Solutions PO camp 2026 day 1

1 Strawberries

Strawberries

Author: Nils Gustafsson

Description

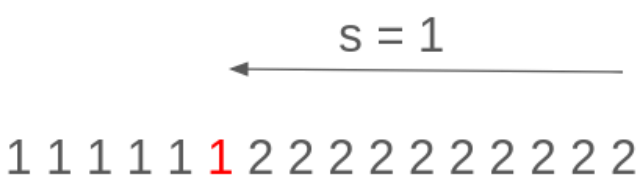
There are N strawberries sorted by weight, with weights between 1 and $M-1$. In one query, you can check if a strawberry is $l = s$ or not. But if it is, it gets destroyed. Find the weights without losing more than $M-1$ berries, in at most N queries.

Strawberries

Description

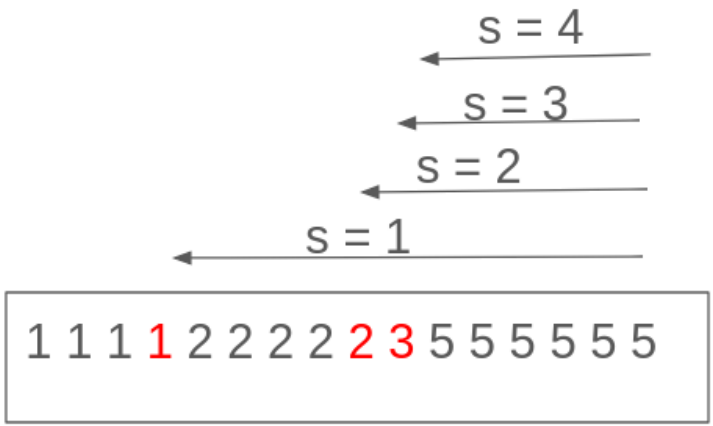
There are N strawberries sorted by weight, with weights between 1 and $M-1$. In one query, you can check if a strawberry is $l = s$ or not. But if it is, it gets destroyed. Find the weights without losing more than $M-1$ berries, in at most N queries.

Subtask 1: M = 2:



Strawberries

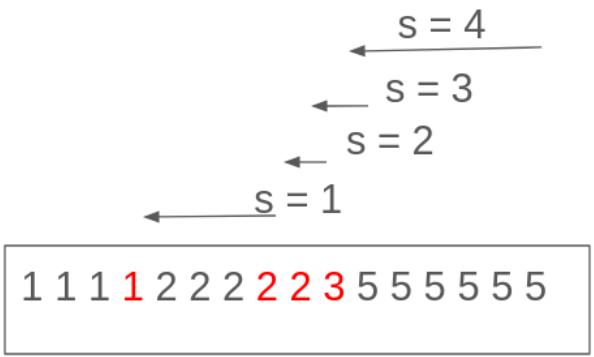
58 points: NM queries:



Do the same thing as in Subtask 1, but repeat for $s = 1, 2, 3, ..., M - 1$.

Strawberries

Full score:



We want to accomplish the same thing as in the 58-point solution, but with a single sweep from right to left. Start with $s = M - 1$, and decrease it every time a berry gets lost. If a berry survives, it must have weight $s + 1$.

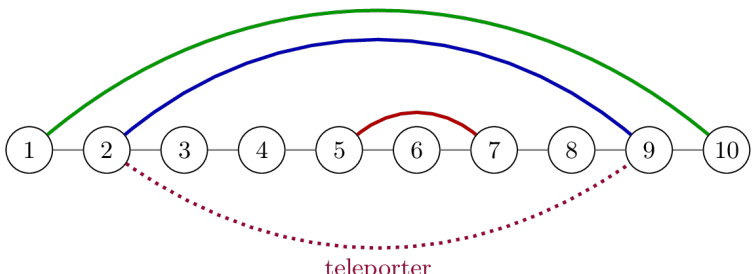
2 The Line

The Line

Author: Joshua Andersson

Description

We have a line graph with N ($N \leq 10^9$) nodes. It takes 1 second to cross an edge. We can add one edge (a, b). Place the edge as to minimize $\sum \text{dist}(s_i, t_i)$ for all K pairs (s_i, t_i). $K \leq 10,000$

