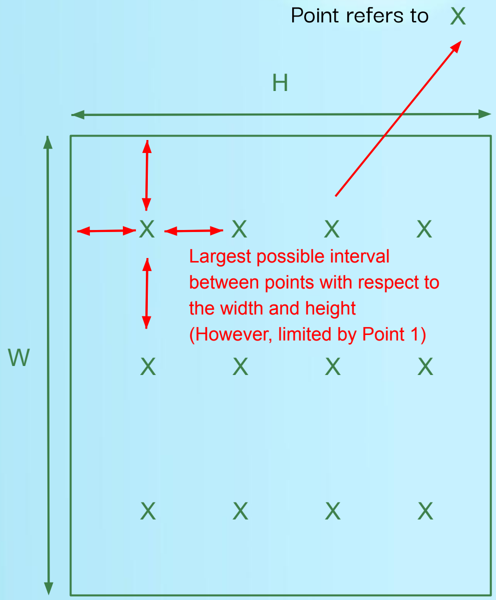
ALGORITHM BREAKDOWN - PART 1: RANDOM SEARCH

Core Idea	Complexity	Explanation for complexity
Allocate a portion of the threshold (0.1%) to conduct Random Search. We strategically allocated half of the threshold (0.05%) for Random Search because we want to leave the other half of the threshold (0.05%) for Local Search	**O(1)**	This is just **declaration of variables**
(a) Randomly search the map, but evenly and uniformly across the rows and columns of the input map (The number of points on each row and column will change accordingly to the dimension of the map) (Refer to figure on right) (b) Before executing the above Random Search, we have to find the "largest possible interval between points" first (Refer to figure on right). E.g., Interval_length = 44 Therefore, Point 1 = (0,0), Point 2 = (0,44) …	**(a) O(WH)** **(b) O(WH)**	(a) We need to iterate through both W and H to get the height of a **point** on the map **[x, y]** (b) We are bounded by the size of the map, ***Threshold = W * H * 0.005***
Get the coordinate with the highest height amongst the points in the Random Search. It will be used for Local Search in Part 2	**O(WHlogWH)**	(1) After getting the height at each point, we append it to a dictionary. **Key = height** **Value = coordinates of a point** [x,y]. **This is *O(1)*.** (2) We sort the keys of the dictionary and return the points with the highest height. **Sort is *O(WHlogWH)*** **Return is *O(1)***

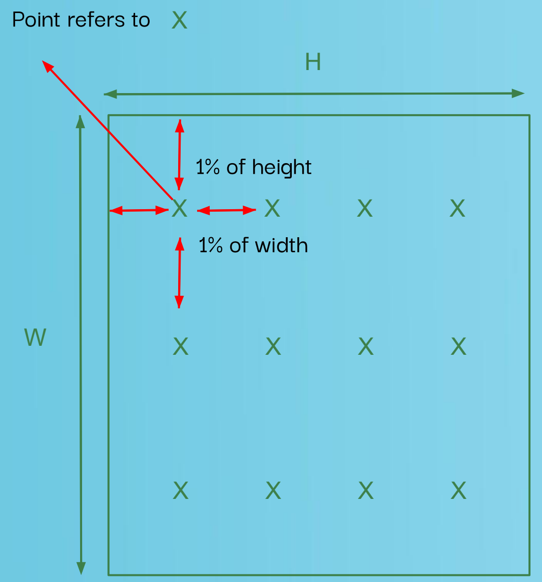


Overall Complexity of Part 1: Random Search

Complexity: O(WH)

ALGORITHM BREAKDOWN - PART 2.1: LOCAL SEARCH

Core Idea - Conduct loose Local Search recursively on the points identified from Part 1



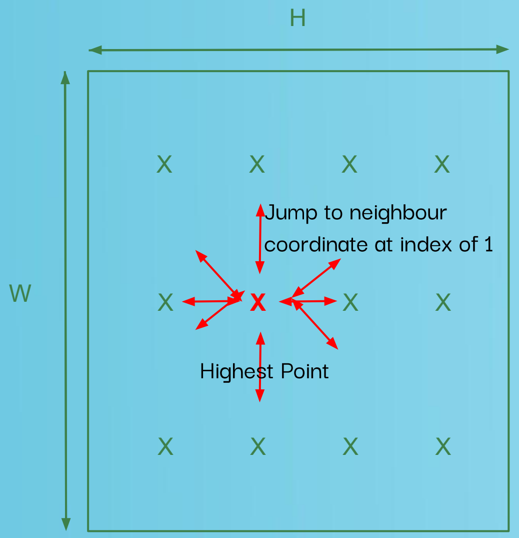
Core Idea	Complexity	Explanation
Conduct Local Search recursively on the points identified from Part 1	**O(1)**	The algorithm checks for elevation at a fix 1% of the map size, thus complexity will be O(1)
For the first local search, the algorithm recursively **Traverse** through top, left, right and bottom of the coordinates identified from part 1. The traversal search for higher elevation at a interval of **1% of the map's Height and Width. This is a loose local search that leaps across the point.** After identifying the elevation of the original coordinates and its surrounding coordinate, the algorithm will recursively call for a traversal function at the highest point of the elevation found. This allows the algorithm to discover the highest point within each point		
The condition to **end** the traversal are 1. If the traversal exceeds the map range 2. If the traverse coordinates have been visited 3. Exceeds given search limit for each point		

ALGORITHM BREAKDOWN - PART 2.2: LOCAL SEARCH

Core Idea - Conduct a tight Local Search recursively on the points identified from Part 2.1

Contributions - Part 1 Lin Tao, Part 2 Jun Yang

Both members participated actively and contributed insightful ideas from their own perspectives



Core Idea	Complexity	Explanation
Conduct Local Search recursively on the points identified from Part 1	O(N)	The algorithm checks at a interval of 1, thus complexity will be O(N) depending on the input size
For the second local search, the algorithm recursively **Traverse** through top, left, right, bottom and the four diagonal points of the coordinates identified from the highest point discovered in part 2. The traversal search for higher points at a interval of **1. Conducting a tight search.**		
The condition to **end** the traversal are 1. If the traversal exceeds the map range 2. If the traverse coordinates have been visited 3. If there are no remaining search leftover (without hitting punishment ratio)		

Conclusion

1. Random Search in Part 1 would consume time complexity of O(WH)
2. Traverse_1 is a recursive function with a time complexity of O(1)
3. Traverse_2 is a recursive function with a time complexity of O(N)

This Algorithm has a Worse Case time complexity of O(WH)