




NOI 2018 Solution Presentation

NOI Scientific Committee



Asia-Pacific Informatics Olympiad Invitation



APIO Invitation

All silver and gold medallists are cordially invited to participate in the Asia-Pacific Informatics Olympiad
The selection for the Singapore International Olympiad in Informatics team will be 50% NOI, 50% APIO
Top 4 Singapore Citizens/Permanent Residents will be invited to form the team
Foreigners are not eligible for the team, but are invited to participate in the APIO

The to-be-confirmed details are as follows:

Date: 12th May 2018 (Saturday)

Time: 9am to 2pm

Venue: NUS COM1 Programming Lab 2

You will receive a formal invitation via email when the details are confirmed

You can find out more information at apio2018.ru



Scientific Committee



Dr. Steven Halim

Head Coach of NUS ICPC Teams
(2008-2014, 2017)

Team Leader of Singapore IOI Team
(2009-2013, 2015-present)

Deputy Chairman for IOI 2020, Singapore





Dr Wing-Kin Sung, Ken

Has multiple awards for research
contribution in Algorithms and
Bioinformatics

Senior group leader in Genome Institute of
Singapore





Gan Wei Liang

IOI International Scientific Committee
Singapore Representative (2018 - present)

Singapore IOI Team Coach (2016)

Singapore IOI Team Member (2012 - 2013)





Dr Frank Stephan

Research Interests:

- Recursion Theory and Kolmogorov Complexity
- Learning Theory
- Computational Complexity
- Automata and Formal Languages





Bernard Teo Zhi Yi

Singapore IOI Team Member (2012 - 2013)





Clarence Chew Xuan Da

Singapore IOI Team Member (2016)



Zhang Guangxuan

Singapore IOI Team Member (2016 - 2017)



Pang Wen Yuen

Singapore IOI Team Member (2015 - 2017)





Lim Li

Singapore IOI Team Member (2017)





Problem Statistics



Interesting Statistics

167 participants, 14 gold, 28 silver, 35 bronze medallists

Collecting Mushrooms: 32 people scored 100

Journey: 22 people scored 100

Lightning Rod: 24 people score 100

City Mapping: highest score is 95

Safety: highest score is 40



Task 1: collectingmushrooms

Problem Author: Lim Li



Abridged Problem

Given an $R \times C$ grid, count the number of mushrooms within a D square of at least K sprinklers.



Subtask 1 (9%)

$1 \leq R, C \leq 500, D = \max(R, C), K=1$

Since every sprinkler can water the whole grid, each mushroom only needs 1 sprinkler to be harvestable, and the grid is guaranteed to have at least 1 sprinkler, you just need to count the number of mushrooms in the grid.



Subtask 2 (10%)

$1 \leq R, C \leq 500, D = \max(R, C)$

Same as subtask 1, but now you also need to count the number of sprinklers. If the number of sprinklers at least K , print the number of mushrooms. Else, print “0”.



Subtask 3 (18%)

$1 \leq R, C \leq 500, D = 1, K = 1$

$D=1$, so check the 3×3 square around each mushroom. If there is a sprinkler, then it is harvestable.

Complexity: $O(9RC)$



Subtask 4 (23%)

$1 \leq R, C \leq 500$, no. of mushrooms ≤ 500 , no. of sprinklers ≤ 500

Since D can be big, checking the $(D+1) \times (D+1)$ square around each mushroom will TLE.

Instead, loop through the list of sprinklers to see which sprinkler can water that mushroom



Subtask 4 (23%)

For each mushroom:

 water:=0

 For each sprinkler:

 if the mushroom is in the sprinklers range:

 water++

 if water \geq K

 This mushroom is harvestable



Subtask 5 (19%)

$$R = 1$$

Since $R=1$, the grid is now a line. The range of each sprinkler is now a line of length $(2D+1)$.

To count the number of sprinklers that can water each mushroom, we can keep a prefix sum.



Subtask 5 (19%)

Value	1				1	1				1			1	1
Sum	1	1	1	1	2	3	3	3	3	4	4	4	5	6



Subtask 5 (19%)

Value	1				1	1				1			1	1
Sum	1	1	1	1	2	3	3	3	3	4	4	4	5	6



Subtask 5 (19%)

Value	1				1	1				1			1	1
Sum	1	1	1	1	2	3	3	3	3	4	4	4	5	6



Subtask 5 (19%)

Value	1				1	1				1			1	1
Sum	1	1	1	1	2	3	3	3	3	4	4	4	5	6



Subtask 5 (19%)

This allows us to find the number of sprinklers in a range in $O(1)$ time.



Subtask 6 (21%)

No additional constraints.

The solution for subtask 6 is a generalisation of the prefix sum idea from subtask 5. We store a 2D prefix sum.



Subtask 6 (21%)

1					
			1		
	1				
				1	
		1		1	
	1				1

1	1	1	1	1	1
1	1	1	2	2	2
1	2	2	3	3	3
1	2	2	3	4	4
1	2	3	4	6	6
1	3	4	5	7	8

Subtask 6 (21%)

1					
			1		
	1				
				1	
		1		1	
	1				1

1	1	1	1	1	1
1	1	1	2	2	2
1	2	2	3	3	3
1	2	2	3	4	4
1	2	3	4	6	6
1	3	4	5	7	8

Subtask 6 (21%)

1					
			1		
	1				
				1	
		1		1	
	1				1

1	1	1	1	1	1
1	1	1	2	2	2
1	2	2	3	3	3
1	2	2	3	4	4
1	2	3	4	6	6
1	3	4	5	7	8

Subtask 6 (21%)

1					
			1		
	1				
				1	
		1		1	
	1				1

1	1	1	1	1	1
1	1	1	2	2	2
1	2	2	3	3	3
1	2	2	3	4	4
1	2	3	4	6	6
1	3	4	5	7	8



Subtask 6 (21%)

This allows us to find the number of sprinklers in a 2D range in $O(1)$ time.



Task 2: Journey

Problem Author: Frank Stephan



Abridged Problem

Input n (Number of Cities), m (Maximum Days used up), h (Flights out per City for Cities $0, 1, \dots, n-2$);

$n \times h$ data pairs (Next Destination, Minimum Stay).

Output: For each number of hotel nights $(0, 1, \dots, m-1)$, the number of itineraries which go from 0 to $n-1$ and go forward in each flight;

Numbers above 5000000001 are replaced by 5000000001 .



Example (Amman -> Cairo -> Dakar -> Brasilia)

In this example, $n = 4$ (for cities) and $h = 3$ (three outgoing flights per city); m is number of total nights (upper bound), say 4. Cannot use flight Dakar -> Cairo, as this is wrong direction. Cannot visit all cities in Amman - Cairo - Dakar - Brasilia, as too many nights.

From Amman	To Cairo, 2+ Nights	To Dakar, 3+ Nights	To Brasilia, 0+ Nights
From Cairo	To Dakar, 3+ Nights	To Dakar, 4+ Nights	To Brasilia, 0+ Nights
From Dakar	To Cairo, 0+ Nights	To Brasilia, 2+ Nights	To Brasilia, 0+ Nights



Algorithm and Data Types

Use array $a[i,p]$ being the number of itineraries from City 0 to City i with p nights;

Let $\text{Minadd}(x,y)$ denote the update $x = \min(x+y, 500000001)$.

Read n,m,h and create $n \times m$ array a with all entries initialised 0; let $a[0,0]=1$;

For $i = 0,1,\dots,n-2$ For each Flight leaving i Do Begin

Read next stop j and minimum stay k ;

If $i < j$ Then Do Begin

For $p=0$ to $m-1-k$ For $q=p$ to $m-1-k$

Do Begin $\text{Minadd}(a[j,q+k], a[i,p])$ End End;

Output $a[n-1,0], a[n-1,1], \dots, a[n-1,m-1]$.



Improved Algorithm

Recall that $\text{Minadd}(x,y)$ denotes $x = \min(x+y, 500000001)$.

Read n,m,h and Initialise all entries $a[i,p]$ with 0 except for $a[0,0]$ which is 1;

For $i = 0, 1, \dots, n-2$ For each Flight leaving i Do Begin

Read next stop j , minimum stay k ;

If $i < j$ Then Do Begin $s=0$;

For $p=0$ to $m-1-k$ Do Begin $\text{Minadd}(s, a[i,p]); \text{Minadd}(a[j, p+k], s)$ End End;

Output $a[n-1,0], a[n-1,1], \dots, a[n-1, m-1]$.



Runtime Analysis

Both the Algorithm and the Improved Algorithm have an outer loop over $h \times n$ items and processes in each loop one outgoing flight from city i to city j and updates the entries at city j .
The inner loops update m entries.

Original algorithm: each entry is updated $O(m)$ times;

Improved Algorithm: each entry is only updated $O(1)$ times.

Original Algorithm has runtime $O(n \times m^2 \times h)$;

Improved Algorithm has runtime $O(n \times m \times h)$.



Subtasks

Subtask 1: Only 1 Stop-Over possible, algorithm can be implemented without needing the array `a` to store intermediate information;

Subtask 2: All flights are forward, solutions without if-statement can score;

Subtask 3: Larger inputs; original algorithm and improved algorithm score;

Subtask 4: Even larger inputs; as far as we tested it, only implementations of the improved algorithm scored for this subtask.



Task 3: lightningrod

Problem Author: Zhang Guangxuan



Abridged Problem

Find the minimum number of points to be chosen as lightning rods among N points, such that every point is covered by a lightning rod.

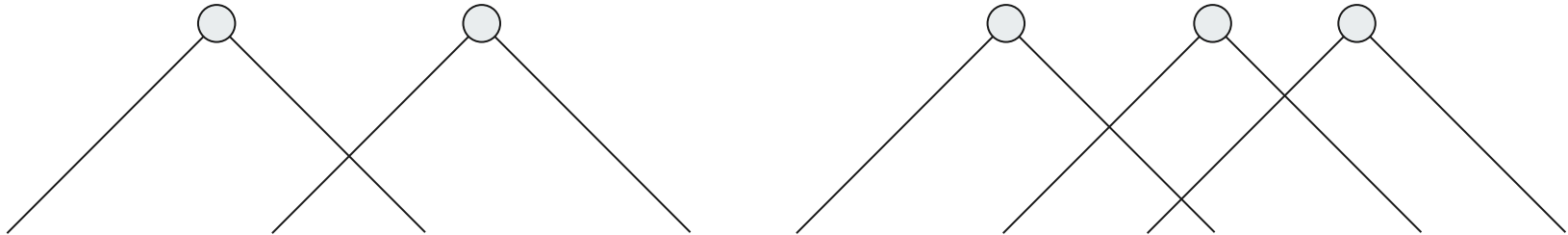
Point i is covered by Point j if and only if $|X_i - X_j| \leq Y_j - Y_i$.



Subtask 1 (4%)

$Y_i = 1$ for all i .

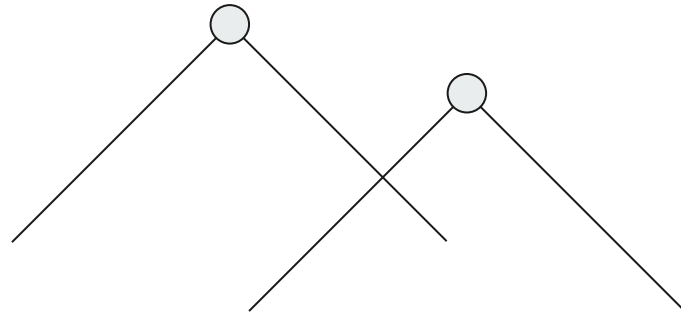
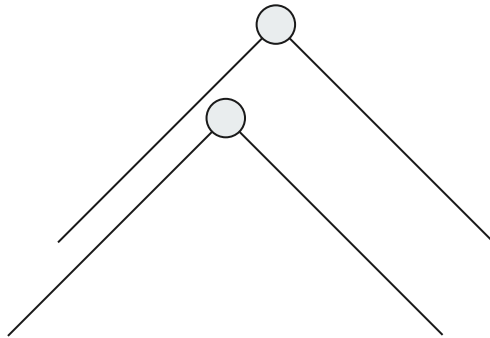
All right angle triangles cover exactly 1 point, so the answer is N



Subtask 2 (7%)

$N = 2$

If $|X_0 - X_1| \leq |Y_0 - Y_1|$, one point can cover the other point, so the answer is 1, otherwise the answer is 2.





Subtask 3 (12%)

$N \leq 20$

Do a $O(2^N)$ brute force for selecting a subset of points, $O(N^2)$ to check if the selection is valid.

Overall complexity: $O(2^N N^2)$



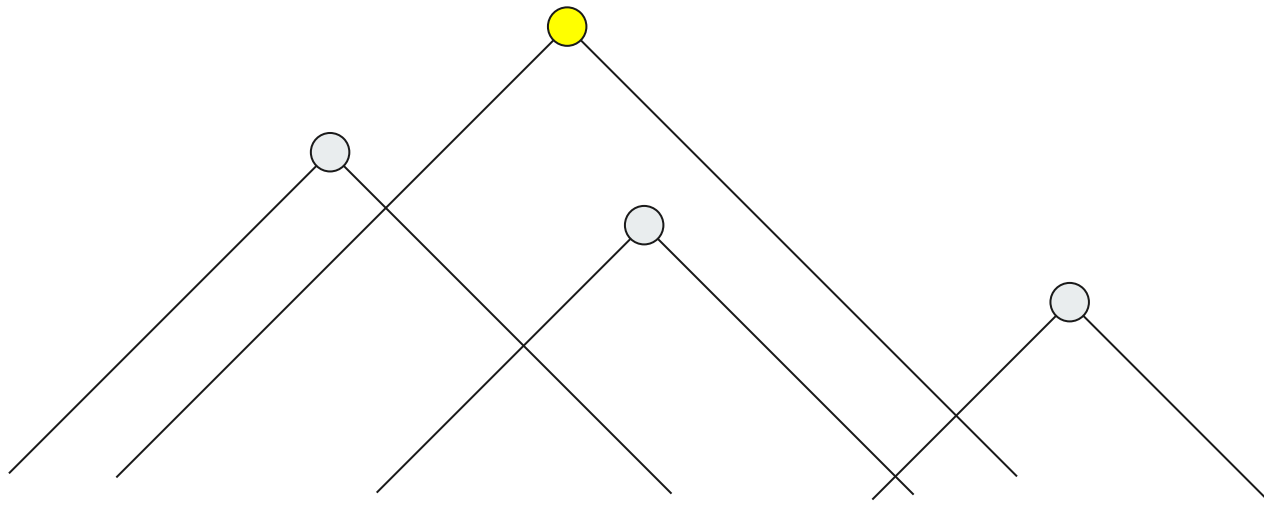
Subtask 4 (21%)

$N \leq 2000$

If we process points by decreasing Y_i , then we have to pick the current point if and only if previous points have not covered this point. We can check this condition for every point in $O(N)$ by looping through previous points, giving a total complexity of $O(N^2)$.