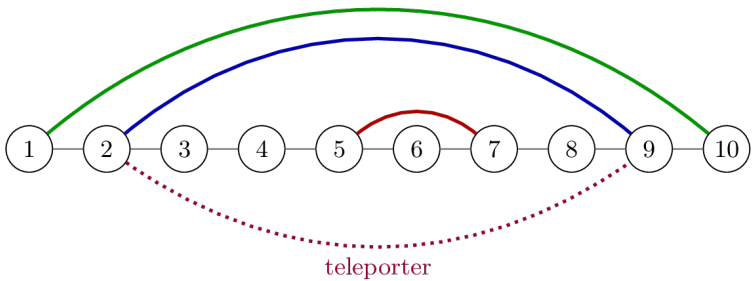


The Line

dist(1,10)=3: follow the path $1 \to 2 \to 9 \to 10$

dist(2,9)=1: follow the path $2 \to 9$

dist(5,7)=2: follow the path $5 \to 6 \to 7$



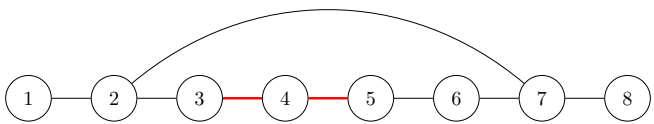
Subtask 1: $N, K \leq 50$

There are $O(N^2)$ possible teleporter placements. For each one, compute the total distance in $O(NK)$ by doing a BFS for every person. $O(N^3K)$ in total.

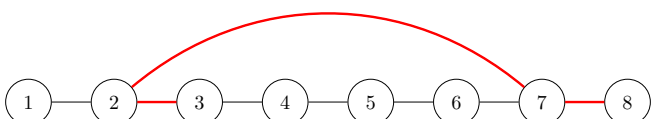
Subtask 2: $N, K \leq 500$

Let's still try all $O(N^2)$ possible placements, but optimize the distance computation. We can realize that we will only use the teleporter at most once! Otherwise there is a cycle, and shortest paths don't have cycles. If we add the teleporter between train stations a and b, then the shortest path will be one of

$$s \to t$$

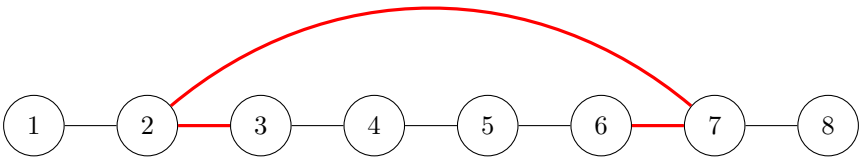


$$s \to a \to b \to t$$



Subtask 2: $N, K \leq 500$ (ii)

$$s \to b \to a \to t$$



The exact formula becomes

$$\begin{aligned}\min(&|s-t|,\\ &1+|s-a|+|t-b|,\\ &1+|s-b|+|t-a|)\end{aligned}$$

Thus, we can compute the sum of distances in $O(K)$ per candidate, for $O(N^2K)$ in total.

