A_i + 1$.
  2. Add A_i to the end of the stack.
  3. At that time, the size of the stack is the impression score when $x = i$.
- The answer is the maximum of the impression scores calculated above.



It may happen that it takes $O(N)$ time to perform an operation of type 2. But the total time complexity is $O(N)$ because the values at added to the stack N times and the number of times of deleting elements in in the operation 2 is at most N.

Subtask 5 ($D = N$)

When $D = N$, we need to choose the data so that the following two conditions are satisfied.

- Bitaro will show the latest sales amount.
- For any two consecutive data, the difference of the dates is at most N days.

We can ignore the second condition because it is always satisfied for any choice of the data for presentation. Thus we shall consider the first condition only. Moreover, the impression score will never increase if we delete the date of the last day. Also, the impression score will remain the same if we delete all of the data where the maximum sales amount is achieved. Therefore, we only need to consider the data where the maximum sales amount is achieved. Hence we need to calculate the following.

> The maximum number of data so that the sales amounts at the chosen days are increasing in chronological order.

This is a famous problem called "Longest Increasing Subsequence (LIS)." It is well-known that it can be solved in $O(N \log N)$ times. There are several algorithms to solve it. Here is one such algorithm.

1. In the beginning, we set the values of $s_1, s_2, \ldots, s_N$ to be ∞.
2. Perform the following operation for $i = 1, 2, \ldots, N$, in this order.

- Find the minimum pos satisfying $s_{pos} \geq A_i$ by a binary search technique, and update the value of s_{pos} to be A_i.

3. Finally, the number of k such that the value of s_k is different from ∞ is the length of the Longest Increasing Subsequence (LIS).

Note that the sequence $s_1, s_2, \ldots, s_N$ calculated by the above algorithm is not necessarily an example of Longest Increasing Subsequence (LIS).

Full Score Solution

Before, we calculate the DP table in chronological order (i.e., the order of the data $A_1, A_2, \ldots, A_N$). However, in the DP solution of Subtask 3, it is more convenient for us to calculate them in the reverse order (i.e., the order of the data $A_N, A_{N-1}, \ldots, A_1$). Here we shall consider calculating the following DP table in the opposite order.

> $dp[pos]$: the maximum number of times of making record-high sales amounts in the past (in chronological order) when we are choosing data opposite to the chronological order and the final chosen data is the data of the pos-th day.

Note that we change the DP states in the opposite order. As in Subtask 3, the change of states $dp[pre] \to dp[pos]$ is possible if and only if the following condition is satisfied.

> (Condition) $A_{pre} > A_{pos}$, and, for each of the $pos+1, pos+2, \cdots, pre-1$-th day, there are no consecutive D days when the sales amount exceeds A_{pre} yen.

As in the solution of Subtask 3, the value of R_i is "the first position among $A_{i+1}, A_{i+2}, \ldots, A_N$ where $D+1$ consecutive values exceed A_i." (If such a position does not exists, then $R_i = N$.) Then we can change the state to be $dp[pos]$ in the range $pos + 1 \leq pre \leq R_{pos}$ satisfying $A_{pre} > A_{pos}$. We can calculate the DP table by the following simple formula. (If we calculate the table by this formula, the total time complexity becomes $O(N^2)$.)

$$dp[pos] = \max\left(\max_{pos+1 \leq i \leq R_{pos}, A_i > A_{pos}} dp[i], 0\right) + 1$$

We need to speed up this algorithm using a data structure. In order not to consider the condition $A_i > A_{pos}$, we shall calculate the values of A_i in decreasing order. We have the following algorithm.

1. We set the values of $dp[1], dp[2], \ldots, dp[N]$ to be $-\infty$.
2. Perform the following operation in decreasing order of the value of A_{pos} (in increasing order of pos if the values are the same).
   - We set $dp[pos] = \max\left(dp[pos+1], dp[pos+2], \ldots, dp[R_{pos}], 0\right) + 1$.

3. Finally, the answer is the maximum of $dp[1], dp[2], \ldots, dp[N]$.

It usually takes $O(N)$ time to calculate one $dp[pos]$. Since calculating the minimum in a range is RMQ (Range Maximum Query), we can calculate it in $O(\log N)$ using a segment tree. Thus, once $R_1, R_2, \ldots, R_N$ are calculated, we can obtain the answer in $O(N \log N)$ time in total.

Method of precalculation (Part 2)

The remaining task is to calculate the values of $R_1, R_2, \ldots, R_N$. We can calculate it in $O(N^2)$ time by the algorithm explained in Subtask 3. We need to speed up it. Recall that R_i is "the first position among $A_{i+1}, A_{i+2}, \ldots, A_N$ where $D + 1$ consecutive values exceed A_i." In order words, it is the first value $j \geq i + D + 1$ where all of the values of $A_{j-D}, A_{j-D+1}, \cdots, A_j$ exceed A_i.

Here we put $M_j = \max(A_{j-D}, A_{j-D+1}, \ldots, A_j)$. Then R_i is the minimum of j in the range $j \geq i + D + 1$ such that $M_j \geq A_i + 1$ is satisfied. Hence we can calculate them by the following way.

1. For $j = D + 1, D + 2, \ldots, N$, calculate $M_j = \max(A_{j-D}, A_{j-D+1}, \ldots, A_j)$. For example, they can be calculated in $O(N \log N)$ time if we use a RMQ segment tree.
2. Then R_i is equal to the leftmost index among $M_{i+D+1}, M_{i+D+2}, \ldots, M_N$ where the value exceeds A_i. For example, it can be calculated in $O(N \log N)$ time in total if we use a binary search technique on a RMQ segment tree.

Hence we can calculate the values of $R_1, R_2, \ldots, R_N$ in $O(N \log N)$ time in total.

There are several ways to calculate the values of $R_1, R_2, \ldots, R_N$ in advance. Here we only explain one way. Some contestants may solve this task by other methods. For example, in the operation 1, we can calculate $M_{D+1}, M_{D+2}, \ldots, M_N$ in $O(N)$ time by a "sliding minimum technique." There are also completely different methods. For example, we can calculate R_i in the increasing order of A_i using `std::set`.