T1

Statement

Alice and Bob are visiting cities on a very long road that stretches from points -10^9 to 10^9. Alice starts at point A while Bob starts at point B.

There are n cities to visit, where the i-th city is at point t_i. Each city must be visited by Alice or Bob at least once, but they can be visited in **any** order.

What is the minimum **total** distance Alice and Bob travel?

Constraints:

- $1 \le n \le 200'000$
- $-10^9 \le A, B, t_i \le 10^9$

Observations:

- The order of cities t_i don't matter, so **sort t in increasing order**.
- The order of A and B don't matter too, so **assume A ≤ B** (swap A and B if not).

Subtask 1 (16 points): $n \le 20$, $-10^6 \le A, B, t_i \le 10^6$

Fix the set of cities visited by Alice $a_1 \le a_2 \le \ldots \le a_k$ (and let Bob visit the other cities $b_1, b_2, \ldots, b_m$). What is the minimum distance Alice needs to travel to visit all cities a_i?

- We can do the same thing for Bob.

We must visit both the leftmost city a_1 and the rightmost city a_k. And if we do visit both of them, we must have passed by all cities!

Therefore, we only need to consider two cases:

- Visit a_1 then a_k
  - Distance = $|A - a_1| + a_k - a_1$
- Visit a_k then a_1
  - Distance = $|A - a_k| + a_k - a_1$
- Take the minimum: Alice needs to travel

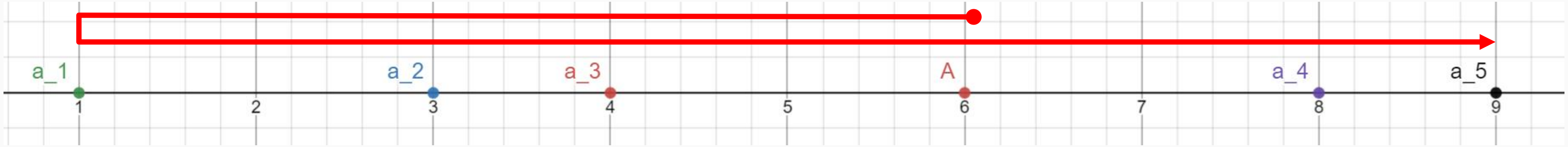
$$\min(|A - a_1|, |A - a_k|) + a_k - a_1$$

- $|x|$ is the absolute value of x, e.g. $|-3| = 3$, $|4| = 4$, $|0| = 0$

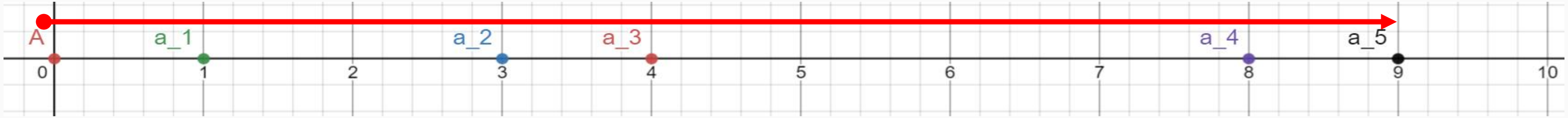
Subtask 1 (16 points): $n \le 20$, $-10^6 \le A, B, t_i \le 10^6$

Visit a_1 then a_k:

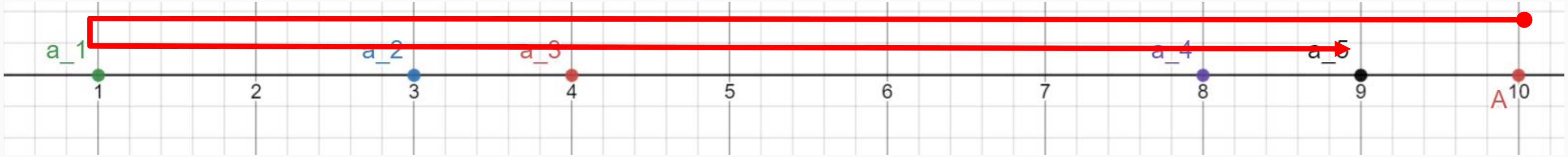
- A between a_1 and a_k



- A on the left of a_1



- A on the right of a_k

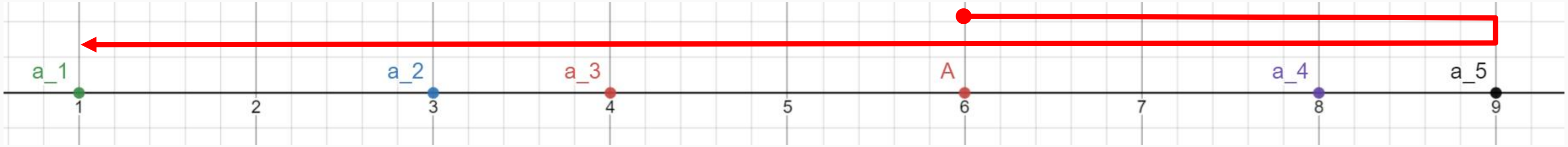


In all cases, the distance is |A - a_1| + a_1 - a_k.

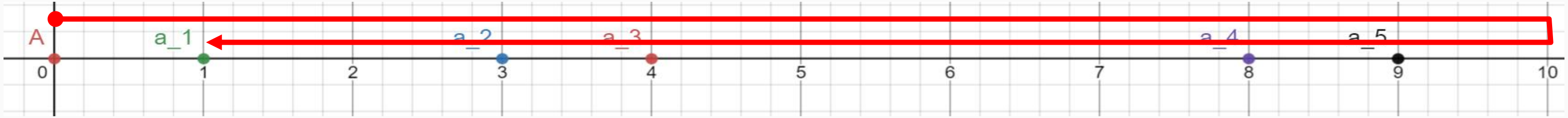
Subtask 1 (16 points): $n \le 20$, $-10^6 \le A, B, t_i \le 10^6$

Visit a_k then a_1:

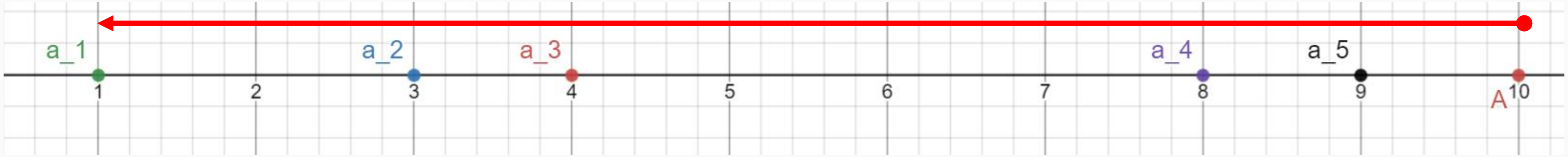
- A between a_1 and a_k



- A on the left of a_1



- A on the right of a_k



In all cases, the distance is $|A - a_k| + a_1 - a_k$.

Subtask 1 (16 points): $n \le 20$, $-10^6 \le A, B, t_i \le 10^6$

"Fix the set of cities visited by Alice $a_1 \le a_2 \le \dots \le a_k$."

Which a_i's to choose?

Try all subsets of all cities t_i: There are $2^{20} = 1048576$ possible choices of a_i's (b_i's are fixed after this). The answer is the minimum value of

$$\min(|A - a_1|, |A - a_k|) + a_k - a_1 +$$
$$\min(|B - b_1|, |B - b_k|) + b_m - b_1$$

over all choices of a_i's. Note that Alice or Bob may visit nothing!

One can implement this with recursion or bitmasks.

Time complexity: $O(n \cdot 2^n)$.

Subtask 2, 3 (36, 21 points): n ≤ 5000

These subtasks are here just in case you do something inefficient or cause integer overflow for some reason.