Subtask 4: $K \leq 500$, N big

Fake idea

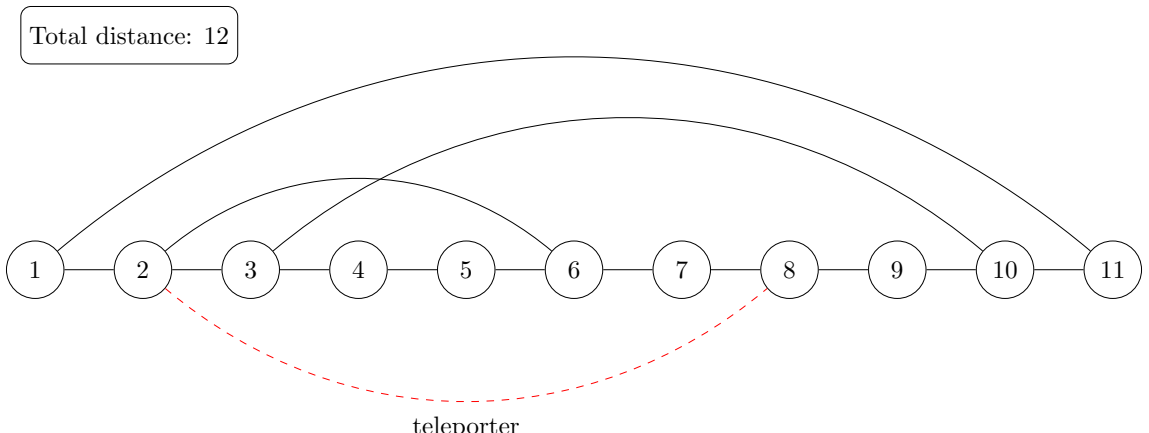
Only try teleporters with endpoints (s, t) for some (s, t) in the input. This doesn't work. Smallest counterexample we found was too complicated to understand.

Fixing the fake idea

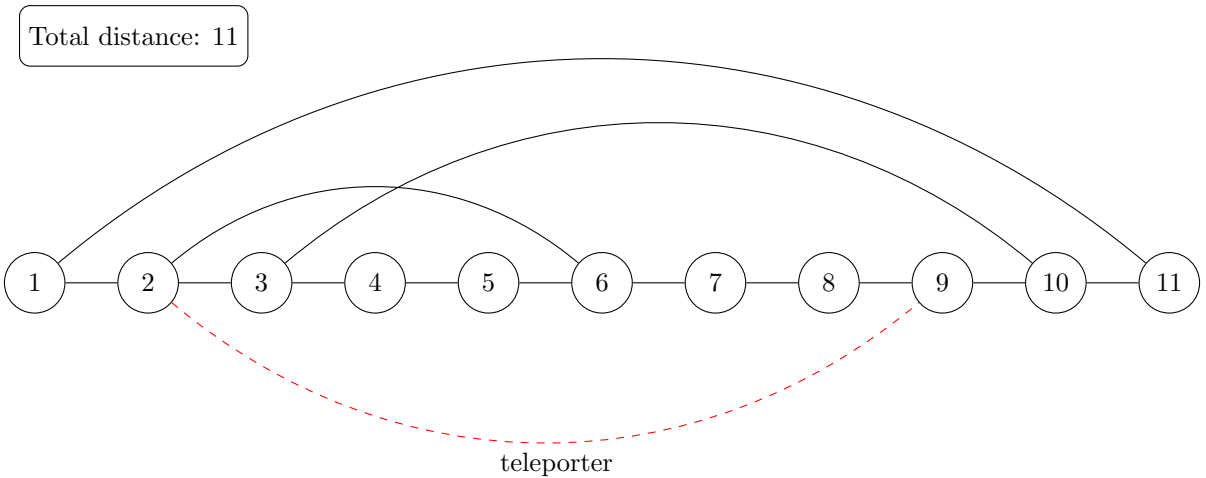
Let E be the set of all endpoints. I.e., every s or t value. It suffices to only check teleporters (a, b) where $a \in E$ and $b \in E$. Why?

Subtask 4: $K \leq 500$, N big (ii)

