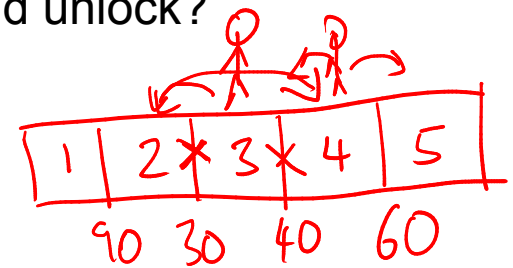
Locked Doors

distinct

There are **N** rooms in a row, connected by **N** - 1 doors. The **i**-th door has a difficulty of D_i. Bangles starts in one room and starts opening doors one at a time. Of the two locked rooms she could go to next, she picks the one with the lower difficulty.

$Q, N \leq 10^5$

Answer **Q** queries of the form: If Bangles starts in room **s**, what is the **k**-th room she would unlock?



$O(NQ)$

$s=3 \quad k=4$

$s=4 \quad k=?$

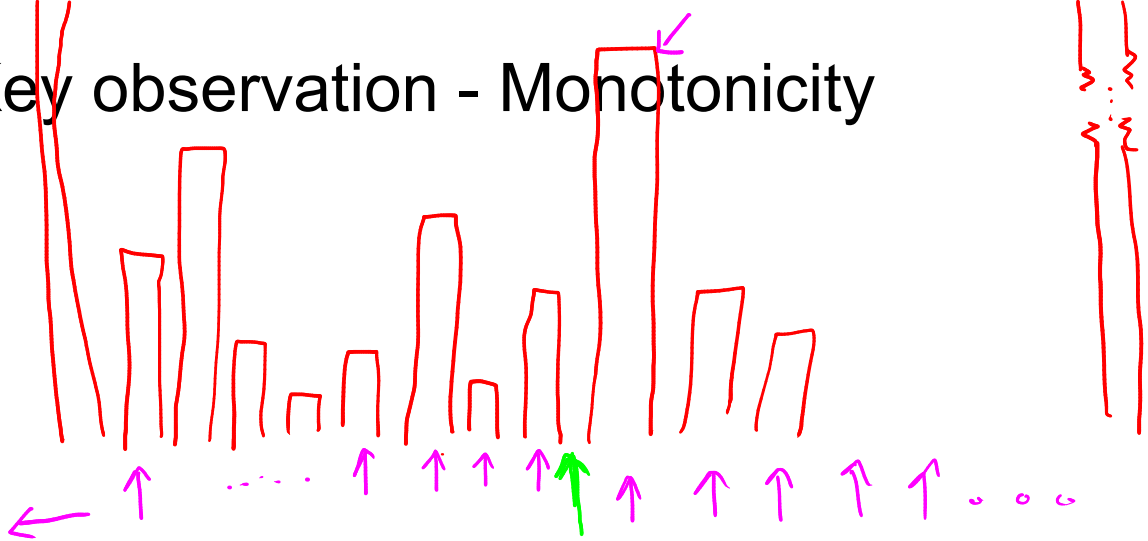
$O(\text{Rooms} \cdot \min \; NQ \; N^2)$