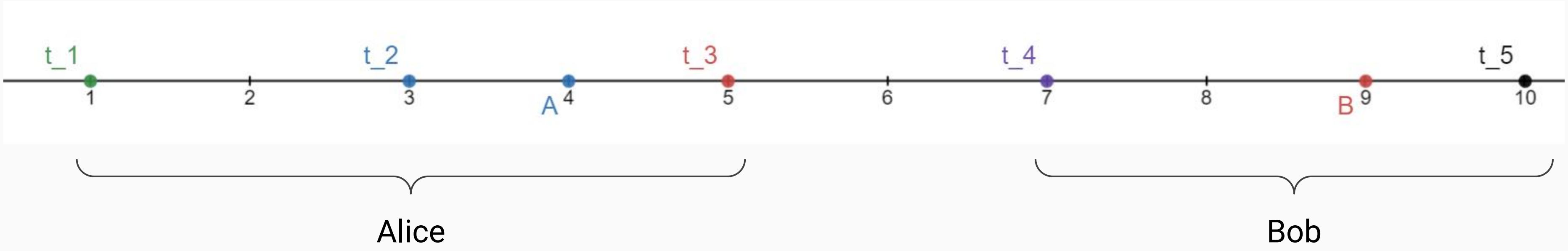
- E.g. Using bubble sort for sorting, this happened before
    - C++ has the built-in std::sort function in the header <algorithm>.
- Coincidentally, the constraints made the maximum answer 2 * 10^9, which fits in an int.

Subtask 4 (16 points): $n \le 200'000$

What would the optimal choice of a_i and b_i look like?

Claim: In one of the optimal choices, Alice visits $t_1, \ldots, t_k$ and Bob visits $t_{k+1}, \ldots, t_n$ for some k.

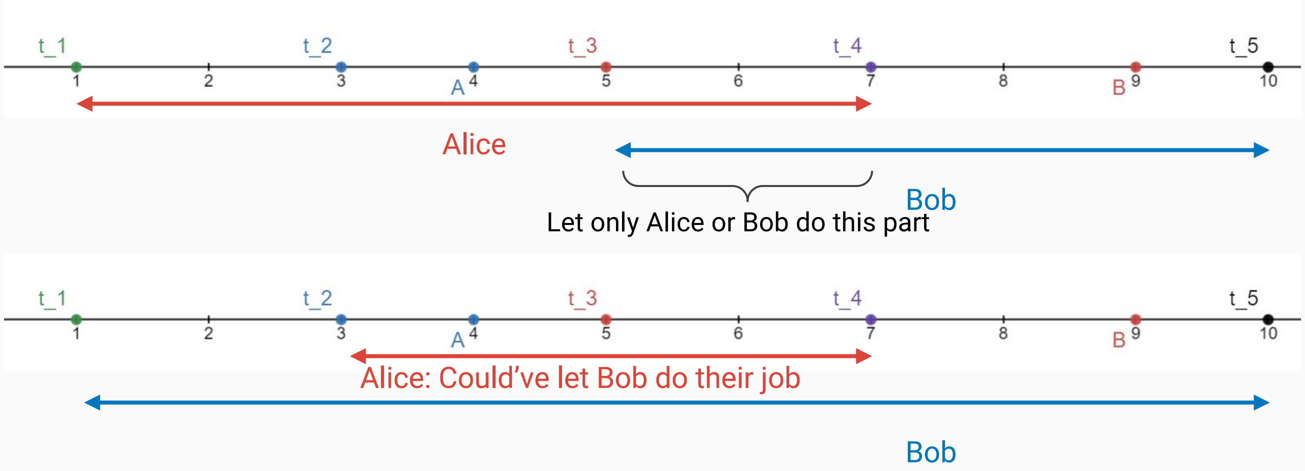
- I.e. Alice visits some left half and Bob visits some right half.
- Remember we assumed that $A \le B$.



Subtask 4 (16 points): n ≤ 200’000

Proof:

- Note that if Alice and Bob visit all cities they passed by, they will each visit a **continuous** range of cities no matter what they do.
- If the two ranges overlap, we can always remove the overlap while visiting all cities and get a lower total distance.
- Since A ≤ B, Alice must visit some left half, Bob must visit some right half.



(same if Bob is contained in Alice)

Subtask 4 (16 points): $n \le 200'000$

Solution: Try every i ($0 \le i \le n$), and let Alice visit the leftmost i cities $t_1, t_2, \ldots, t_i$, and Bob visit the remaining cities $t_{i+1}, \ldots, t_n$.