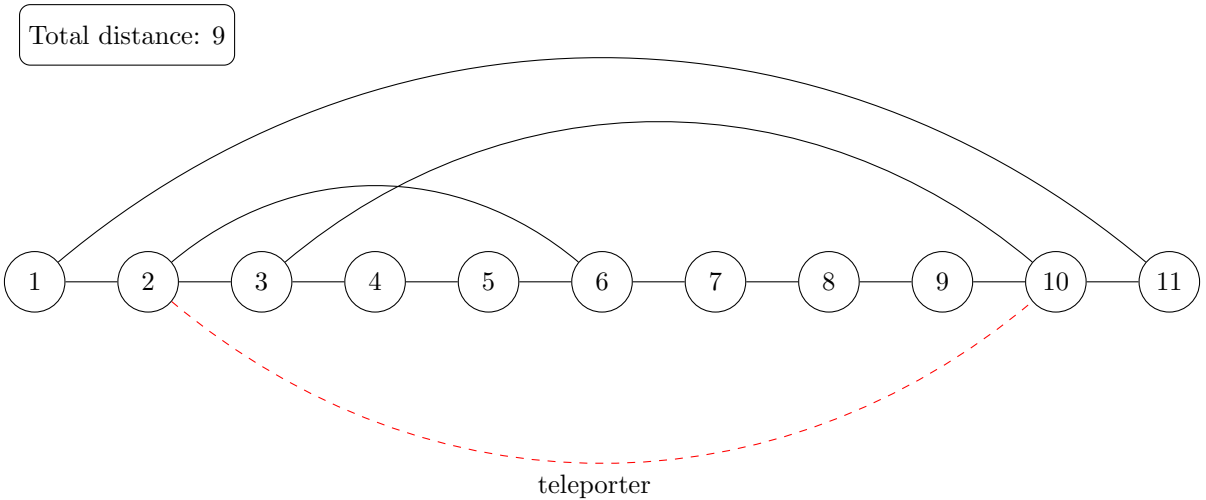
Suppose that $b \notin E$. Then, we can always move the teleporter towards the side with more people and decrease the total distance.



Subtask 4: $K \leq 500$, N big (iii)



Subtask 4: $K \leq 500$, N big (iv)



Subtask 4: $K \leq 500$, N big (v)

The algorithm

Thus, we have reduced our problem to one where we only have to consider $O(K^2)$ pairs of endpoints for the portal instead of $O(N^2)$. In total, we get $O(K^3)$.

Subtask 3: $N, K \leq 2000$

Let's consider the matrix $\text{cost}[a][b] = $ total cost if the teleporter is built at (a, b). Let's see how a single pair (s, t) influences cost. Here, we have a single person $(1, 6)$.

5	5	5	5	4	3	2	3
5	5	5	4	3	2	1	2
5	5	5	5	4	3	2	3
5	4	5	5	5	4	3	4
4	3	4	5	5	5	4	5
3	2	3	4	5	5	5	5
2	1	2	3	4	5	5	5
3	2	3	4	5	5	5	5

Subtask 3: $N, K \leq 2000$ (ii)

So we get manhattan-like circles centered around (s, s) and (t, t). The function is still pretty annoying to work with. Let's instead try to maximize savings relative to $\sum \text{dist}(s_i, t_i)$. The new function is then

$$\begin{aligned}&\max(0,\\ &\quad |s-t| - (1 + |a-s| + |b-t|),\\ &\quad |s-t| - (1 + |b-s| + |a-t|))\end{aligned}$$

... which looks more complicated, but has a nice shape.

0	0	0	0	1	2	3	2
0	0	0	1	2	3	4	3
0	0	0	0	1	2	3	2
0	1	0	0	0	1	2	1
1	2	1	0	0	0	1	0
2	3	2	1	0	0	0	0
3	4	3	2	1	0	0	0
2	3	2	1	0	0	0	0

Subtask 3: $N, K \leq 2000$ (iii)

Solving the manhattan circles

If we fix the row, every person adds a function of shape $|x - c_1| + c_2$ to the row.

Since N is small, we can explicitly construct this matrix. We can do the correct additions by adding $+1$ to the start of the function and -1 to the end, and then taking prefix sum over every row twice.