5

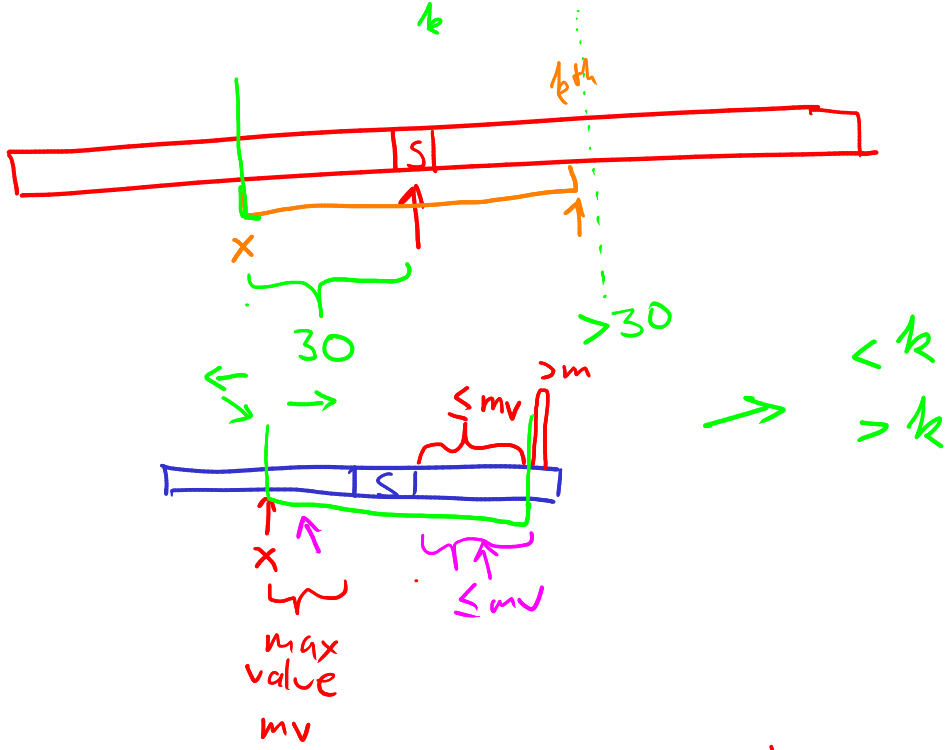
3, 2, 4, 5, 1

4 3 2 5, 1 k

Key observation - Monotonicity



Calculating the answer “quickly”