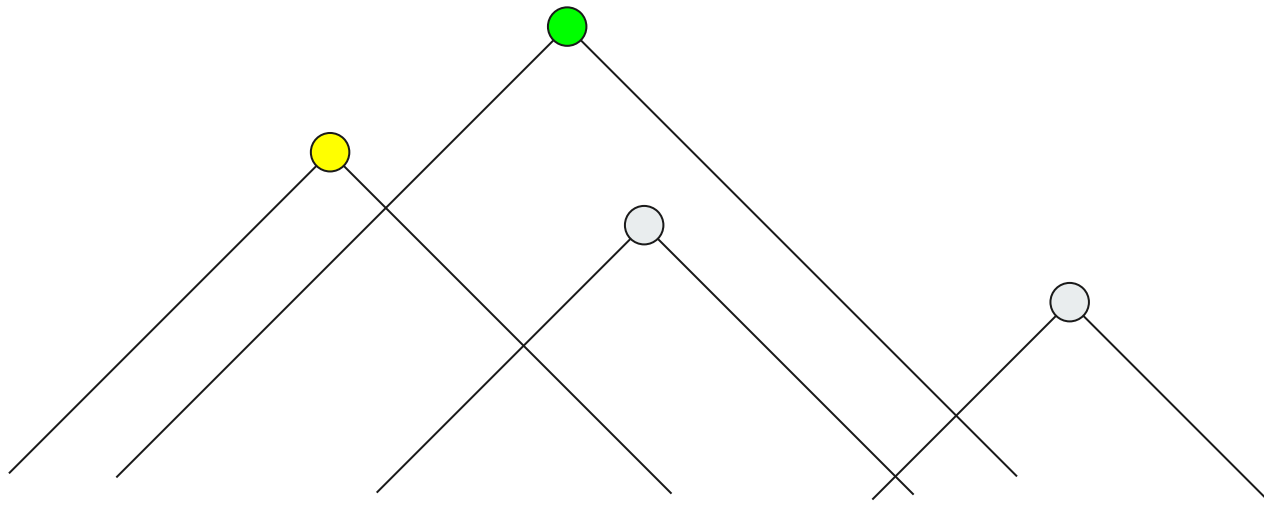


Subtask 4 (21%)



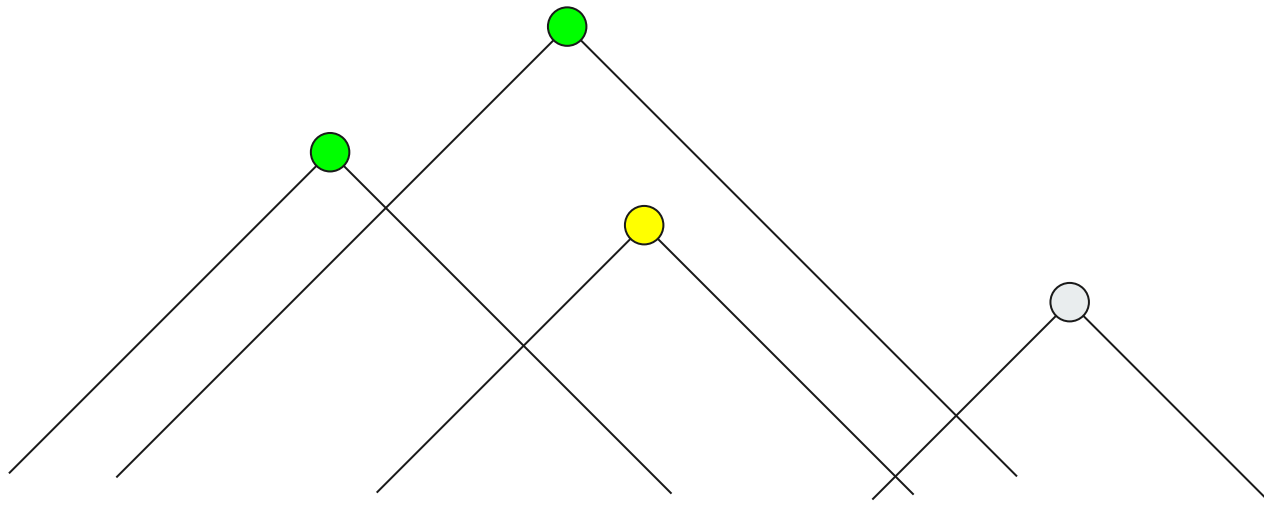


Subtask 4 (21%)



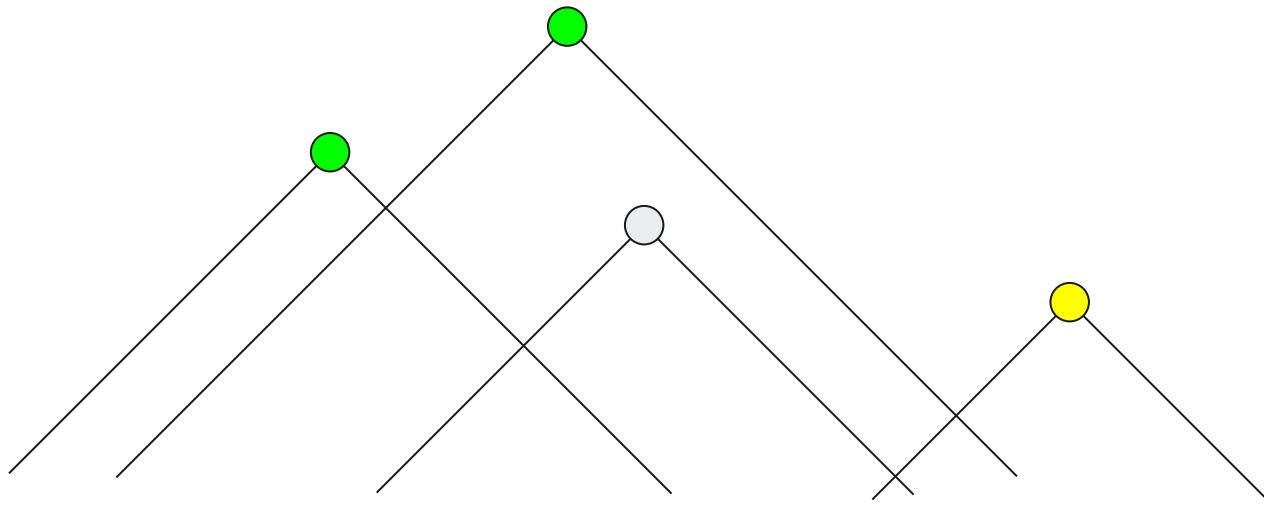


Subtask 4 (21%)



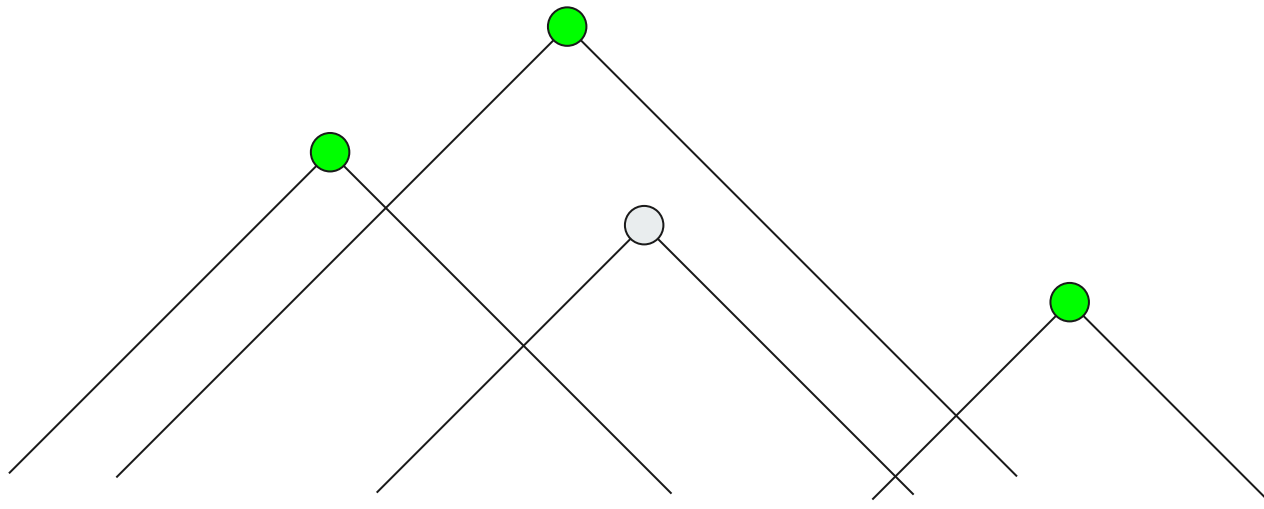


Subtask 4 (21%)





Subtask 4 (21%)





Subtask 5 (26%)

$N \leq 200000$

We can improve on the subtask 4 solution. Instead of looping through all previous points to check if the current point is already covered, we just have to look for the highest point on the left and right of the current point.



Subtask 5 (26%)

Let the current point be (X_i, Y_i) . For points to the left, we look for the largest $X_j + Y_j$, and compare it with $X_i + Y_i$. For points to the right, we look for the largest $Y_j - X_j$, and compare it with $Y_i - X_i$. The current point is chosen if and only if no points cover the current point.



Subtask 5 (26%)

This can be in $O(\log N)$ using 2 range max point update segment trees or fenwick trees, where the first tree stores $X_i + Y_i$, second tree stores $Y_i - X_i$. For each point, we do a prefix query on the first tree, suffix query on the second tree. If the current node is chosen, update the current position with $X_i + Y_i$ in the first tree, $Y_i - X_i$ in the second tree



Subtask 6 (10%)

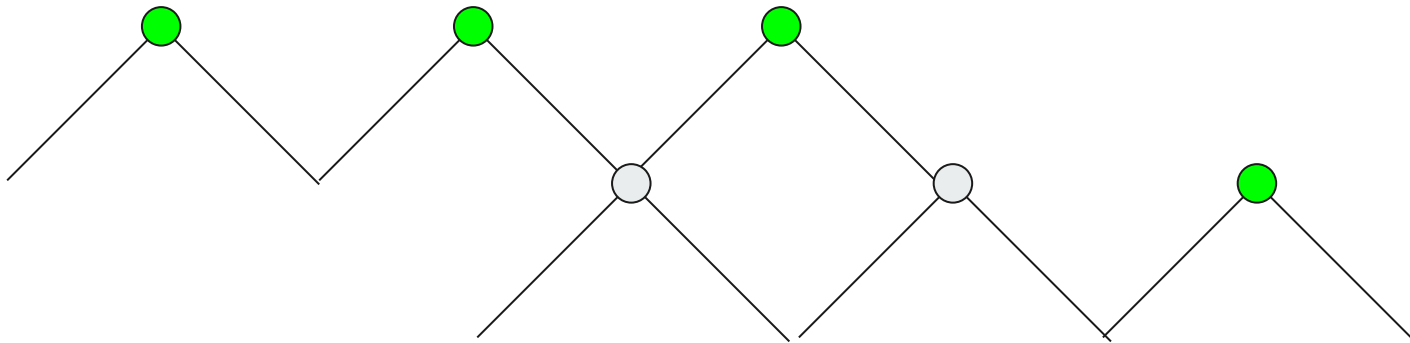
$N \leq 100000000$, $X_i = i$, $0 \leq Y_i \leq 1$

Each point can only be covered by the point directly on the left or right. It can be easily checked in $O(1)$ for each point.

Note that for subtasks where $N \leq 10\,000\,000$, fast input is required to ensure the programme runs within 1 second.



Subtask 6 (10%)





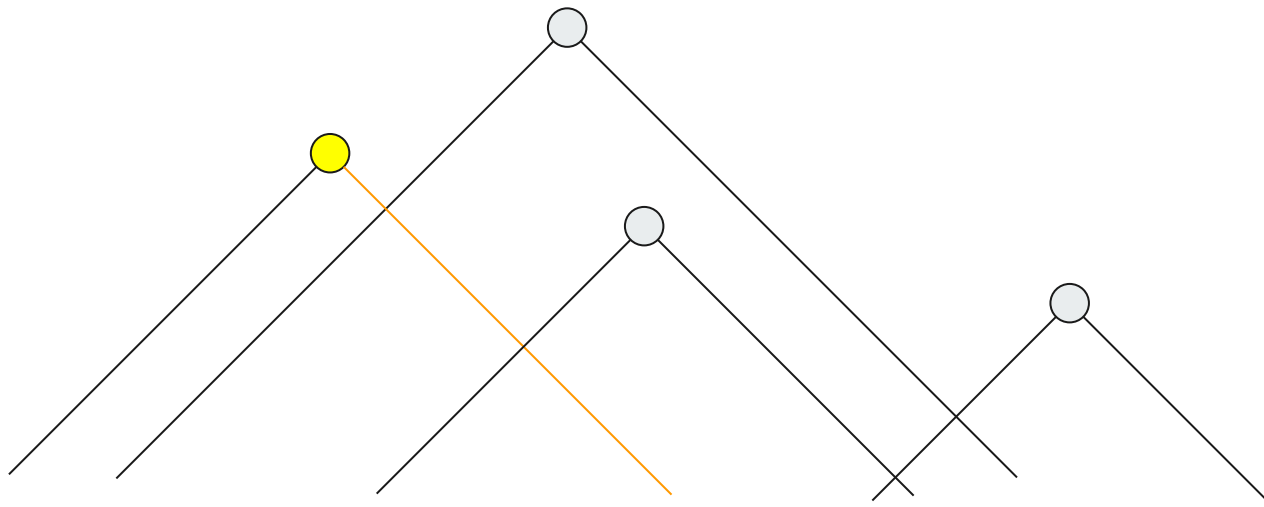
Subtask 7 (20%)

We observe that we can first assume all points are chosen, then pick out points which are not covered by other points.