The minimum distance for a fixed i ($1 \le i \le n - 1$) is

```
min(|A - t_1|, |A - t_i|) + t_i - t_1 +
 min(|B - t_i|, |B - t_n|) + t_n - t_i
```

Special cases:

Alice visits all cities (i=n):

```
min(|A - t_1|, |A - t_n|) + t_n - t_1
```

Bob visits all cities (i=0):

```
min(|B - t_1|, |B - t_n|) + t_n - t_1
```

Take the minimum among all cases. Time complexity: O(n log n) due to sorting.

T2

Abridged Statement

Dragon 1 wants to send a love letter to dragon t. Dragon i's age is a_i, and there are K pairs of close friends (u, v).

All dragons can send a letter (after getting it) to all other dragons. The time taken for dragon u to send a letter to dragon v is:

- 0 if u and v are close friends.
- $|a_u - a_v|$ otherwise.

For every dragon t, find the minimum time needed for dragon 1 to send a letter to dragon t.

Constraints:

- $1 \le n, K \le 200'000$
- $1 \le a_i \le 10^9$

Equivalent Statement

Given a complete weighted undirected graph with n vertices (there is an edge between every pair of vertices), n integers $a_1, \ldots, a_n$, and K pairs of close friends (u, v).

The weight of edge (u, v) is:

- 0 if u and v are close friends.
- $|a_u - a_v|$ otherwise.

Find the minimum distance from vertex 1 to every vertex i.

Constraints:

- $1 \le n, K \le 200'000$
- $1 \le a_i \le 10^9$

Subtask 1 & 2 (18, 14 points): $n, k <= 2000$ (and $a_i = i$ for Subtask 1)

Naively adding all the edges:

- For every pair of vertices i, j, add an edge of weight $|a_i - a_j|$ or 0 between node i and node j (depending on whether they are close friends).
- There are $O(N)$ nodes and $O(N^2)$ edges.
- Run a normal Dijkstra's algorithm.
- Time complexity: $O(N^2 \log N)$

Subtask 4, 5, 6: reducing the number of edges

Subtask 4 : K = 0, the min path between i and j is just |ai-aj|

Why?

Instead of adding all O(n^2) edges, sort the nodes by ai. Let v1,v2,...,vn be the sorted order. Adding the edges (vi,vi+1) with weight |a_vi - a_vi+1| is enough

Subtask 5 : ai <= ai+1

Add the edges (i,i+1) with weight ai+1-ai and also the k edges of weight 0

Subtask 6 : no constraint

Sort the nodes into v1,v2...vn.add edges between (vi,vi+1) with weight |a_vi - a_vi+1| and the k edges of weight 0 from the input. Run a naive dijkstra from node 1 and we are done

Final complexity : O((n+k)logn)

T3

Abridged Problem Statement

Find the minimum distance two people on a number line, starting at A and B, need to travel in total to visit n targets at positions t[i] in order. Either person can visit each target.

Subtask 1

Subtask 1 (5 points): |t[i]|, |A| ≤ 1000, B=10^9

Bob would have to travel a minimum distance of 10^9 - 1000 to get to a single event.

If Alice attended all the events, she would drive a max. distance of 2000*10^5 = 2*10^8 ≤ 10^9 - 1000. (from -1000 to 1000)

So Alice must visit all events, under these constraints.

Time complexity: O(n)

Subtask 2

Subtask 2 (8 points) $n \leq 20$

Brute force over all combinations of Alice and Bob for each event

Either Alice or Bob can visit each event, so there are 2^n combinations.

Time complexity: $O(2^n)$ or $O(n*2^n)$

Subtask 3

Subtask 3 (19 points) $n \leq 3000$

When events up to event i have been visited, either Alice or Bob must be at position t[i]. The other one can be anywhere visited before, including Alice or Bob's starting positions.

