Task 3: Towers (`towers`)

Authored and prepared by: Ng Yu Peng

Terminology

Throughout this writeup, points only refer to points where cities are located, and we will refer to building towers in cities as picking points. A **column** refers to points with the same x-coordinate and a **row** refers to points with the same y-coordinate.

Subtask 1

Limits $N \leq 3$

Check if all points are in the same row or column. If they aren't, pick all the points. If they are all in the same row, pick the leftmost and rightmost points. If they are all in the same column, pick the topmost and bottommost points.

Time Complexity: $O(1)$

Subtask 2

Limitsl: $N \leq 16$

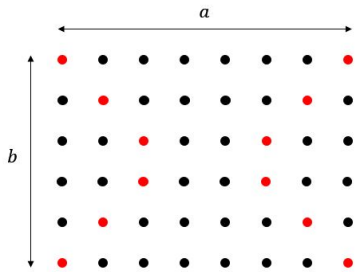
Use a bitmask to simulate all 2^N ways of building towers, and for each of them, iterate through the points to store the max/min x-coordinates in each row, and the max/min y-coordinates in each column, as well as the number of points in each row/column. Now iterate through all the points again and check the conditions are satisfied.

Time Complexity: $O(N2^N)$

Subtask 3

Limits: $N = ab$ for some positive integers a, b and $(X_{ai+j}, Y_{ai+j}) = (i+1, j)$ for all integers i, j with $0 \leq i \leq b-1, 1 \leq j \leq a$

Basically the points form a rectangular lattice, just consider the cases $a > b$ and $a \leq b$. In the case $a > b$, choose the points (red points are chosen) like so:



The case $a \leq b$ is similar.

Time Complexity: $O(N)$

Subtask 4

Limits: For every integer a, there are at most two cities whose x-coordinate is a

Pick the leftmost and rightmost cities in each row/ Since each column has at most 2 points the conditions are satisfied,

Either sort the sets of points in each row to find the leftmost and rightmost, or keep track of the leftmost and rightmost x-coordinates in each row while iterating through the points.

Time Complexity: $O(N \log N)$ or $O(N)$

Subtask 5

Limits: $N \leq 5000$

We now describe a general algorithm to choose where to build the towers. Let's extend the idea in the previous subtask: first pick the leftmost and rightmost points in each row.

Since there may be more than 2 points in a column, some columns may have more than 2 points picked. Thus we will keep making changes to the set of points picked, while making sure each row still has at most 2 points picked and each point is either picked or lies between two picked points, until no column has more than 2 points picked.

Consider any column with more than 2 points picked, then look at a picked point in that column which is not the topmost or bottommost picked point. If it is the only point picked in its row, delete it from the set of picked points. If it is the rightmost picked point in its row, delete it from the set and add the point immediately to its left in the same row. If it is the leftmost picked point in its row, delete it from the set and add the point immediately to its right in the same row.

Clearly, any point in the column of the removed point will still be between two points in the same row or column, and since we effectively just replaced a point with the point to its immediate left/right, all points in its row will still be between two points in the same row/column, and obviously each row will still have at most 2 points picked. The process terminates when there is no column with more than 2 picked points, and when this happens all conditions are simultaneously satisfied.

Let the distance between the 0/1/2 picked points in each row be the number of points between them including themselves. Each time we update the set of picked points, the distance decreases by 1 in the row we change, so the sum of distances in all rows decreases by 1. Since the sum of distances at the start is N, the process terminates in at most N changes to the picked points.

This subtask is catered to let solutions using $O(N)$ per change pass.

Time Complexity: $O(N^2)$

Subtask 6

Limits: $N \leq 100000$

You can store points as pairs (row, index in row) to find the point to its immediate left/right in the same row quickly. Use a set to store all picked points in each column in this way, then keep changing the picked points as long as there is some set with size more than 2.

Time Complexity: $O(N \log N)$ with large constant.

Subtask 7

Limits: $N \leq 1000000$

Since the topmost/bottommost picked point in each column can only get higher/lower every time we update the set, we can just store these two points for each column, and every other point goes into a waiting list for removal.

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