We pick a point (X_i, Y_i) if $X_i + Y_i$ is higher than all of $X_j + Y_j$ for $i > j$, and $Y_i - X_i$ is higher than all of $Y_j - X_j$ for $i < j$. This can be done with a static prefix max of $X_i + Y_i$ and suffix max of $Y_i - X_i$, for an overall complexity of $O(N)$.

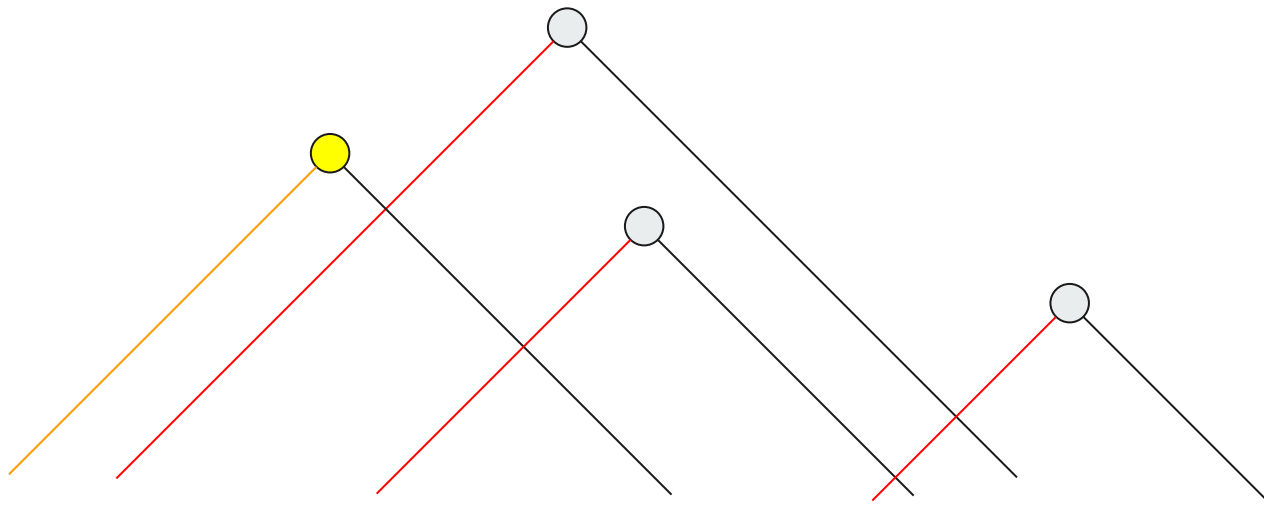


Subtask 7 (20%)



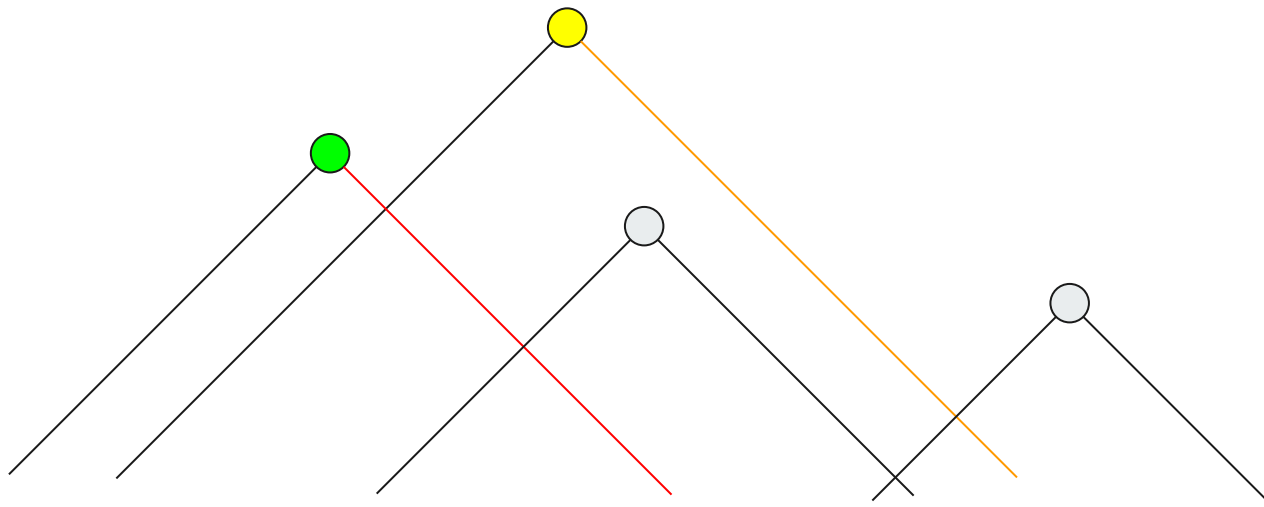


Subtask 7 (20%)



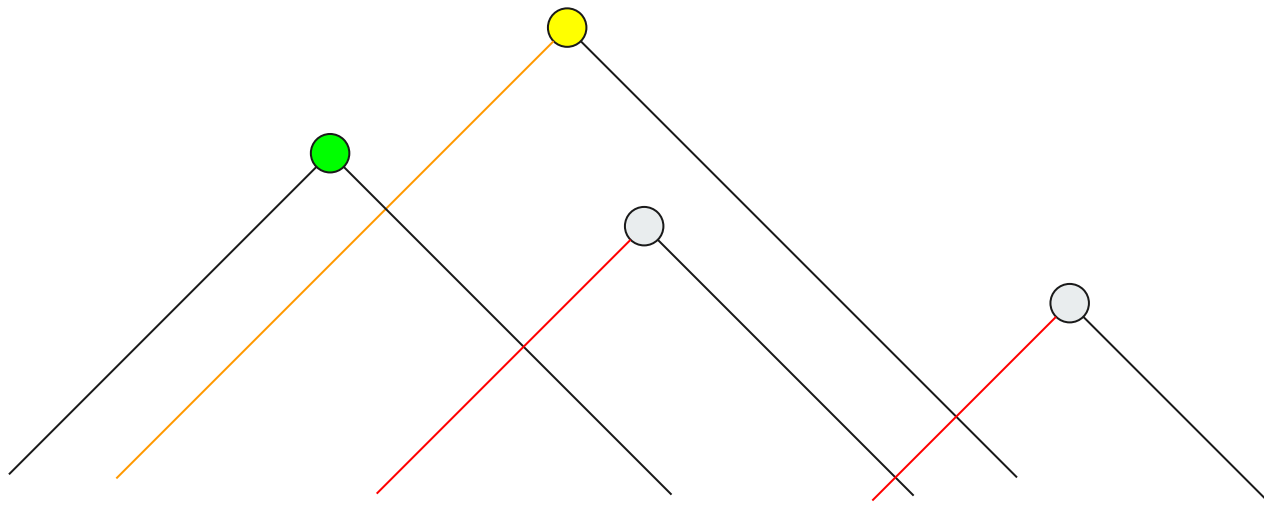


Subtask 7 (20%)



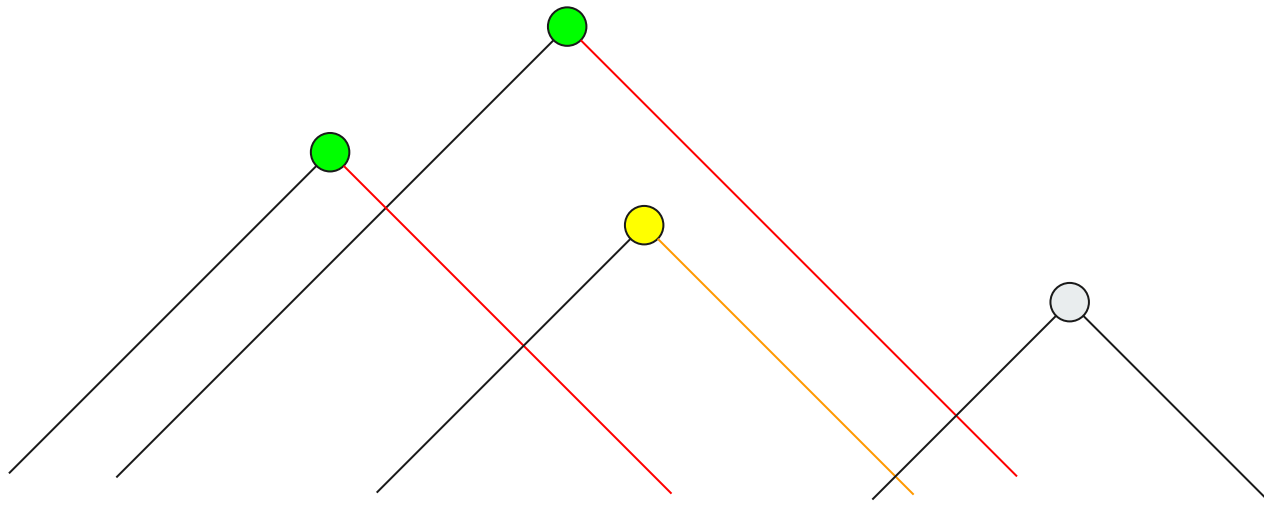


Subtask 7 (20%)



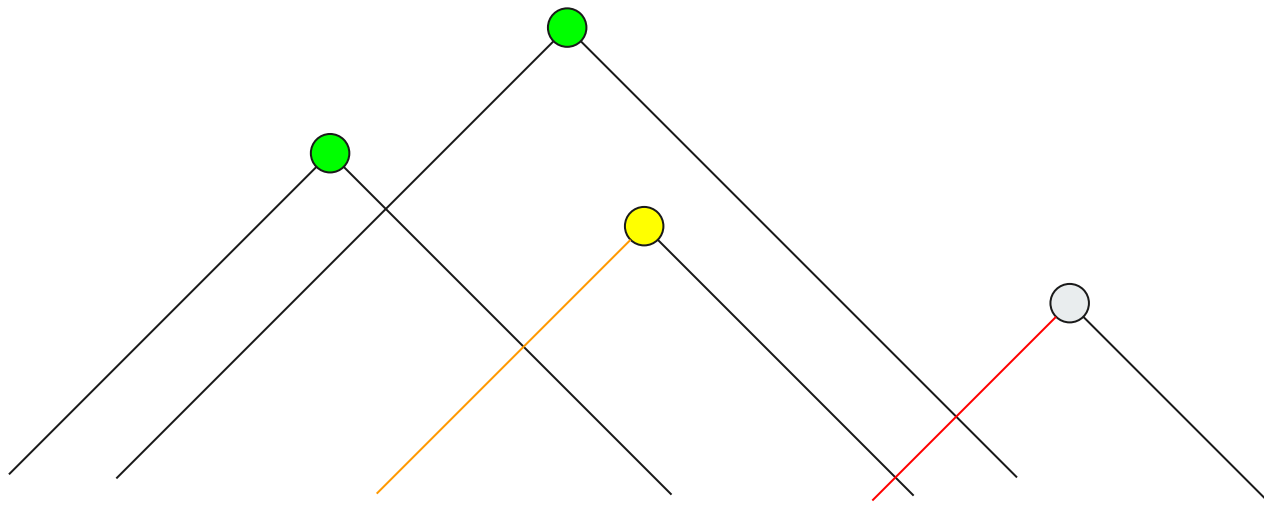


Subtask 7 (20%)



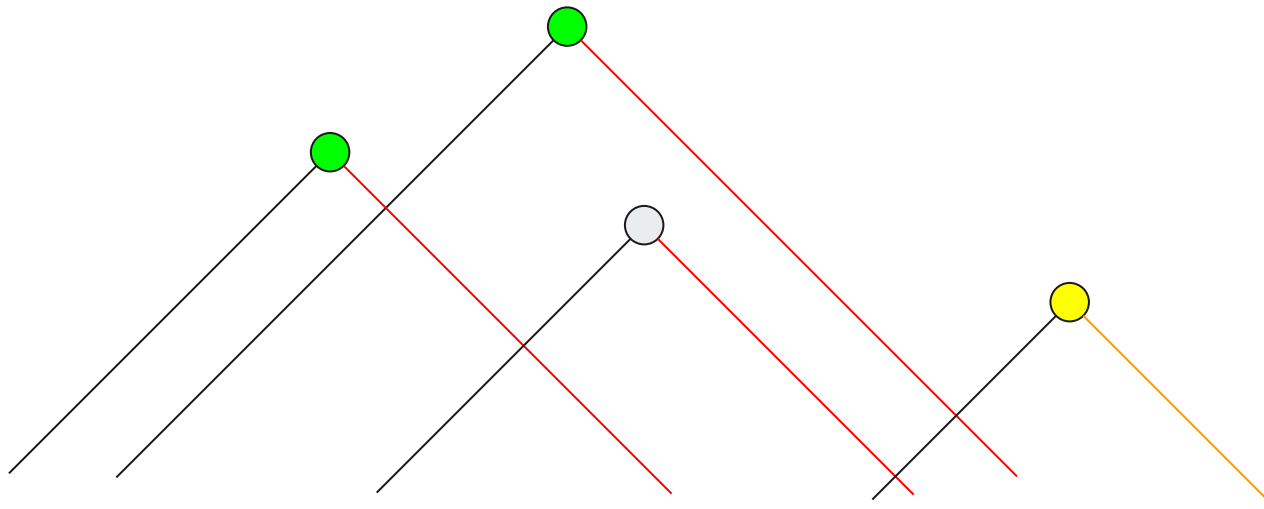


Subtask 7 (20%)



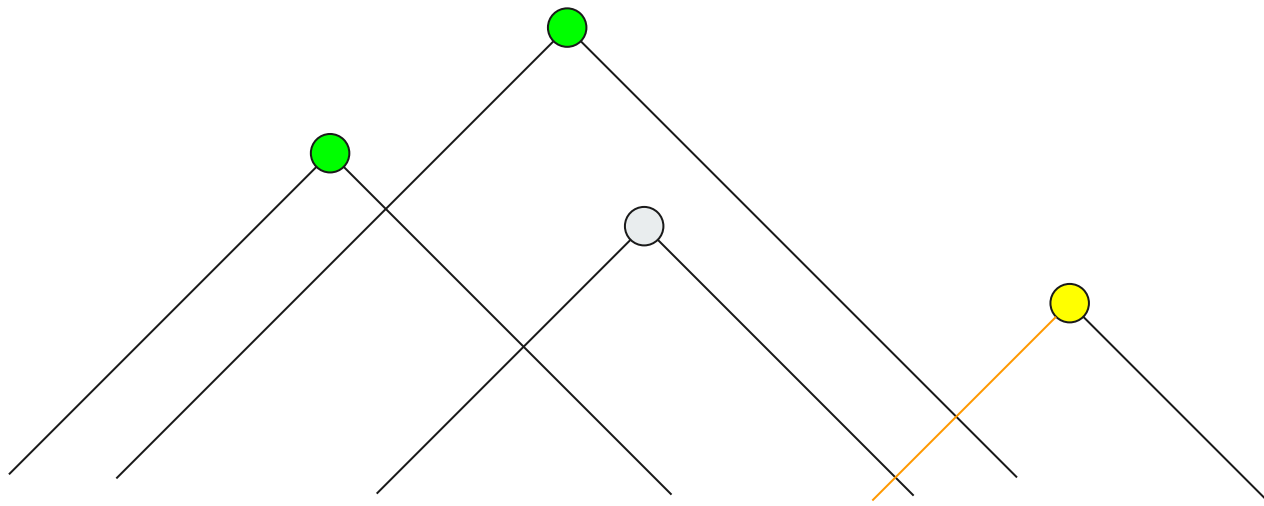


Subtask 7 (20%)





Subtask 7 (20%)





Task 4: citymapping

Problem Author: Pang Wen Yuen



Abridged Problem

You are **not** given a weighted tree of up to $N = 1000$ nodes, and each node degree ≤ 3 .

