Editorial

https://discuss.codechef.com/t/sead-editorial/4626

You are given ascending-sorted array **A[ ]**, consisting of **N** integers. You are to calculate the answers for **M** queries of the following type: You are given two integers **T** and **D**. You should find the minimal integer **L** such that there is some integer **R (L≤R)** for which the following conditions are satisfied:

A[L]+D>=A[L+1],
A[L+1]+D>=A[L+2],
…
A[R-1]+D>=A[R],
A[R]<=T < A[R + 1] (we assume, that **A[N + 1]** is infinitely big).

Explanation:
The first observation, that we should make, is that there is the only **R** for each query **(T, D)**, that could satisfy to the conditions from the statement.

It isn't hard to be sure about that. As far as **A[ ]** is an ascending-sorted array, let's define **R** as the largest integer, such that **A[R] <= T**. Then **T** is less than **A[R + 1]**(otherwise **R** is not the largest integer that we are looking for).

Please, note that as far as **A[1]<=T**, such **R** always exists.

How to find **R** efficiently? We can use a binary search and find it in **O(log N)** time.

OK, now we know how to find **R**. But we were asked to find **L**, not **R**.

So, let's do it.

Firstly, let's fix **R**. We already know, that it's unique for each query and there is no way to choose another **R**, so we need to adapt **L** under **R**.

Let's understand, that if **L = R,** then the conditions from the statement are satisfied. The problem is that **R** may be not the minimal such **L**, that satisfies. On the other hand, we can observe, that if for **L = X** the conditions are satisfied, then for **L = X + 1** the conditions are satisfied as well.

More formally, let's consider boolean function **G(X), 1 <= X <= R. G(X)** is true if and only if the conditions are satisfied for **L = X**, otherwise it's false. This function is monotone.

We will find **L** using binary search on answer in **O(log N * < complexity of calculation of G >)** time.

So, the last subtask, that we should discuss, is how to calculate **G** efficiently.

Firstly, let's forget about this condition: **A[R]<= T < A[R + 1]**. It doesn't play any role as far as **R** is fixed and has already satisfied to that condition.

Let's assume, that we are calculating **G(X)** right now. What should we check?

In order to make **G(X)** true, the following conditions must be satisfied:

A[X] + D >= A[X + 1],
A[X + 1] + D >= A[X + 2],
…
A[R - 1] + D >= A[R].

Let's work a little bit with the inequalities. I.e. let's rewrite them in such a way, that **D** is the only summand to the right of the comparison sign:

A[X + 1] - A[X] <= D,
A[X + 2] - A[X + 1] <= D,
…
A[R] - A[R - 1] <= D.

Let's build up array **B[]**, such that **B[i] = A[i + 1] - A[i]** for each **1 <= i < N**.
Then,

B[X] <= D,
B[X + 1] <= D,
…
B[R - 1] <= D.

We can finally rewrite the inequalities like:

max(B[X], B[X + 1], …, B[R - 1]) <= D.

Well, now it's quite a well-known problem.

G(X) is true if and only if the maximal number of array **B[ ]** on the range **[X, R - 1]** is not greater than **D**.
We can use a sparse table in order to answer this query efficiently. If you don't know sparse tables at all, don't be upset. The constraints in this problem are quite soft, you can use a simple segment tree.

Nevertheless, the solution, which uses a sparse table instead of a segment tree, works much faster and has a better complexity.

Here is a pseudocode, that shows the implementation of building up array **B[ ]** and making sparse table **P[ ][ ]** based on **B[ ]**.