binary search $\log(N)$

+RMQ $\log(N)^{O(1)}$

+ binary search $\log N$

RMQ $\log N$ $O(1)$

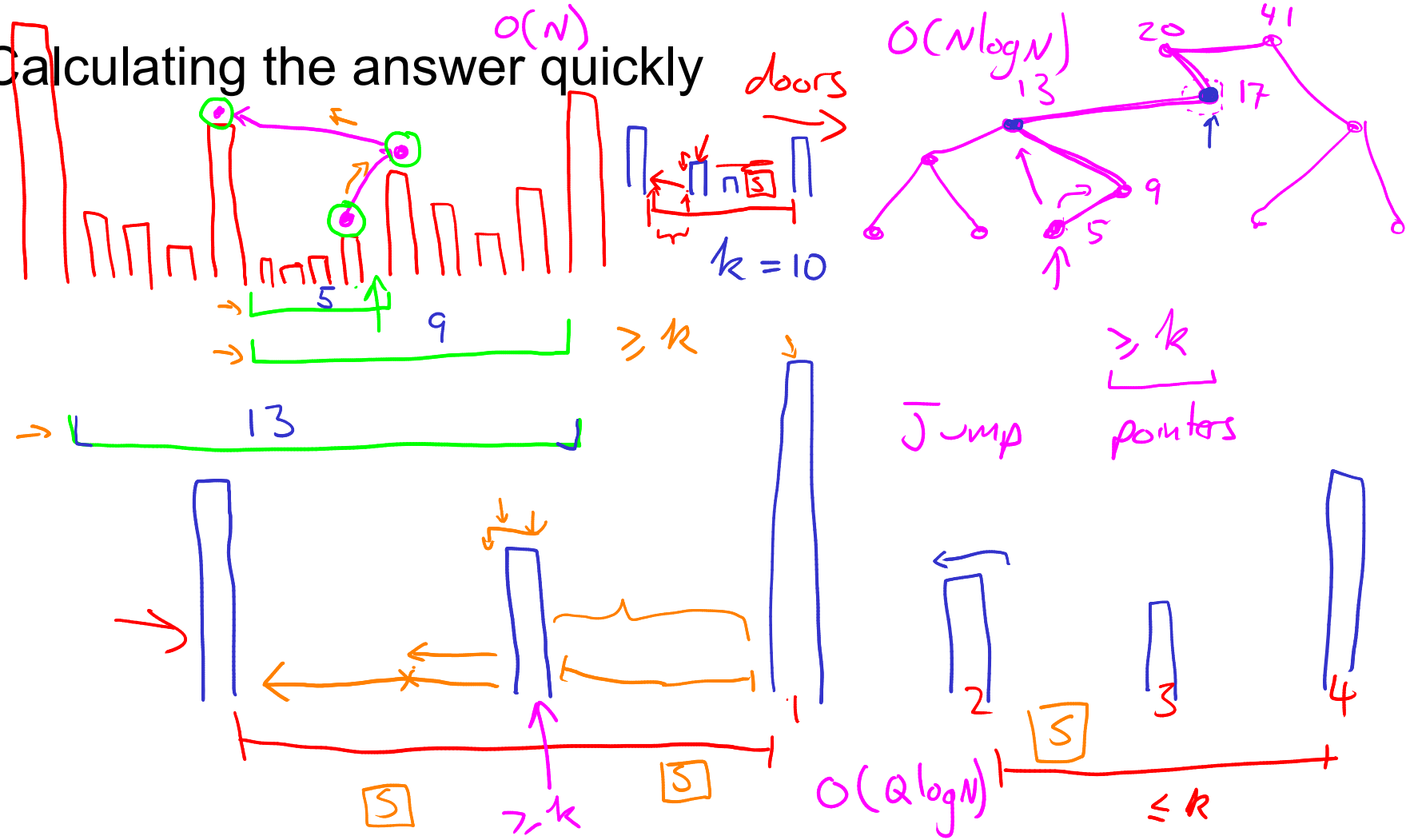
$\log^3 N$

$\log^2 N$

$N \log^2 N$

Reduction to a cartesian tree

Calculating the answer quickly



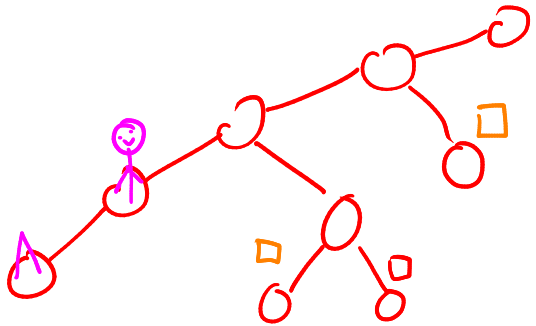
Dog Catcher

Dani starts at vertex 1 (a leaf) in a tree of **N** vertices. There are also **D** dogs in the tree. Dani and the dogs take turns:

- Dani moves to an adjacent vertex. If there are any dogs there, she catches them.
- Then, all the dogs move to any vertex they can reach, without moving through Dani's vertex.

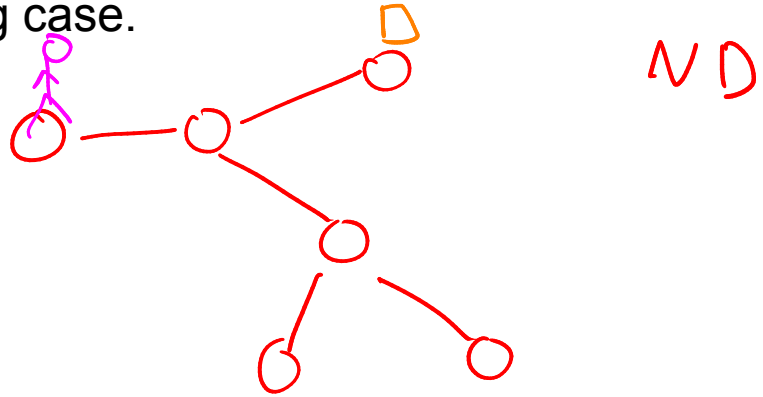
Dani wants to minimise the number of turns it takes to catch all the docs, while the dogs want to maximise it. How many turns does it take?