Use up to $Q = 6500$ queries of **distance between two nodes** to obtain the entire tree.



Subtask 1 (9%)

$Q = 500000$, all weights are 1.

Observation: Two nodes are connected by an edge if and only if their distance is 1.

Solution is to **query all pairs of nodes** and see if they are connected, then report the edge list.



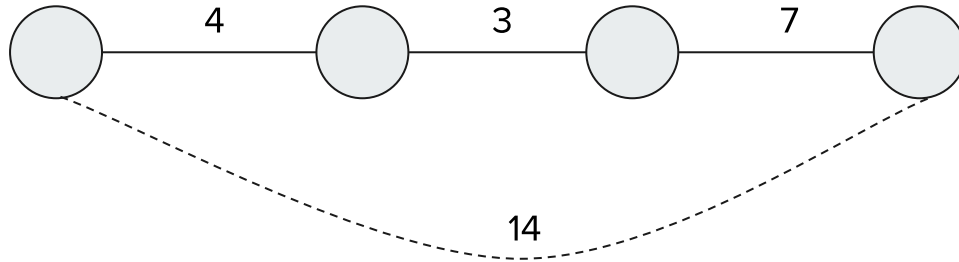
Subtask 2 (16%)

$Q = 500000$.

Unlimited queries!

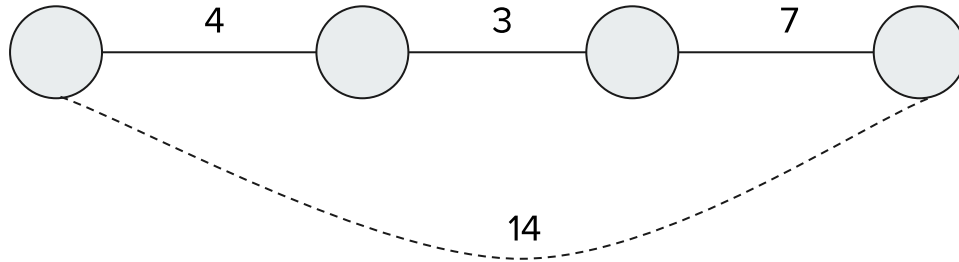
Subtask 2 (16%)

Construct a complete graph where the weight of the edge between two nodes is the distance between them.



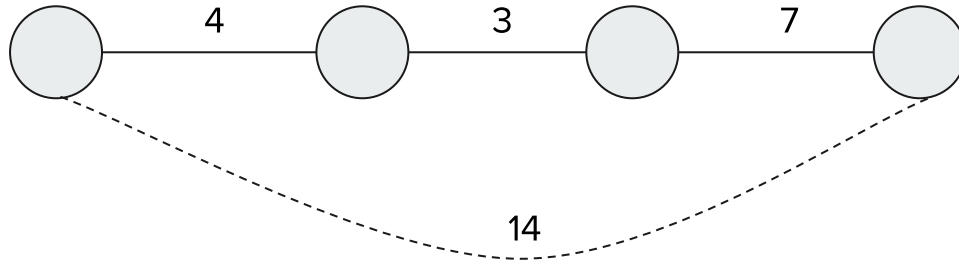
Subtask 2 (16%)

Notice that a “fake edge” will always connect two nodes already connected by edges with weights smaller than it.



Subtask 2 (16%)

To eliminate “fake edges”, we can run Kruskal, and the actual tree will be the Minimum Spanning Tree (MST) of the complete graph.





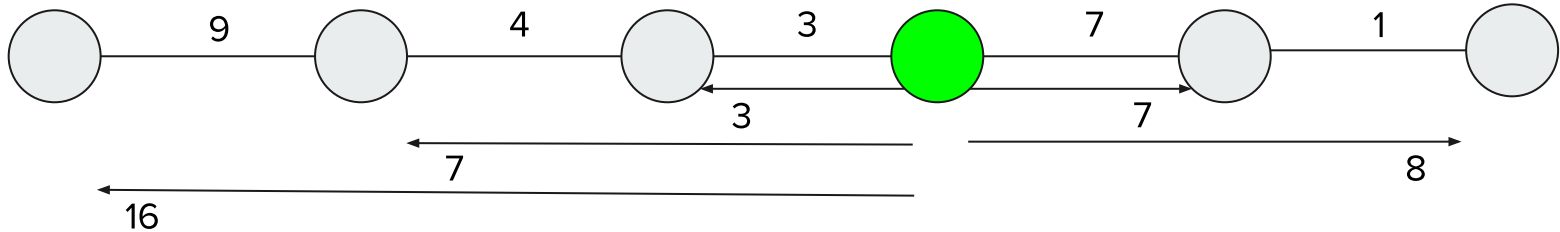
Subtask 3 (13%) + Subtask 4 (19%)

The tree is a **line**, $Q = 12000$.

(There exists solutions that only solve Subtask 3 but we won't go through them.)

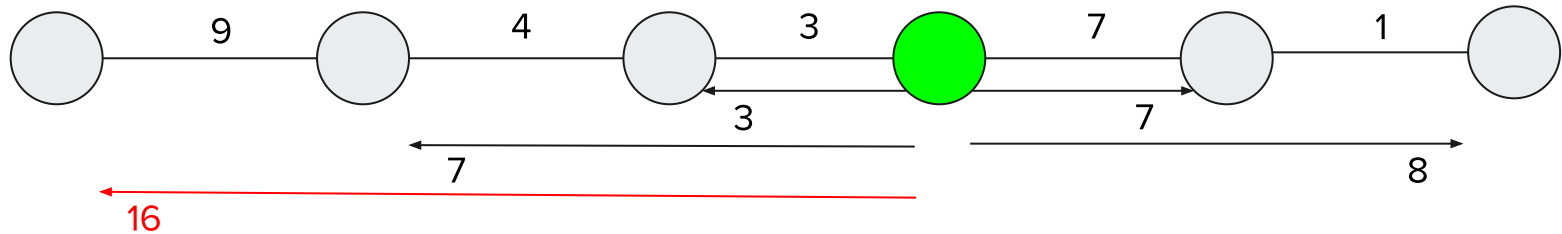
Subtask 3 (13%) + Subtask 4 (19%)

We take a random root node (node 0) and query its distance to all nodes.



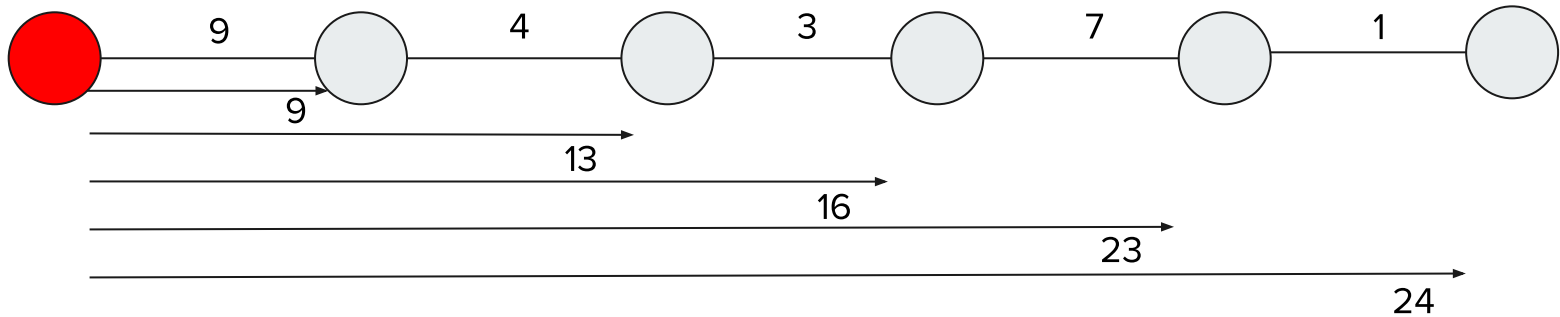
Subtask 3 (13%) + Subtask 4 (19%)

Notice that the node furthest away must lie on one end of the line.



Subtask 3 (13%) + Subtask 4 (19%)

From this, we take the distances and sort them. An edge exists between two nodes which have adjacent distances in sorted order.





Subtask 5 (43%)

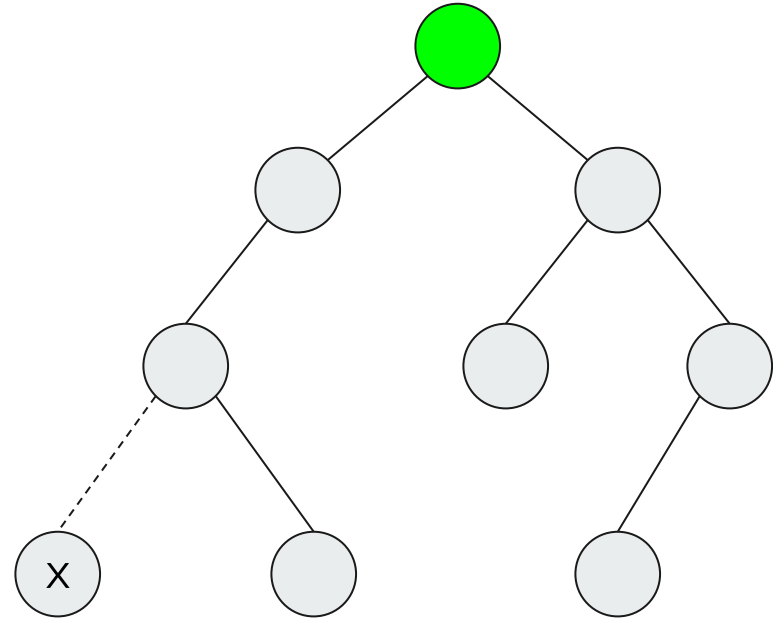
Full problem, $Q = 6500$.

First, we query the distance from a root node (node 0) to all other nodes. Then sort them in ascending order.

Subtask 5 (43%)

Full problem, $Q = 6500$.

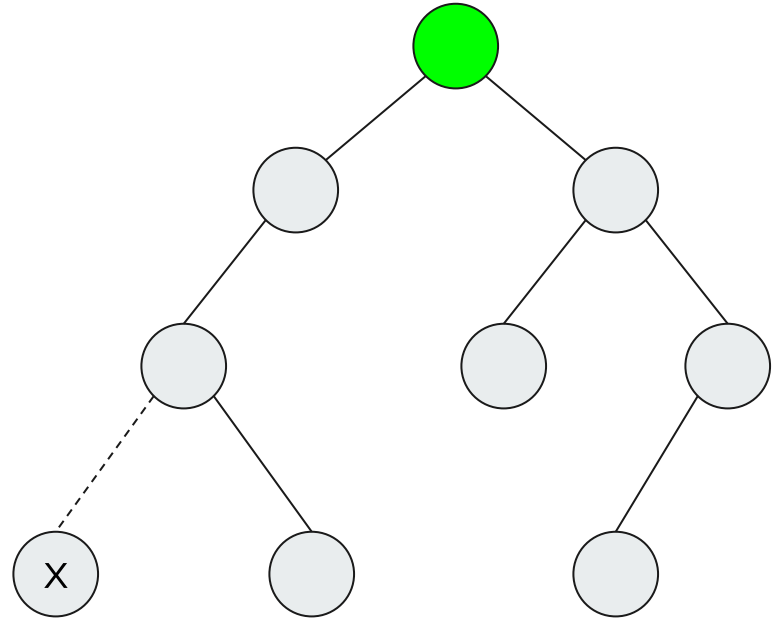
Notice that a path to the root from a certain node X must only pass through nodes that have a **lower distance** to the root than node X .