Use dynamic programming

Subtask 3

dp[i][j] = minimum distance travelled if Alice and Bob are at t[i] and t[j], in any order, and events up to event i have been visited

Candidate values for next DP:

If the person at t[i] moves to t[i+1], other person is at t[j] -> contribute to dp[i+1][j]

dp[i][j]+abs(t[i+1]-t[i]) contributes to dp[i+1][j]

If the person at t[j] moves to t[i+1], other person is at t[i] -> contribute to dp[i+1][i]

dp[i][j]+abs(t[i+1]-t[j]) contributes to dp[i+1][i]

Subtask 3

DP transitions:

dp[i+1][i]=min(dp[i][j]+abs(t[i+1]-t[j]))

dp[i+1][j]=dp[i][j]+abs(t[i+1]-t[i])

The dp[i+1][j] transition for j=i is accounted for within dp[i+1][i]

Subtask 3

Base cases:

Set t[-1] to A and t[0] to B, before all other t[i]

dp[0][-1]=0

Time complexity: O(n^2)

Subtask 4

Subtask 4 (12 points) $n \le 10^5$, $|t[i]|$, $|A|$, $|B| \le 100$

Same DP, but DP only on each distinct t[i] or for each t[i] from -100 to 100

Time complexity: $O(n \max(|t[i]|))$

Subtask 5

Subtask 5 (43 points) |t[i]|, |A|, |B| ≤ 2*10^5

Subtask 4 DP:

$dp[i+1][t[i]]=\min(dp[i][j]+abs(t[i+1]-j))$

How to optimise this from O(n^2)?

Subtask 5

Segment tree

Keep two min segtrees: one for travelling to the left and one to the right

The segtree for the left will store min(dp[i][j] + j) for each $j > x$ at index x

$\min(dp[i][j] + j) - x = \min(dp[i][j] + \text{abs}(j - x))$ as $j - x > 0$

The segtree for the right will store min(dp[i][j]-j) for each $j \leq x$ at index x

$\min(dp[i][j] - j) + x = \min(dp[i][j] + \text{abs}(j - x))$ as $j - x < 0$

Min of both segtrees = $\min(dp[i][j] + \text{abs}(j - x))$ over all j

Subtask 5

Update dp[i+1][j] -> this adds a constant to all elements, so store the offset from this constant instead of the value in this segtree, and add the constant back in afterwards

Update dp[i+1][i] -> update left segtree in range [min t_i, i], update right segtree in range [i, max t_i]

Time complexity: $O(n \log(\max(t[i])) + \max(t[i]))$

Subtask 6

Subtask 6 (13 points) No additional constraints

Instead of creating the segtree over the full range of -10^9 to 10^9, only create elements for A, B, and each t[i], sorting them and using coordinate compression.

Time complexity: O(n log n)

T4

Idea by: Zi Song

Abridged Statement

Given arrays x and a of length n. For a permutation p of length n, define f(p) as follows:

- Initially, the number line is white.
- For all i ($1 \le i \le n$), color [$x_i - a_i$, $x_i + a_i$] black.
- f(p) = total length of black segments.

Find the sum of f(p) over all permutations p, modulo 10^9+7.

Since order doesn't matter, sort x and a. **Assume x and a are increasing from now on.**

Subtask 1 (7 points): All a_i are equal

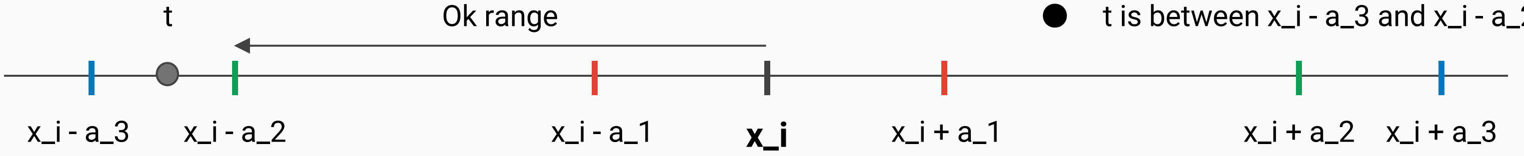
- Since the segments are the same for all permutations, the answer is
  n! * (total length of black segments for any permutation)
- We need to solve the problem: Given n segments [Li, Ri], what is the total length of the union of these segments?
- Solution:
  - Sort all segments by L.
  - Process segment 1 to n (sorted). We will maintain the last connected segment group. Let it be [last_l, last_r]. Suppose we are processing [L, R].
  - If L > last_r, we have a new segment group. Add last_r - last_l to the answer, then set the last connected segment group to [L, R].
  - Otherwise, we extend the segment group. Set [last_l, last_r] to [last_l, R].
  - In the end, we need to add last_r - last_l again for the last group.
- Time complexity: O(n log n) due to sorting.