N, **D** ≤ 50



Claim 1: The game is finite

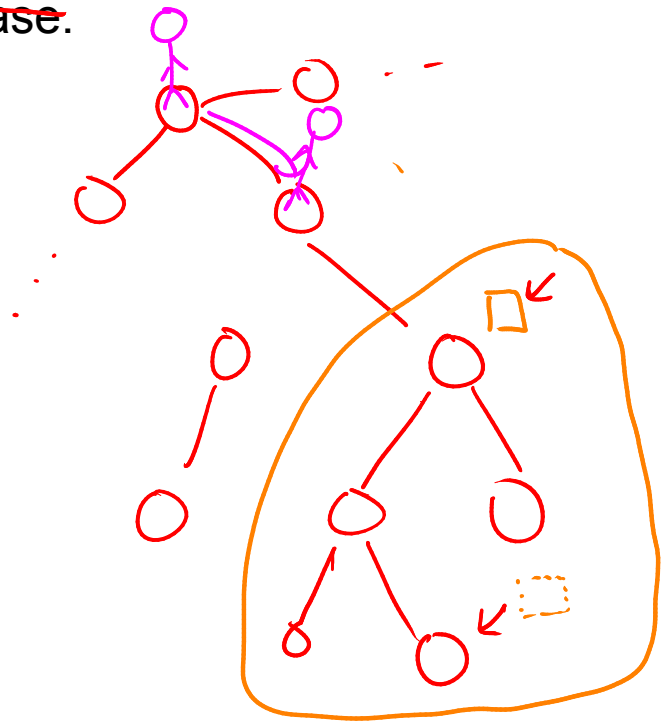
Consider the 1 dog case.



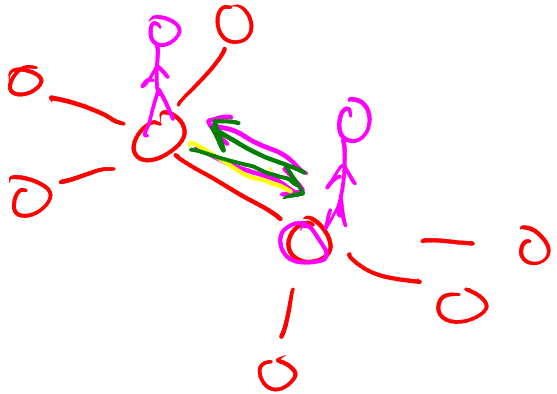
N D

Claim 2: The dogs hide in leaves

~~Consider the 1 dog case~~.