Subtask 5 (43%)

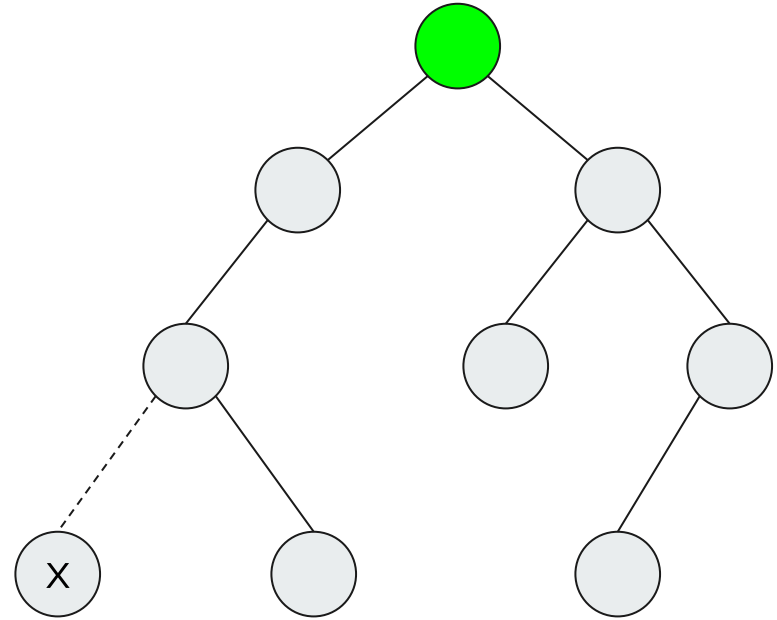
Full problem, $Q = 6500$.

Therefore, we can process the nodes **in sorted order**, and connect them one by one into the “found tree”.



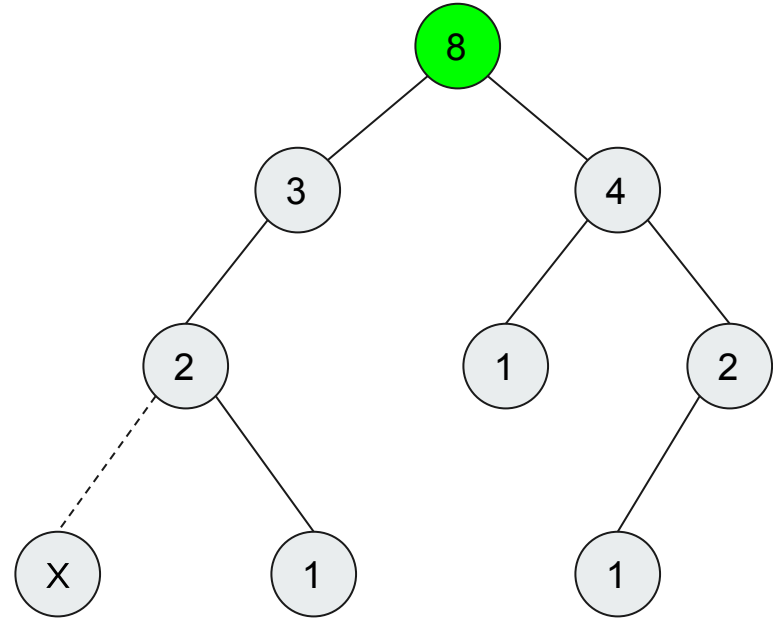
Subtask 5 (43%)

Let's consider the connecting of node X.



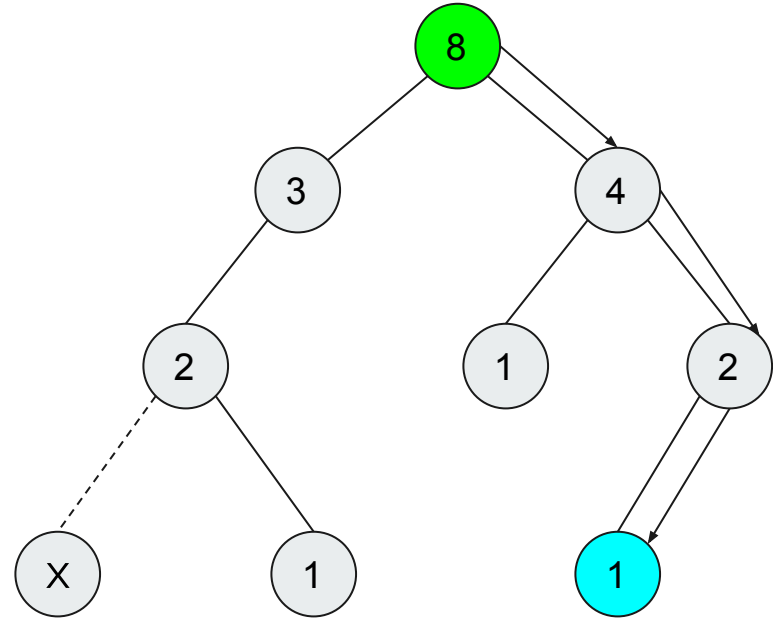
Subtask 5 (43%)

We first count the number of nodes in each subtree in the “found tree”. (can be done with a single Depth-First Search)



Subtask 5 (43%)

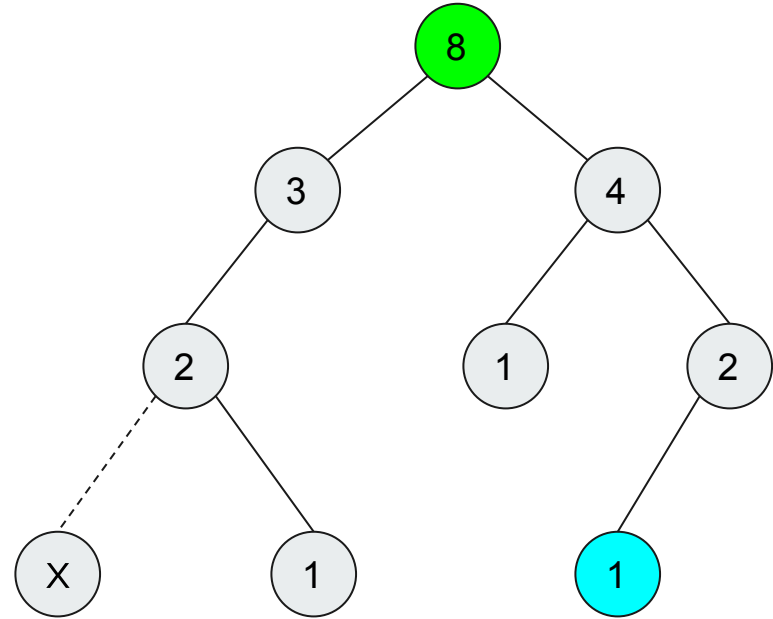
We start from the root. At each point we travel down to the child with the most children. (If tie, pick anyone.)



Subtask 5 (43%)

We query the distance from X to this leaf node.

The answer is **6**.



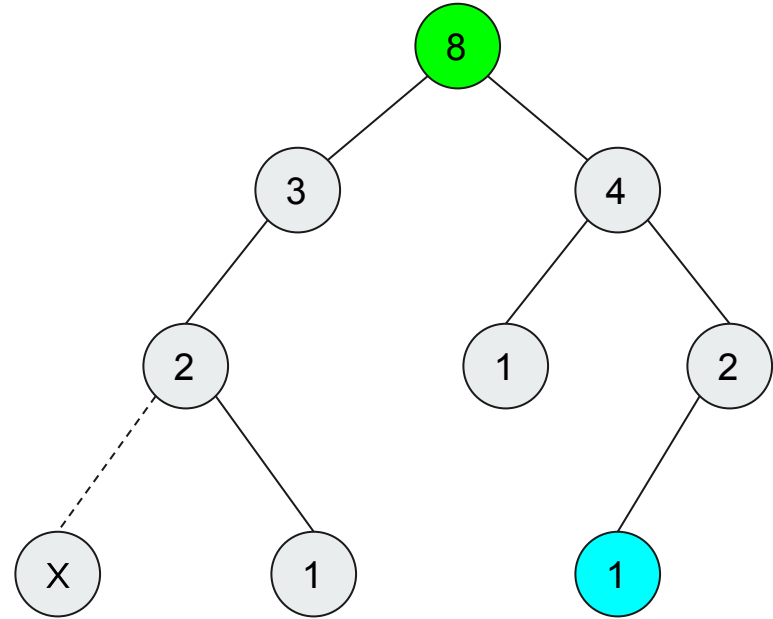
Subtask 5 (43%)

We have three distances now.

$d(R, L)$ $R = \text{root}$

$d(L, X)$ $L = \text{leaf}$

$d(R, X)$ $X = \text{new node}$

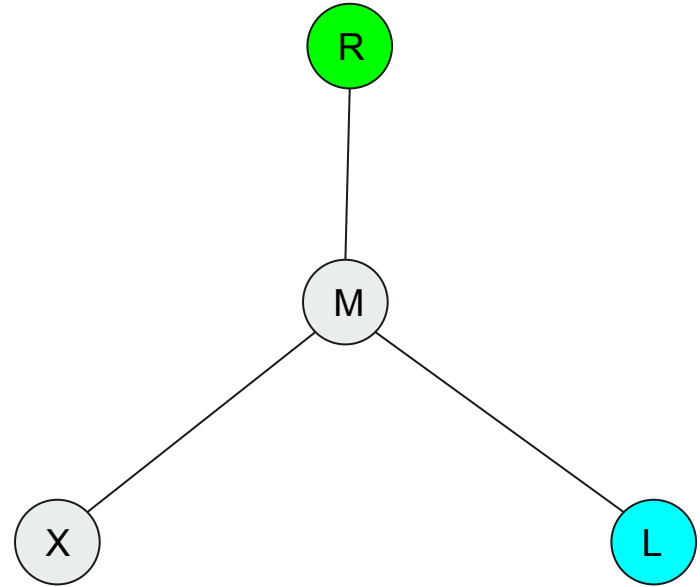


Subtask 5 (43%)

From this we can find

$d(M, R)$, $d(M, X)$ and $d(M, L)$

where **M** is a centre connector node (it can end up being R, L or X)



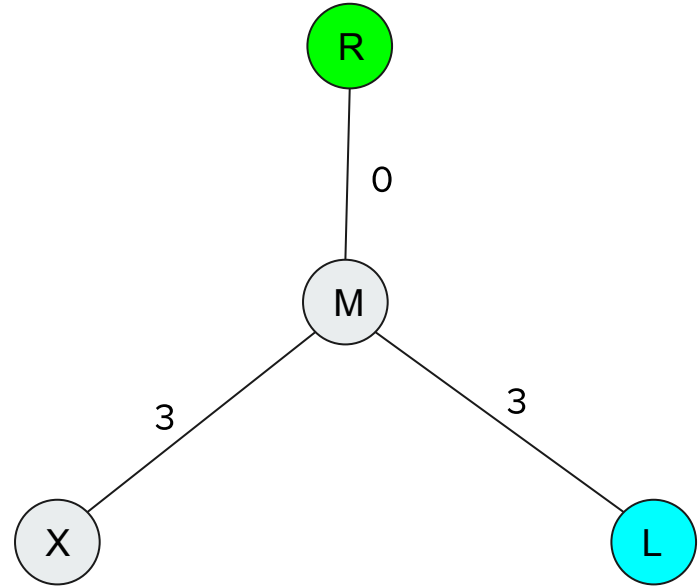
Subtask 5 (43%)

In this case, since:

$$d(R, L) = 3, d(R, X) = 3, d(X, L) = 6.$$

We can deduce:

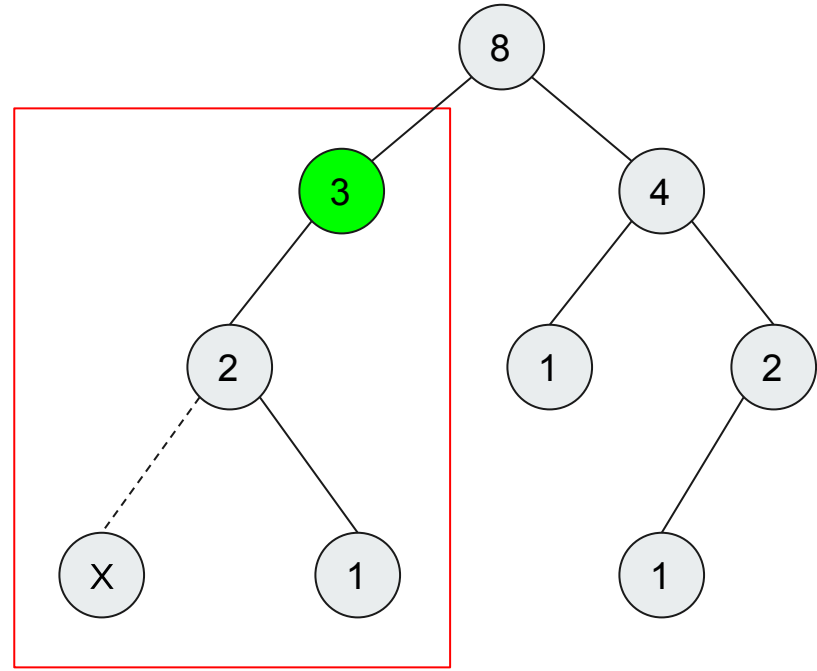
$$d(R, M) = 0, d(M, L) = 3, d(M, X) = 3.$$



Subtask 5 (43%)

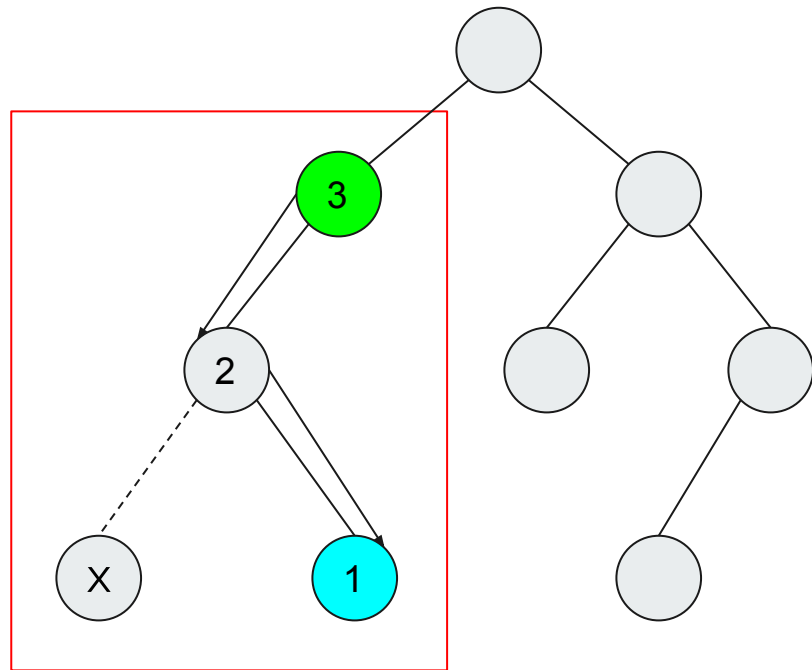
From this we can deduce X lies in this particular subtree, and the distance between the **new root** and X is 2 (since $3 - 1 = 2$).