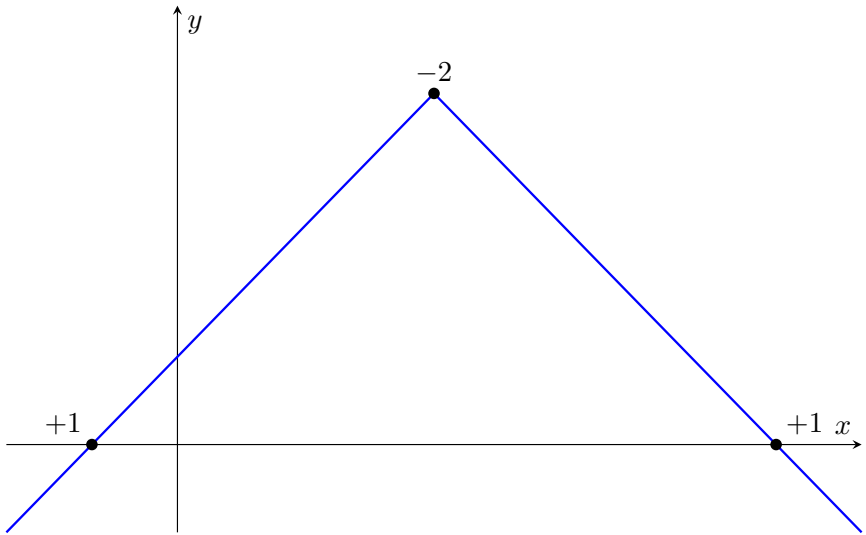
Subtask 5: $K \leq 2000$, $N \leq 2 * 10^5$

Let's combine the ideas from the two previous subtasks. We only need to care about rows in the matrix where there is a person. This immediately gives $O(NK + K^2 \log(K))$. (With a lot of determination, this might be enough for this subtask).

Of course, we also know that we only need to consider columns where $b \in T$. We will to do a sweep. For most columns, the value of the derivative is constant, so we can jump between columns that exist in T, and those where the derivative changes. To compute the double prefix sum along the way, we maintain y and dy (the derivative of y).

Every person adds 3 (or 6) events to every row. The two manhattan circles cannot intersect.

Subtask 5: $K \leq 2000$, $N \leq 2 * 10^5$ (ii)



Subtask 5: $K \leq 2000$, $N \leq 2 * 10^5$ (iii)

This results in $O(K^2 \log(K))$, where the log factor comes from sorting the events for every row.

Full solution: $K \leq 10,000$

The previous solution is almost fast enough, we just need to save the log factor by finding an $O(K^2)$ solution. (Python? Womp womp). We will again consider each row in the matrix. Every person adds a triangle shape to the cost. As we go through the rows one by one, the triangle intervals first grow and then shrink. Thus, if we have two pointers for each person that store the endpoints of their triangle, these pointers will move $O(N)$ amortized each. Since we only care about endpoints, we can make this $O(K)$ amortized if we are careful.

To get full score, it is probably a good idea to just have K events in the sweep, one for each right endpoint, as opposed to $3K$ events. These K events are stationary, so no sorting required.

Alternative (slightly slower) full solution

It might be tempting to store the $+1$, -2, $+1$ in separate lists, and update them. Unfortunately, their relative order does change. If we split each $+1$ into two lists based on the "sign" of the abs, and then move elements between the lists as we sweep, we can maintain 5 sorted lists of events in $O(K^2)$ time amortized. When sweeping, we do a 5-way merge.

Cheesy full solution

Use radix sort to sort the events. This is tight to fit in the time limit. Some tips to save constant factor if you want to squeeze in this:

- Tune the radix sort well. A base of 2^{11} worked best for me.
- Only choose $a \in S$ and $b \in T$, not $a \in (S \cup T)$ and $b \in (S \cup T)$
- Make sure you only get $3K$ events, not $6K$. This is possible because the optimal teleporter will WLOG have $a \leq b$.
- Ensure you use $O(K)$ memory by considering every left endpoint independently. Avoid allocating new memory. Avoid using `push_back`/`emplace_back`.

Challenge

Can we query for the value of a teleporter in $O(1)$ or $O(\log(N+K))$?

Hard challenge: solve it for $N, K \leq 2 * 10^5$.

