Atefe Rajabi

40230563

HW1 minimax

1. Generate Affine Functions (Section 1):

- The code defines a function `generate_affine_functions` that generates a specified number of random affine functions.

- Each affine function is represented as a dictionary with 'slope' and 'intercept' properties.

2. Plot Affine Functions (Section 1):

- The code uses Matplotlib to plot the generated affine functions.

- It sets x-axis ticks explicitly, ensuring precision of 1 between -10 and 10.

- The `x` values for the plot range from -5 to 5.

- Each affine function is plotted as a line on the graph.

3. Sort Functions Based on Slope Ascending, Intercept Descending (Section 2):

- The `affine functions` list is sorted based on two criteria: slope in ascending order and intercept in descending order.

- This sorting order is achieved by using the `sorted` function with a custom key function.

- The time complexity of this part is O(n log n), where 'n' represents the number of affine functions.

4. Finding Min of Maximums (Section 3):

- The `minimax` function calculates the minimum of maximum values for the sorted affine functions.

- It uses a stack `S` to keep track of points and iterates through the sorted functions to find min-max points.

- The minimum of maximums is determined based on the second element of the tuples in `S`.

- The result is returned as the minimum value of maximums.

- The time complexity of the function mentioned in the article, as stated in the problem document, is O(n).

5. Optimization Method:

The provided `minimax` function calculates the minimum of maximum values (min-max) of a set of affine functions based on their slopes and intercepts. It finds the minimum value by constructing a stack-like data structure `S` and iteratively processing the list of affine functions `affine functions`.

Here's how the code finds the minimum value:

1. The code then iterates through the list of affine functions `L`, starting from the second element, and compares the slope of each function `r` with the slope of the last function in `S`.

2. If the slope of the current function `r` is the same as the slope of the last function in `S`, it indicates that these two functions are parallel, and there is no need to calculate `t`. Calculating `t` in such cases can lead to division by zero and is unnecessary because parallel lines will not intersect. Therefore, the code continues to the next function in the loop. This step is both for eliminating duplicate slopes and avoiding division by zero errors when dealing with parallel lines.

3. If the slope of the current function `r` is different from the slope of the last function in `S`, the code proceeds to calculate the intersection point `t` between the current function and the last function in `S`. The intersection point `t` is determined by solving the equation:

$$t = \frac{r['\text{intercept}'] - S[-1][0]['\text{intercept}']}{S[-1][0]['\text{slope}'] - r['\text{slope}']}$$

4. After calculating `t`, the code enters a while loop to remove elements from `S` while `t` is less than or equal to the value of `t` in the last element of `S`. This is done to maintain only the maximum points on the functions.

5. After processing the loop and finding the maximum points, the code appends the current function `r` and its calculated `t` to the `S` list.

6. Finally, the code uses the `min()` function with a key function to find the minimum value in `S`. Specifically, it finds the element with the minimum `t` value, representing the minimum of maximum values for the set of affine functions.

The result, `min value`, represents the minimum of the maximum values of the affine functions.

This approach essentially keeps track of the maximum points of the functions, avoids unnecessary calculations for parallel lines, and then finds the minimum value among those maximum points.

6. Overall Report:

- The code successfully generates, visualizes, sorts, and analyzes a set of random affine functions.

- It finds the minimum of maximum values based on specific criteria.

- The precision of the x-axis ticks has been set to 1, which ensures a clear representation of the functions in the plot.

- The overall time complexity is O(n log n + n), where O(n log n) is for sorting n affine functions, and O(n) accounts for the minimax function.