This is because we know $d(M, R) = 0$, so M (which is the branching point) is at the root.



Subtask 5 (43%)

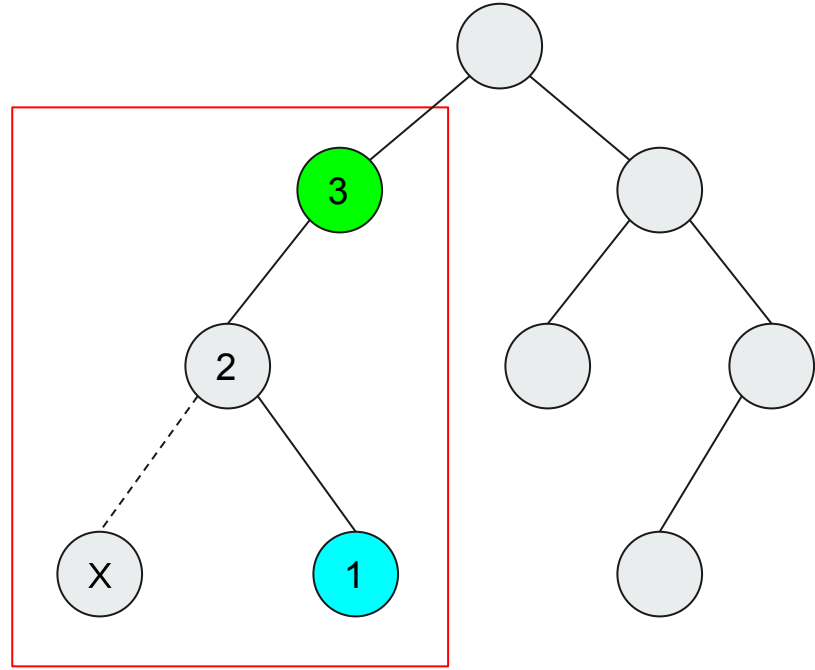
We can repeat this process.



Subtask 5 (43%)

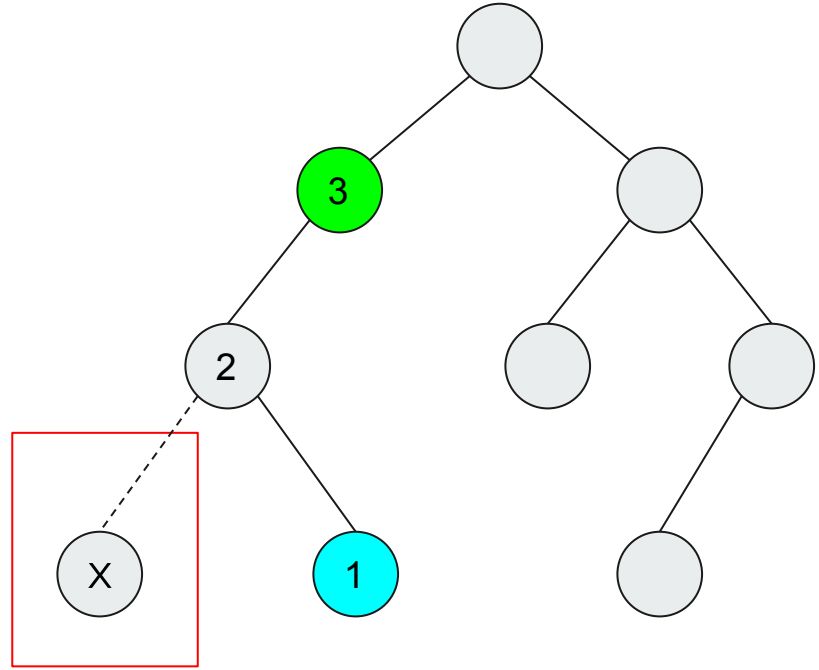
$d(R, X) = 2$ (known), $d(R, L) = 2$ (known), $d(X, L) = 2$ (costs 1 query).

From this we can deduce that $d(R, M) = d(M, X) = d(M, L) = 1$.



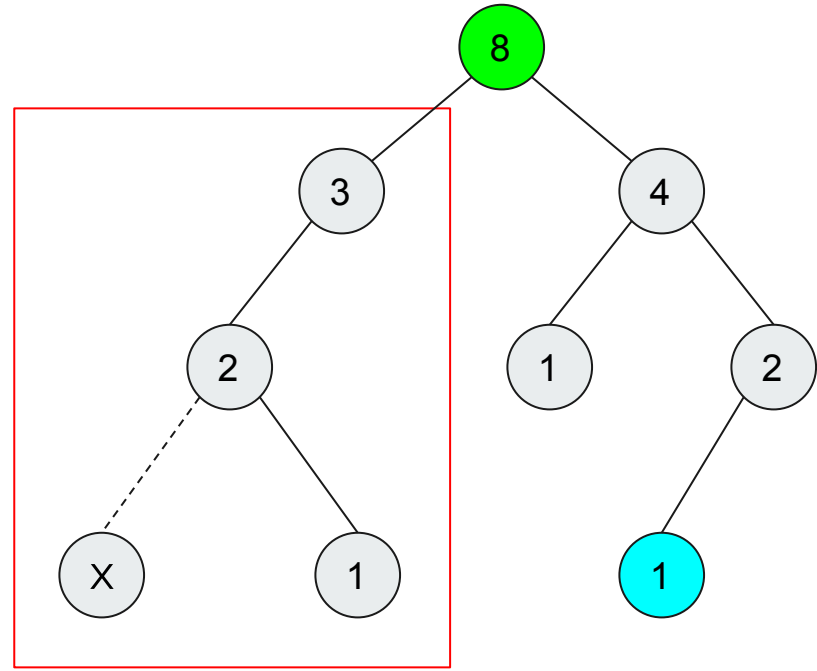
Subtask 5 (43%)

Now we can narrow down X to a single location.



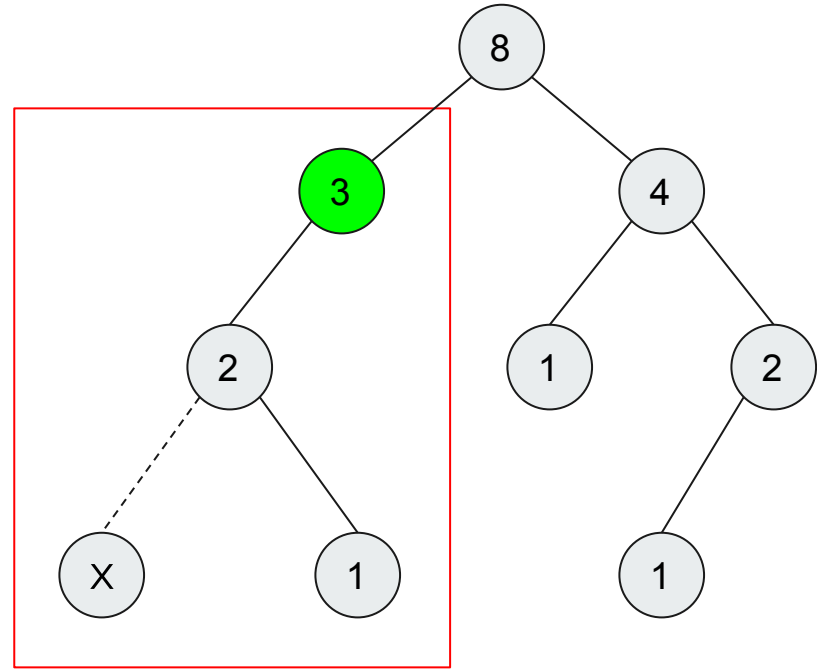
Subtask 5 (43%)

Each query will narrow down the search space for X to a single branch on the root \rightarrow leaf path (since we can find out $d(R, M)$). This is repeated until X is found exactly.



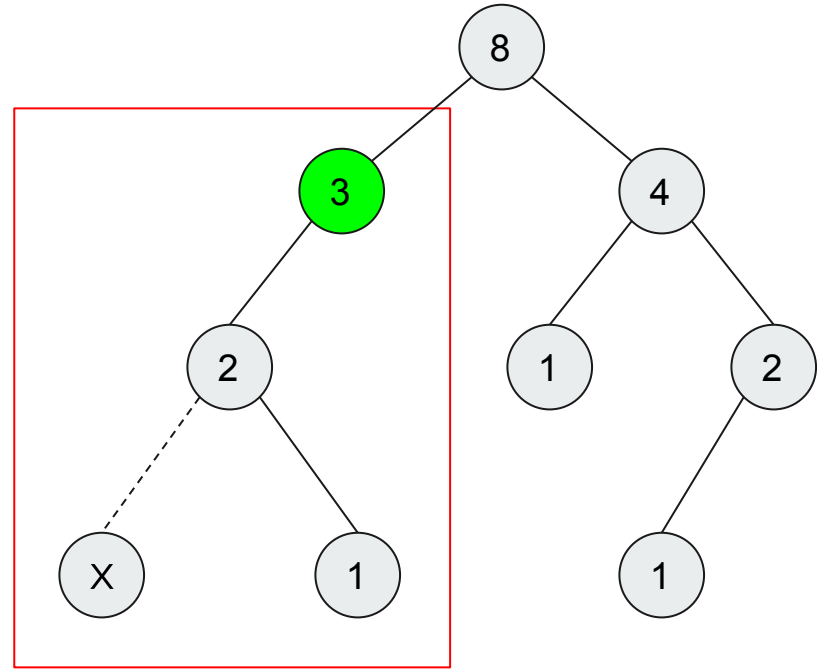
Subtask 5 (43%)

Since each time we travel down to the node with the **most nodes in its subtree**, we will prune out **at least half of the nodes** in every iteration (except for when the root has 3 children, then one-third).



Subtask 5 (43%)

Therefore, the overall complexity of queries is $N \log N$ with a low constant, which passes under the $Q = 6500$ limit.





Task 5: safety

Problem Author: Bernard Teo Zhi Yi



Abridged Problem

Given an array of N integers, find the minimum number of increment/decrement operations (on elements in the array) required such that all pairs of adjacent elements differ by no larger than H .



Abridged Problem

N = size of array

H = required maximum difference between adjacent elements

K = maximum possible height of any element

N = size of array

H = required maximum difference

K = maximum possible height



General Observation #1

Given two sequences **A** and **B**, both of length **N**, the minimum number of steps needed to get from **A** to **B** is

$$\sum_{i=1}^N |A[i] - B[i]|$$

where **A**[**i**] is the **i**th element of **A**, and **B**[**i**] is the **i**th element of **B**.

N = size of array

H = required maximum difference

K = maximum possible height



General Observation #2

If $H > K$, we can just set H to K .

Largest possible difference between two elements
= (largest possible height) – (smallest possible height)
 $\geq K - 0$
 $= K$

N = size of array

H = required maximum difference

K = maximum possible height



Subtask 1 (3%)

$1 \leq N \leq 10, 0 \leq H \leq 4, K = 4$

For each candidate sequence of length N (there are $(K+1)^N$ of them):

If the sequence is safe, find the number of steps required

Output the minimum number of steps required amongst all the safe sequences.

Time complexity: $O(K^N N)$

N = size of array

H = required maximum difference

K = maximum possible height



Subtask 2 (4%)

$$1 \leq N \leq 14, 0 \leq H \leq 1, K = 4$$

$H \leq 1 \Rightarrow$ reduced branching factor

If $S[i]$ is determined to be X , then $S[i+1]$ can only be $X-1$, X , or $X+1$

It is faster to do a DFS that will only look through *safe* sequences



Subtask 2 (4%)

N = size of array

H = required maximum difference

K = maximum possible height

$$1 \leq N \leq 14, 0 \leq H \leq 1, K = 4$$

For each possible $S[1]$ in between 0 and K (inclusive):

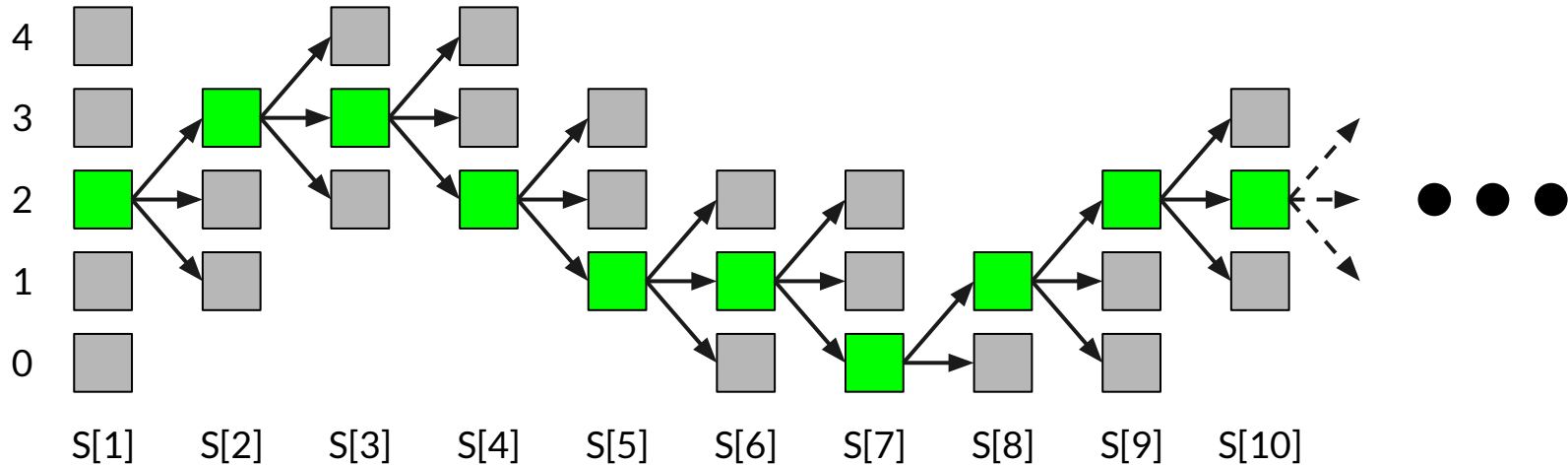
DFS to look through all possible *safe* sequences,
finding the minimum number of steps required

N = size of array

H = required maximum difference

K = maximum possible height

Subtask 2 (4%)



Time complexity: $O((2H)^{N-1}NK)$

N = size of array

H = required maximum difference

K = maximum possible height



Subtask 3 (9%)

$$1 \leq N \leq 10, 0 \leq H \leq 2, K = 10^9$$

Constraint for **K** is increased to 10^9 , but **N** and **H** remains small

⇒ Branching factor is still small, but how to choose starting points?