Subtask 2 (8 points): $n \le 9$, $-2000 \le x_i, a_i \le 2000$

- Try every permutation of length n, then do the same as Subtask 1.
  - In C++, one can use next_permutation. To deal with duplicates, permute the sequence of (a_i, i) instead to make all values "distinct".
- Time complexity: $O(n! * n \log n)$.

Subtask 3 (31 points): $n \le 300$, $-10^5 \le x_i, a_i \le 10^5$

- Let's look at one point t and ask: How many permutations are there such that point t is colored black?
  - I.e. how many permutations are there such that at least one segment $[x_i, x_i - a_{p_i}]$ or $[x_i, x_i + a_{p_i}]$ covers point t?
- It's hard to count this directly, let's count the opposite: How many permutations are there such that no segment touches point t (segment's endpoint touching t is ok)?
  - The answer to the original question is $n!$ - answer to this
- Condition: For all i, $a_{p_i} \le |x_i - t|$



- t is between $x_i - a_3$ and $x_i - a_2$, so $c_i = 2$

- Fix one x_i. Then $a_1, a_2, \ldots, a_k \le |x_i - t|$ for some k, then $|x_i - t| < a_{k+1}, \ldots$
  - Example above has $a_1, a_2 \le |x_i - t|$, then $|x_i - t| > a_3, \ldots$
- In fact, if t is between $x_i \pm a_j$ and $x_i \pm a_{j+1}$, then $k = j$.
- **So p is not covered by any segment iff for all i, $p_i \le c_i$, where**
  - **c_i is the unique integer j s.t. t is between $x_i \pm a_j$ and $x_i \pm a_{j+1}$**

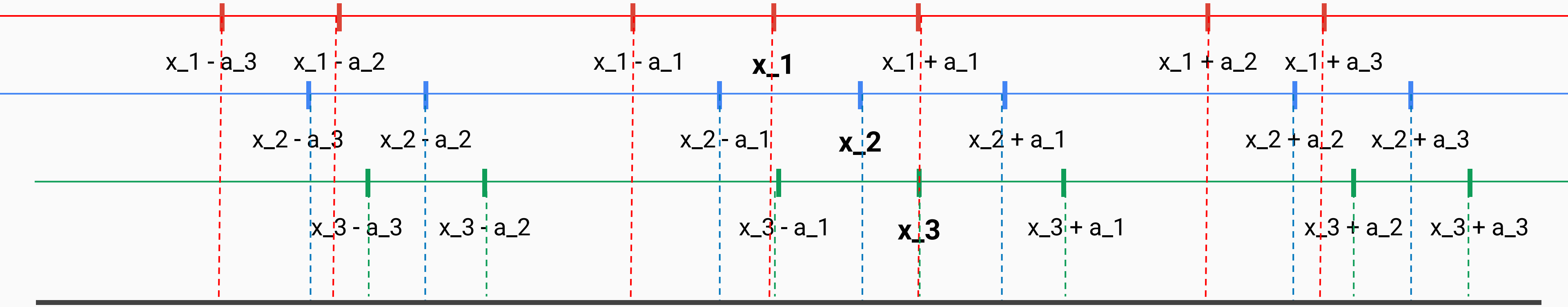
Subtask 3 (31 points): $n \le 300$, $-10^5 \le x_i, a_i \le 10^5$

- For a fixed point p, the problem reduces to:
  - Given an array c of length n ($1 \le c_i \le n$), how many permutations p satisfy $p_i \le c_i$ for all i?
  - E.g. c = (2, 3, 2), the answer is 2: p = (1, 3, 2), (2, 3, 1).
- Note that again, the order of c_i doesn't matter!
  - Let's **sort c in increasing order**.