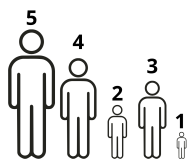
3 Under Supervision

Under Supervision

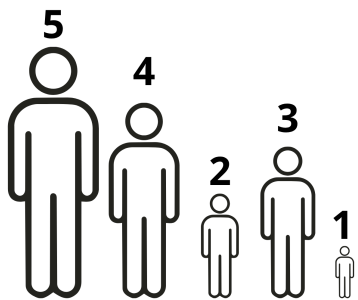
Author: Joshua Andersson Prepared by: Harry Zhang

Description

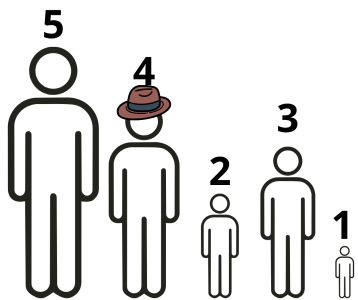
Given a permutation of $1, \ldots, N$, find the minimum number of "observers" such that every citizen is either an observer or visible to one. Two citizens i, j see each other if and only if everyone between them is shorter than both.



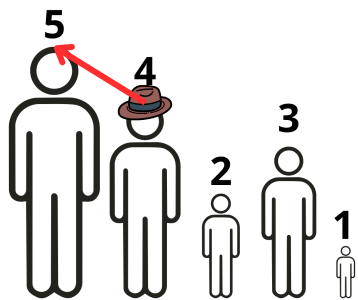
Under Supervision (ii)



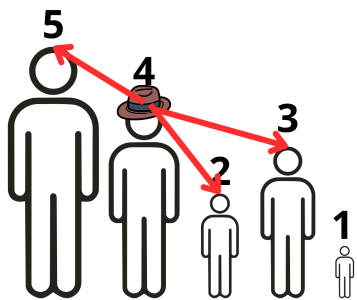
Under Supervision (iii)



Under Supervision (iv)



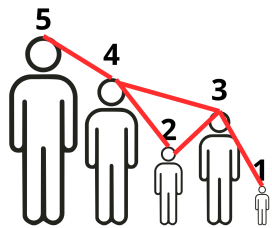
Under Supervision (v)



Under Supervision (N <= 15)

If person i sees person j, then j will also see person i.

We can build a visibility graph, where two people share an edge if they can see each other. $O(N^3)$.



Under Supervision (N <= 15) (ii)

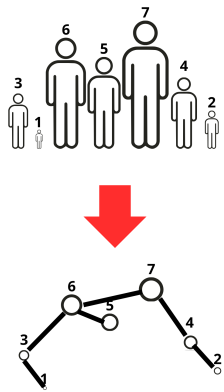
The problem then becomes to find the minimum dominating set of this visibility graph – We want to find the size of the minimum set of special nodes such that all nodes are neighbors to at least one special node.

We can thus brute all 2^N possible sets, and for each set verify that all nodes are neighbors to some node in the set.

One can write a solution that runs in $O(2^N \cdot N^2)$.

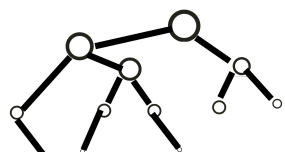
Under Supervision (N <= 200)

The key structural insight is that the visibility graph relates directly to the Cartesian tree of the permutation: the root is the maximum element, and the left/right subtrees consist of the elements to its left/right.



Under Supervision (N <= 200) (ii)

A crucial property is that within an interval $[l, r]$ with maximum at position m, the set of people in $[l, m-1]$ who can see m is exactly the right spine of the Cartesian tree of $[l, m-1]$ (the path from the root following right children down to $m-1$). Symmetrically, from $[m+1, r]$, the people who can see m are the left spine.



Under Supervision (N <= 200) (iii)

We define

$$\mathrm{dp}[l][r][\mathrm{el}][\mathrm{er}][\mathrm{ol}][\mathrm{or}]$$

as the minimum number of observers needed to supervise everyone in $[l, r]$. The four boolean parameters describe how observation flows between this subtree and its parent:

- el/er: whether this subtree receives observation from **outside** (entering from the parent), flowing down the left/right spine respectively.
- ol/or: whether this subtree needs an observer that can observe **outward** toward the parent, along the left/right spine, respectively.