N = size of array

H = required maximum difference

K = maximum possible height



Subtask 3 (9%)

$$1 \leq N \leq 10, 0 \leq H \leq 2, K = 10^9$$

It can be shown that at least one stack need not be modified!

(Try proving by contradiction. This is left as an exercise for the reader.)

Time complexity: $O((2H)^{N-1}N^2)$

N = size of array

H = required maximum difference

K = maximum possible height



Subtask 4 (5%)

$$1 \leq N \leq 2 \times 10^5, H = 0, K = 10^9$$

The constraint $H = 0$ means that all stacks must have the same height after modification

⇒ the optimal height is the median height amongst all stacks

Time complexity: $O(N \log N)$



Subtask 5 (6%)

N = size of array

H = required maximum difference

K = maximum possible height

$1 \leq N \leq 500$, $H = 10^9$, $K = 400$

Dynamic programming (on **N** and **K** dimensions)

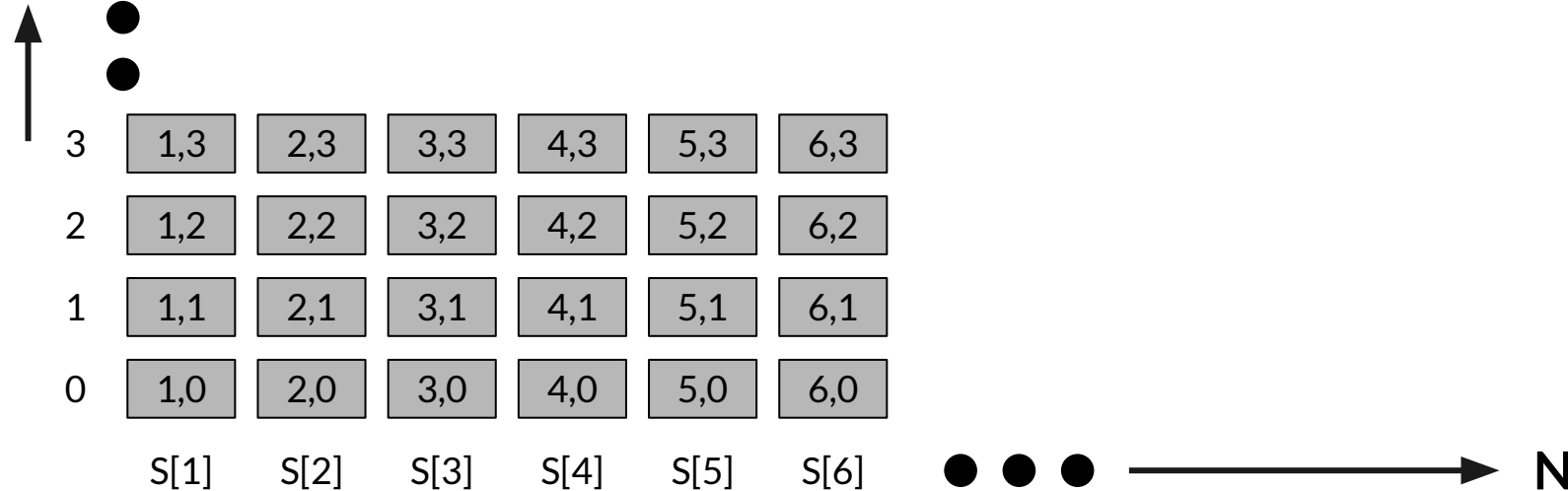
$\text{best}(n, k)$ = minimum number of operations required from stack 1 to stack n such that the n^{th} stack has height exactly k

N = size of array

H = required maximum difference

K = maximum possible height

Subtask 5 (6%)



$\text{best}(n, k)$ = minimum number of operations required from stack 1 to stack n such that the n^{th} stack has height exactly k



Subtask 5 (6%)

N = size of array

H = required maximum difference

K = maximum possible height

$1 \leq N \leq 500$, $H = 10^9$, $K = 400$

$\text{best}(n, k)$ depends only $\text{best}(n-1, ?)$ (i.e. the previous column)

\Rightarrow there is no cyclic dependency; dynamic programming is possible

N = size of array

H = required maximum difference

K = maximum possible height



Subtask 5 (6%)

$1 \leq N \leq 500, H = 10^9, K = 400$

$$\text{best}(n, k) = \min_{k' = \max(k-H, 0)}^{\min(k+H, K)} \{ \text{best}(n-1, k') \} + |k - S[n]|$$

State: $O(NK)$

Transition: $O(H)$

Time complexity: $O(NHK)$

N = size of array

H = required maximum difference

K = maximum possible height

Subtask 6 (11%)

$1 \leq N \leq 500, H = 10^9, K = 5000$

$$\text{best}(n, k) = \min_{k' = \max(k-H, 0)}^{\min(k+H, K)} \{ \text{best}(n-1, k') \} + |k - S[n]|$$

Use a balanced binary search tree to store $\text{best}(n-1, ?)$ within valid range – C++ `std::multiset` or Java `TreeSet` (careful with duplicates!)

Time complexity: $O(NK \log H)$

N = size of array

H = required maximum difference

K = maximum possible height



Subtask 7 (11%)

$1 \leq N \leq 5000, H = 10^9, K = 5000$

In the previous subtask, we have reduced the transition time from $O(H)$ to $O(\log H)$. We can further reduce transition time to amortized $O(1)$ using a double-ended queue.

The sliding range min query problem is well-known. (Proof omitted.)

Time complexity: $O(NK)$

N = size of array

H = required maximum difference

K = maximum possible height



Subtask 8 (22%)

$1 \leq N \leq 5000$, $H = 10^9$, $K = 10^9$

How to reduce the time taken beyond the DP state complexity?

Can we avoid storing $\text{best}(n, k)$ as is?

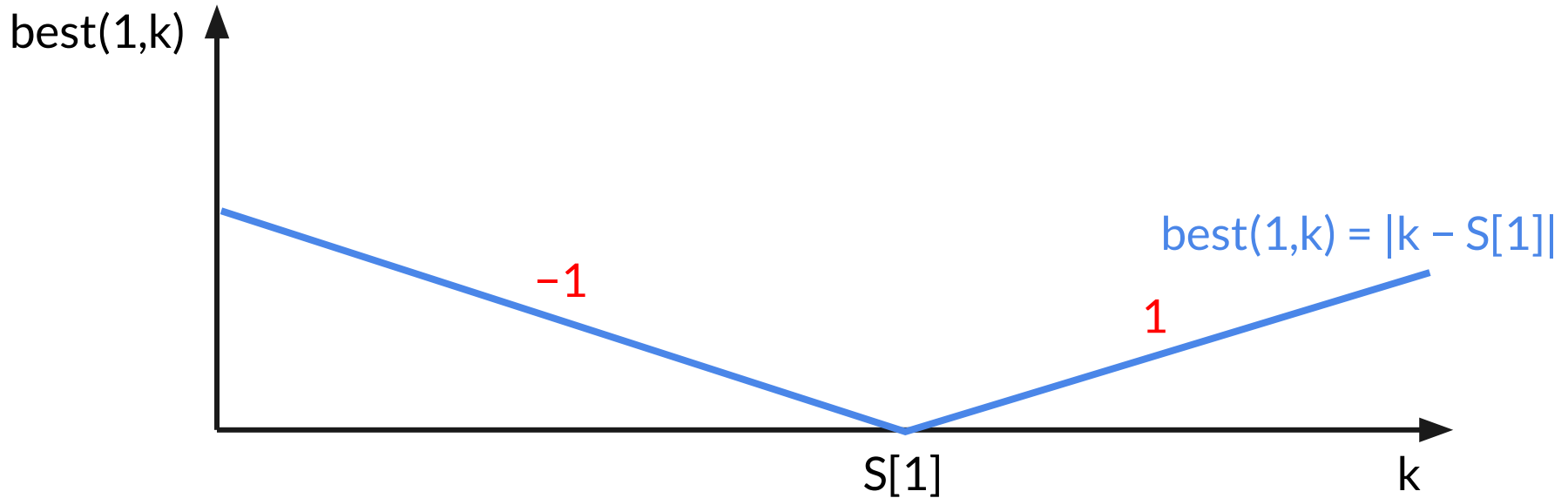
Can we quickly obtain the whole column $\text{best}(n, ?)$ from the previous column $\text{best}(n-1, ?)$?

Subtask 8 (22%)

N = size of array

H = required maximum difference

K = maximum possible height

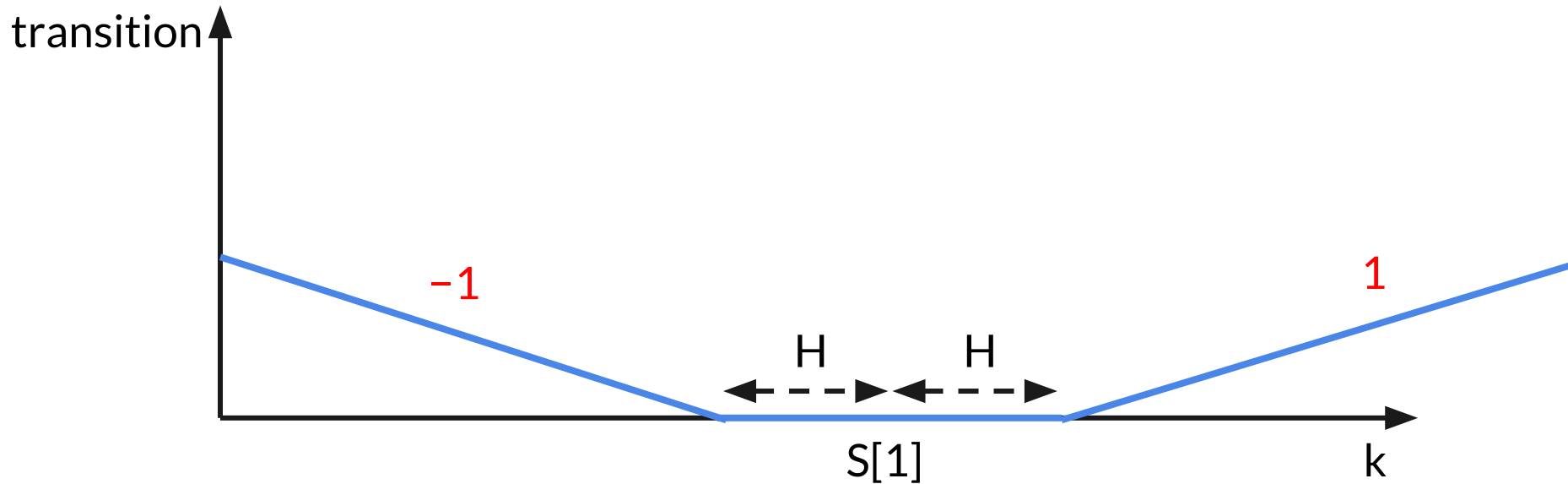


Subtask 8 (22%)

N = size of array

H = required maximum difference

K = maximum possible height

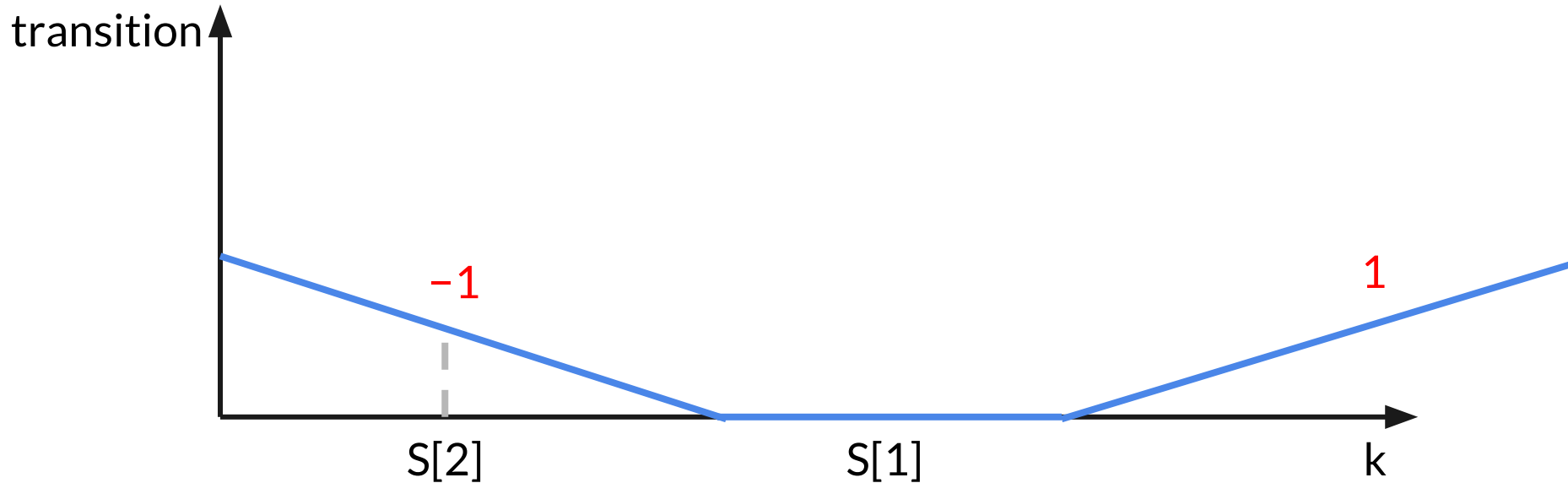


Subtask 8 (22%)