- Now $c_1 \le c_2 \le \dots \le c_n$. How many permutations satisfy $p_i \le c_i$?
  - p_1: There are c_1 choices.
  - p_2: 1 number is used for p_1, and since $c_1 \le c_2$, there are $c_2 - 1$ choices.
  - p_3: 2 numbers are used for p_1 and p_2. There are $c_3 - 2$ choices.
  - …
  - p_i: i - 1 numbers are used. There are $c_i - (i - 1)$ choices.
- Thus, the answer is $c_1 * (c_2 - 1) * (c_3 - 2) * \dots * (c_n - (n - 1))$.
- Time complexity: $O(n \log n)$ time ($\log n$ due to sorting).
- Remember this is only for a fixed point p!

Subtask 3 (31 points): $n \le 300$, $-10^5 \le x_i, a_i \le 10^5$

- Finally, note that there are only $O(n^2)$ segments with different values of $\{c_i\}$.
  - Each x_i creates $\le 2n + 2$ segments:
    - $[x_i, x_i + a_1], [x_i + a_1, x_i + a_2], \ldots, [x_i + a_n, INF]$
    - $[x_i - a_1, x_i], [x_i - a_2, x_i - a_1], \ldots, [x_i - a_n, -INF]$
  - There are $\le n$ different x_i's.
    - Their intersections create at most $n * (2n + 2) = O(n^2)$ segments
- Let's compute the answer for each segment in $O(n \log n)$, multiply it by the segment length, then sum up the answers over all segments.



- Time complexity: $O(n^2 * n \log n) = O(n^3 \log n)$.

Subtask 6 (x points): $n \le 2000$, $-10^9 \le x_i, a_i \le 10^9$

$O(n^3 \log n)$ will TLE; we need something like $O(n^2 \log n)$.

Recall that we took $O(n \log n)$ time to compute this for each of the $O(n^2)$ segments:

- Given an <u>increasing</u> array c of length n ($1 \le c_i \le n$), the number of permutations p that satisfy $p_i \le c_i$ for all i is

$$c_1 * (c_2 - 1) * (c_3 - 2) * \ldots * (c_n - (n - 1)).$$

Can we calculate this for each segment faster?

Subtask 6 (x points): $n \le 2000$, $-10^9 \le x_i, a_i \le 10^9$

Suppose for a segment s_1, we know $c = \{c_1, c_2, \ldots, c_n\}$, and the value of

$$f(c) = c_1 * (c_2 - 1) * (c_3 - 2) * \ldots * (c_n - (n - 1)).$$

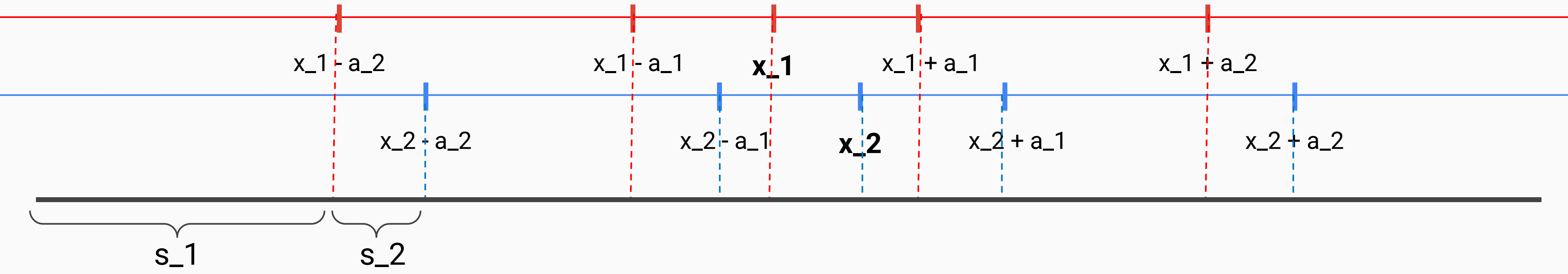
- In the example: $n=2$, $c = \{2, 2\}$, $f(c) = 2 * (2 - 1) = 2$.