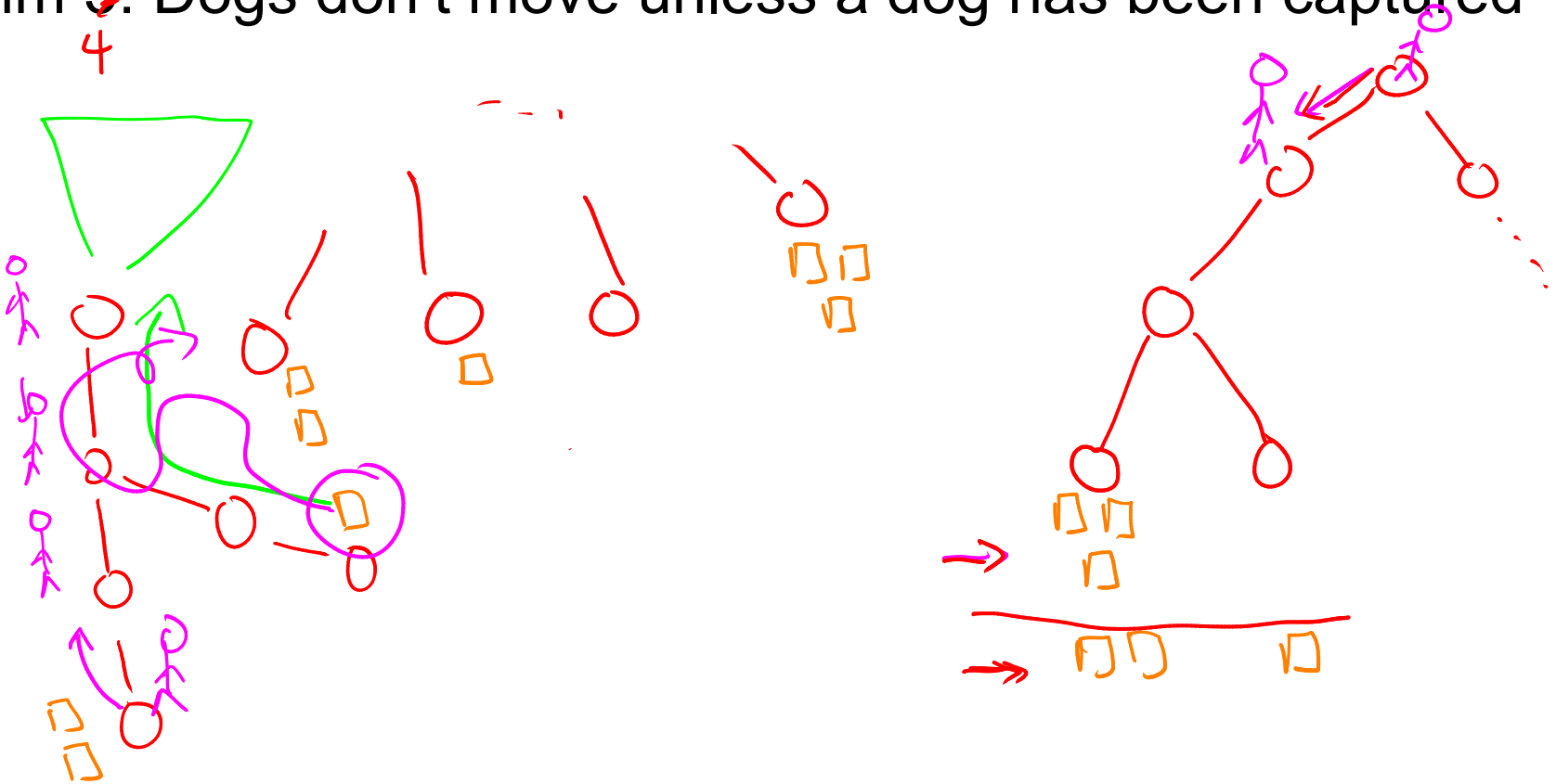


Claim 3: Dani never goes backwards (unless she catches a dog)



~~Claim 4: Dogs always hide in leaves~~

Claim ~~5~~ 4: Dogs don't move unless a dog has been captured



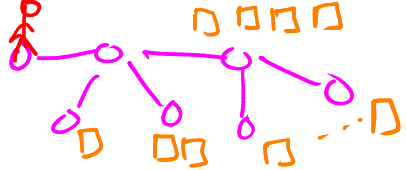
Dog Catcher II

Dani starts at vertex 1 in a tree of **N** vertices. There are also **D** dogs in the tree. Dani and the dogs take turns:

- The dogs distribute themselves among the leaves (other than the one Dani is in)
- Dani picks a leaf, travels there (paying cost equal to the distance), then captures all the dogs there.

Dani wants to minimise the cost, the dogs want to maximise it.

DP to the rescue



Let dp[at][d] = the minimum total cost to catch d dogs, if Dani starts in leaf "at".

If k dogs are in leaf x, then Dani can go there and catch them for a total cost of dist(at, x) + dp[x][d-k]. So the dogs want to arrange themselves so that the **minimum** of all dist(at, x) + dp[x][d-k] is **maximised**.

Suppose that dp[at][d] = t. Then at leaf x, binary search for the largest value k such that dist(at, x) + dp[x][d-k] <= t (since dp[x][d-k] decreases as k gets larger).

If the sum of all k's >= d, then dp[at][d] >= t

If the sum of all k's < d, then dp[at][d] <= t.

So binary search on t!

O(LD) states, each can be computed in **O(LlogTlogD)**, where **L** is the number of leaves, and **T** is the max answer (which we already established is **ND**).

Shaving off a log factor

This is already plenty fast enough (**O(N^5)** was enough to pass), but for fun, we can shave off a log factor when computing dp[at][d].

Start with no dogs anywhere. Pick the leaf where adding a dog would reduce the minimum t the least. Repeat until d dogs total have been assigned.

So **O(LD^{2}logL)** in total. Funny side note, this problem has GT (Floyd-warshall), DS (priority queue) and DP (DP) all in one.