N = size of array

H = required maximum difference

K = maximum possible height

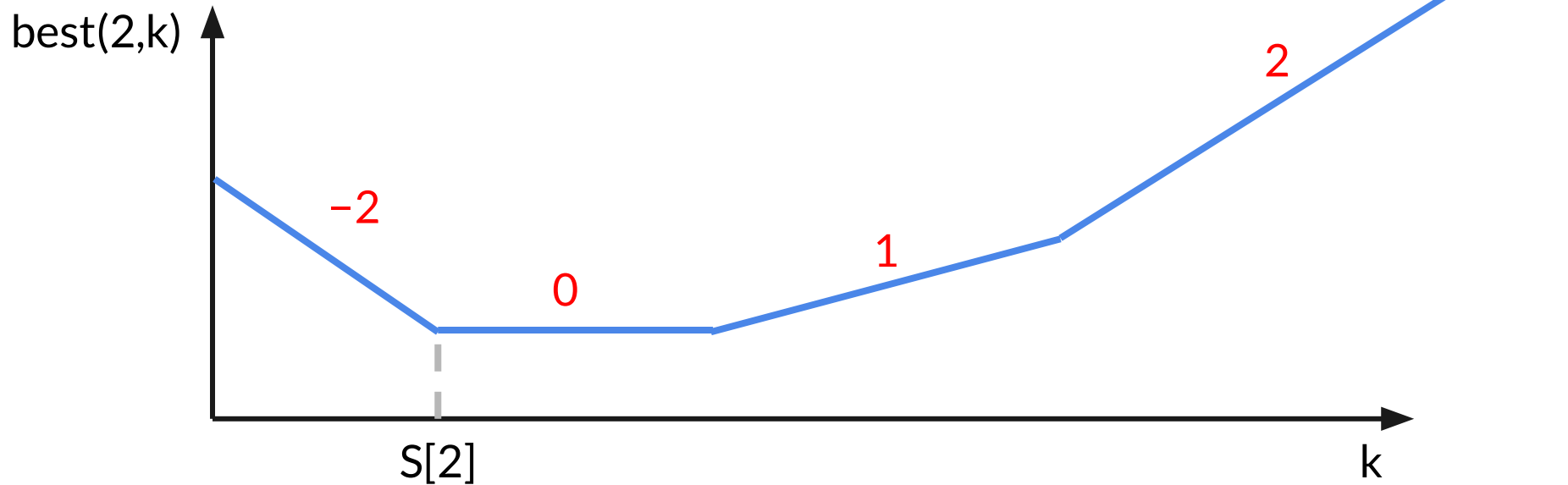


Subtask 8 (22%)

N = size of array

H = required maximum difference

K = maximum possible height

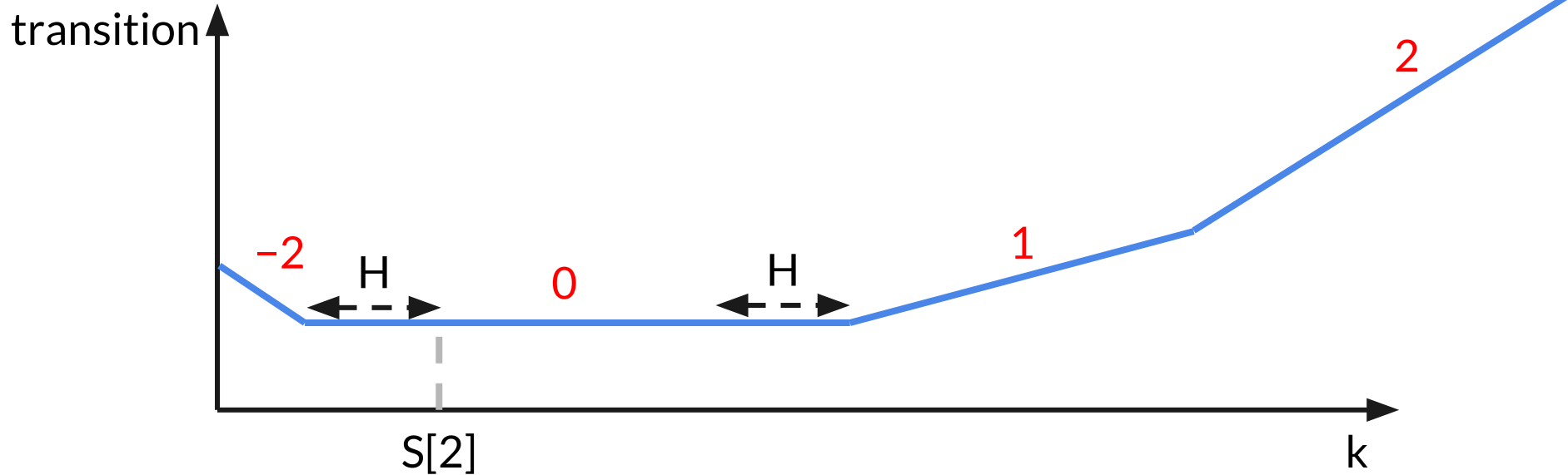


Subtask 8 (22%)

N = size of array

H = required maximum difference

K = maximum possible height

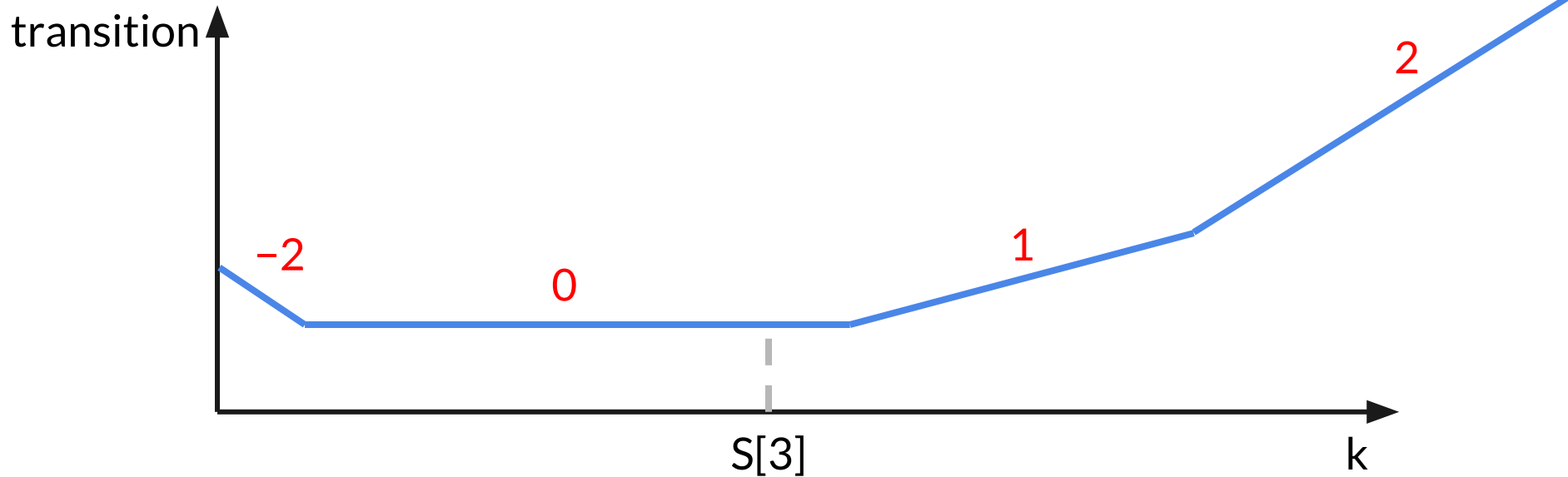


Subtask 8 (22%)

N = size of array

H = required maximum difference

K = maximum possible height



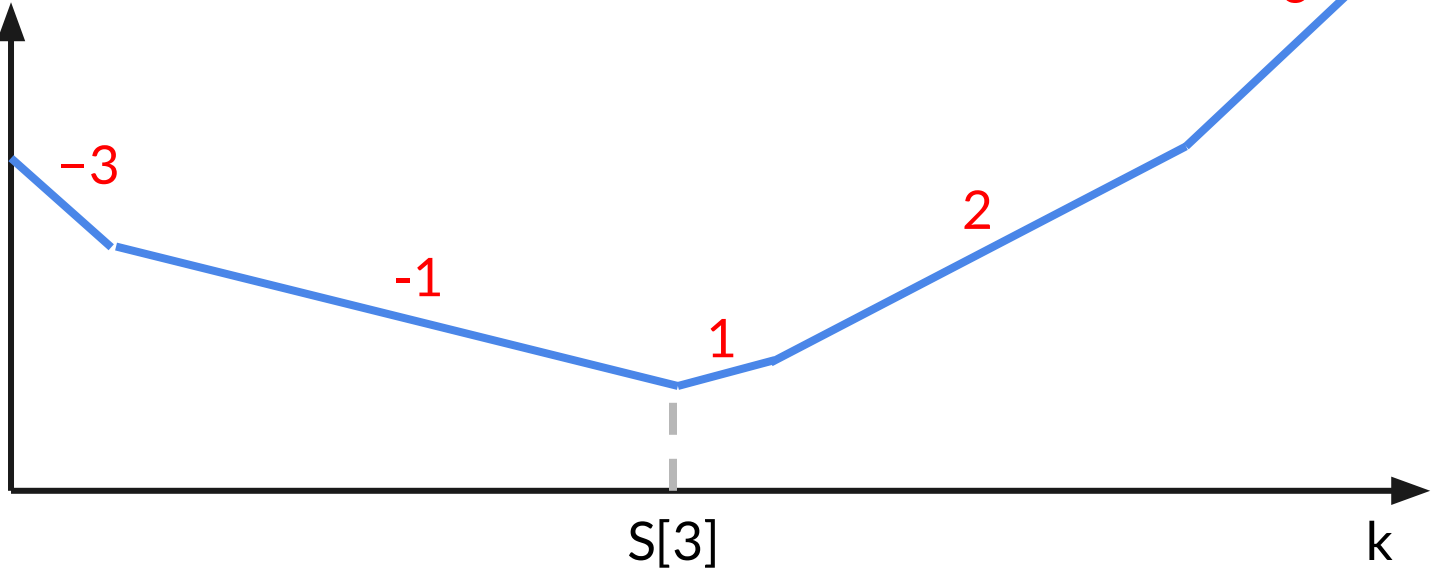
Subtask 8 (22%)

N = size of array

H = required maximum difference

K = maximum possible height

$\text{best}(3,k)$



N = size of array

H = required maximum difference

K = maximum possible height



Subtask 8 (22%)

Notice that the number of points where the gradient changes is at most $2N$

⇒ We can adjust these critical points in $O(N)$

⇒ Obtaining the whole column $\text{best}(n, ?)$ from the previous column $\text{best}(n-1, ?)$ can be done in $O(N)$

Time complexity: $O(N^2)$

N = size of array

H = required maximum difference

K = maximum possible height



Subtask 9 (29%)

$$1 \leq N \leq 2 \times 10^5, H = 10^9, K = 10^9$$

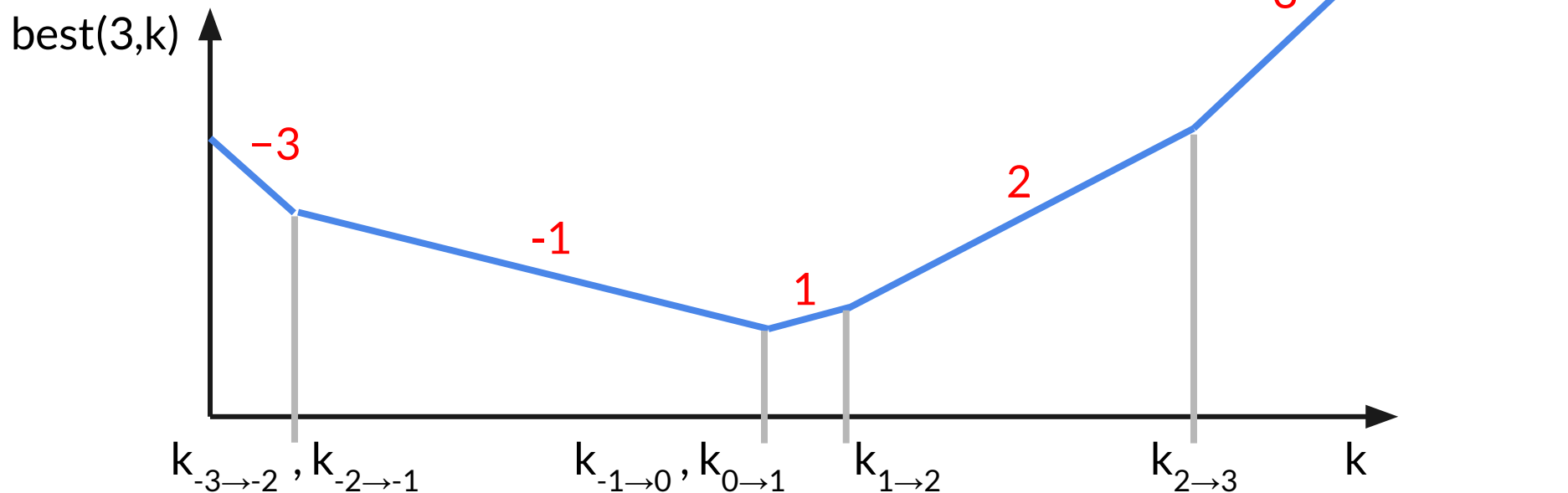
Can we obtain the whole column $\text{best}(n, ?)$ from the previous column $\text{best}(n-1, ?)$ even more efficiently?

Subtask 9 (29%)

N = size of array

H = required maximum difference

K = maximum possible height



N = size of array

H = required maximum difference

K = maximum possible height



Subtask 9 (29%)

Store the k -values of all the critical points in pair of priority queues or multisets (see task analysis for more details)

⇒ Obtaining the whole column $\text{best}(n, ?)$ from the previous column $\text{best}(n-1, ?)$ can be done in $O(\log N)$

Time complexity: $O(N \log N)$



End