Now, we move from s_1 to the next segment, s_2. We want to find $f(c)$ fast. What changes?

- c_1: 2 -> 1
- c_2: No change
- $f(c) = \mathbf{1} * (2 - 1) = \mathbf{1}$.

Generally, only one term in $f(c)$ increases or decreases by 1!

But which term changes by what?



Subtask 6 (x points): $n \le 2000$, $-10^9 \le x_i, a_i \le 10^9$

Suppose we know the current $f(c) = c_1 * (c_2 - 1) * (c_3 - 2) * \dots * (c_n - (n - 1))$

If we go from the left of $x_i - a_j$ to the right:

- The value j was in c, say $c_k = j$
- c_k becomes $c_k - 1$

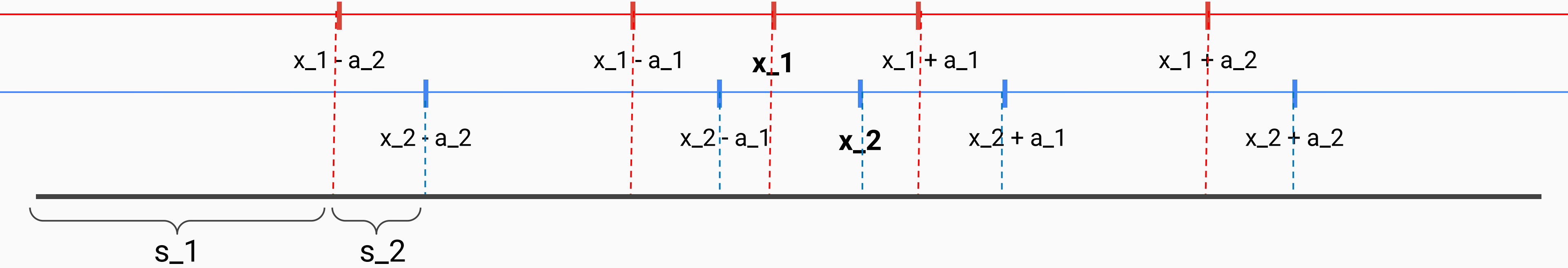
New example (not the picture):

$$c = \{3, 3, 4, 4, 5\}$$

Suppose we go from the left of $x_i - a_4$ to the right: one 4 becomes 3

$$c = \{3, 3, 3, 4, 5\}$$

We need to maintain $f(c)$ fast when c changes. How does c change?



Subtask 6 (x points): $n \le 2000$, $-10^9 \le x_i, a_i \le 10^9$

$$c = \{3, 3, \mathbf{4}, 4, 5\}$$
$$f(c) = 3 * (3-1) * \mathbf{(4-2)} * (4-3) * (5-4) = 12$$

Suppose we go from the left of $x_i - a_4$ to the right. One 4 becomes 3:

$$c = \{3, 3, \mathbf{3}, 4, 5\}$$
$$f(c) = 3 * (3-1) * \mathbf{(3-2)} * (4-3) * (5-4) = \mathbf{12 / (4-2) * (3-2) = 6}$$

Let's maintain the sequence c and f(c).

Generally, suppose we go from the left of $x_i - a_{\mathbf{j}}$ to the right.

- Find the smallest position k where $c_k = j$.
    - In the example, k=2 (0-indexed), as $c_2 = 4$.
- f(c) becomes $f(c) / (c_k - k) * [(c_k - 1) - k]$.
    - In the example: 12 / (4-2) * (3-2)
- Update c_k to $c_k - 1$.

The case for $x_i + a_j$ is similar.

Almost done! How do we find k fast? Binary search! c is sorted.

Subtask 6 (x points): $n \le 2000$, $-10^9 \le x_i, a_i \le 10^9$

But can we do division modulo $M = 10^9 + 7$ (prime)?

- $1/a = a^{M-2}$ (by Fermat's little theorem).
- You can compute this in $O(\log M)$ with binary exponentiation.

Each update takes $O(\log n)$ time due to binary search. We have $O(n^2)$ segments.
Time complexity: $O(n^2 \log n)$.