These flags propagate along the spines through the recursion. For example, or $= 1$ for interval $[l, m-1]$ means some node on its right spine is an observer that can see m. Similarly, er $= 1$ propagates down the right spine: if the parent's root m is an observer, it observes every node on the right spine, modeled by passing er into each successive right child.

For each interval $[l, r]$, we find the maximum m and consider two cases:

Under Supervision (N <= 200) (iv)

1. **Make m an observer.** We pay cost 1. Since m can see every node on the right spine of $[l, m-1]$ and the left spine of $[m+1, r]$, both children receive external observation (the left child gets $\mathrm{er} = 1$, the right child gets $\mathrm{el} = 1$). Since m is an observer, it can observe the parent in both directions, so $\mathrm{ol} = \mathrm{or} = 1$. We recurse on both children and combine the costs, minimizing over all values of their outward flags.

2. **Don't make m an observer.** Then m must be observed by someone else. The only candidates are:
   - External observation from the parent (el or er).
   - An observer on the right spine of $[l, m-1]$ (left child has $\mathrm{or} = 1$).
   - An observer on the left spine of $[m+1, r]$ (right child has $\mathrm{ol} = 1$).

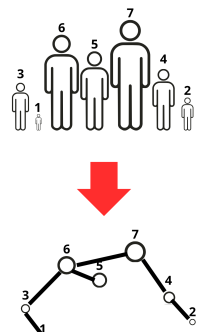
   If none of these hold, the state is invalid. Otherwise, the children don't receive external observation from m (since m is not an observer), and the outward flags of the whole interval are inherited from the outermost children: ol from the left child, or from the right child.

The final answer is $\min_{\mathrm{ol},\ \mathrm{or}} \mathrm{dp}[0][n-1][0][0][\mathrm{ol}][\mathrm{or}]$, since the root interval has no external observation.

There are $O(N^2)$ intervals, and finding the maximum per interval takes $O(N)$ time, giving $O(N^3)$ total. The boolean flags add only a constant factor ($2^4 = 16$ states per interval).

Under Supervision (N <= 1000)

The interval DP above has $O(N^2)$ states. But notice that only $O(N)$ of those intervals actually appear in the recursion: each interval corresponds to a node in the Cartesian tree. We can make this explicit by building the Cartesian tree and indexing the DP by node rather than by interval.



Under Supervision (N <= 1000) (ii)

We build the Cartesian tree by recursively finding the maximum in $[l, r]$ with a linear scan, making it the root, and recursing on $[l, m-1]$ and $[m+1, r]$. Each subtree now represents an interval, and we can use the index of its root to represent it. The DP becomes

$$\mathrm{dp}[i][\mathrm{el}][\mathrm{er}][\mathrm{ol}][\mathrm{or}]$$

with the same transitions as before. Since there are exactly N nodes, the DP runs in $O(N)$.

The bottleneck is building the tree. The recursive max-finding takes $O(N^2)$ in the worst case (when the tree is maximally unbalanced, e.g. a sorted permutation), giving $O(N^2)$ total.

Under Supervision (N <= 3e5, random permutation)

On a uniformly random permutation, the Cartesian tree is a random binary search tree, which is balanced in expectation with $O(\log N)$ depth. The recursive max-finding then takes $O(N \log N)$ expected time (analogous to quicksort with random pivots), which is fast enough.

Under Supervision (Full Solution)

To handle adversarial inputs, we need to build the Cartesian tree in $O(N)$ worst case. This can be done using a monotonic stack:

Process positions left to right, maintaining a stack of nodes whose right child is not yet determined. When processing position i:

1. Pop all stack elements with value less than p_i. The last popped element becomes the left child of i.
2. If the stack is non-empty, i becomes the right child of the top element.
3. Push i onto the stack.

After processing all positions, the bottom of the stack is the root.

This builds the full Cartesian tree in $O(N)$ time, and combined with the $O(N)$ tree DP gives an $O(N)$ solution overall.

It is also possible to build the Cartesian tree in $O(N \log(N))$ by speeding up the recursive max-finding using a segment tree.