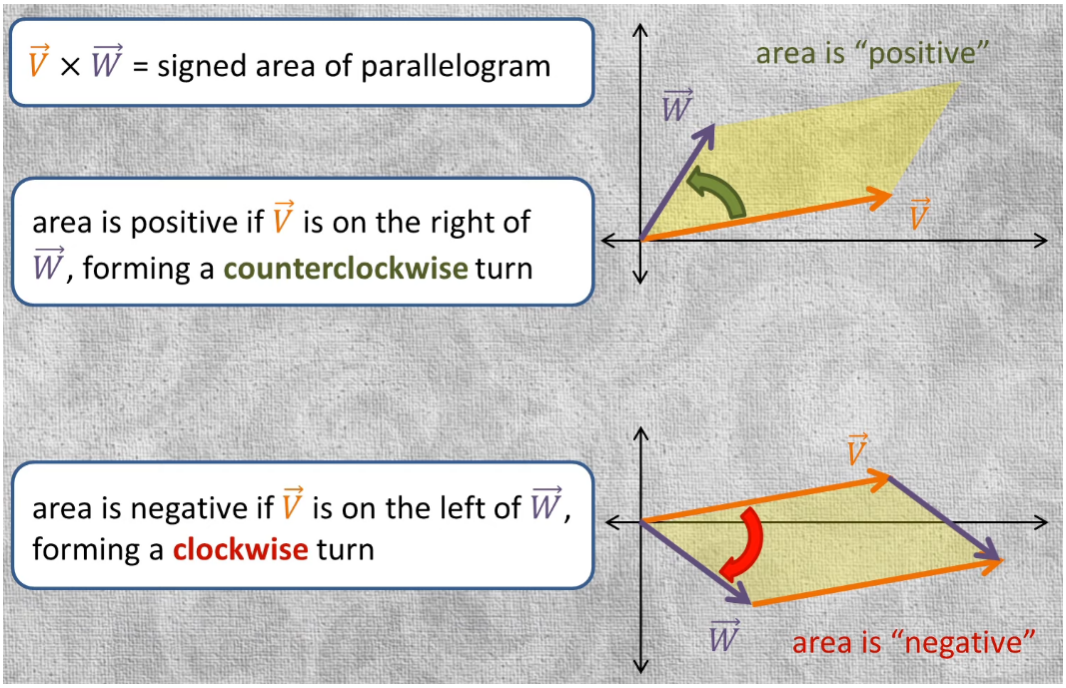
Q1: Convex Hull Computation:

Algorithm Overview

Intution (Graham Scan)

1. Duplicates are handled at the time of input coordinate using map.
2. Base Case if input contain only one element that itself is convex hull.
3. Find coordinate have lowest y value (and the leftmost x if there are multiple) as the starting point minXY.
4. Sort input coordinates based on their area with respect to minXY.
    - Area is negative if clockwise turn.
    - Area is positive if counterclockwise turn.
    - Area is zero if collinear(in this case nearest point is taken first).



5. Initialize an vector hull to store the convex hull with minXY and first sorted coordinate.

6. Iterate through the sorted points:
    - Here we have used the same above area method to check the rotation
    - While the last two points in hull and the current point form a clockwise turn or being collinear, pop last point from hull.
    - Add the current point to hull.
7. Add the starting point minXY again to close the convex hull.

Time Complexity Analysis

The time complexity of this algorithm mainly depends on two parts:

1. Sorting the input points based on polar angles: **O(n*log(n))**
2. Constructing the convex hull: **O(n)**

the time complexity is dominated by the sorting step, which is **O(n*log(n)).***

Observations and Insights

- We have taken care of all edge cases, including collinear and duplicate points, correctly.
- The use of area to sorting ensures that the convex hull is computed efficiently in counterclockwise order.

References

- https://stackoverflow.com/questions/7774241/sort-points-by-angle-from-given-axis
- https://stackoverflow.com/questions/482278/test-case-data-for-convex-hull

Q2: Radix Sort

Algorithm Overview

Radix Sort is a linear sorting algorithm that sorts elements by processing them digit by digit. It is an efficient sorting algorithm for integers or strings with fixed-size keys.

Input Processing

1. Input `n` and a list of `n` integers.
2. Radix Sort is a non-comparative sorting algorithm that sorts integers based on their digits from least significant to most significant.

Radix Sort Steps

1. Find the maximum absolute value (`maxx`) among the input numbers.
2. Sort the numbers based on each digit (in powers of 10) using Counting Sort (`cSort`).
3. Repeat step 2 for all digits.

Time Complexity

The time complexity of Radix Sort depends on the number of digits (`d`) in the maximum input number and the number of elements (`n`) in the list.

- Radix Sort: O(d * n).

Insights

- Radix Sort efficiently sorts integers by processing digits.
- It achieves linear time complexity for each digit pass, making it efficient for large datasets.
- Handles both positive and negative numbers.

References

- https://www.geeksforgeeks.org/radix-sort/

Q3: Median of Medians

Algorithm Overview

It is an approximate median selection algorithm that helps in creating asymptotically optimal selection algorithms by producing good pivots that improve worst case time complexities for sorting and selection algorithms.

Intuition

1. If the array size is small, sort it and return the Kth largest element.
2. Divide the array into subgroups and find subgroup medians.
3. Recursively find the median of medians (MoM).
4. Partition the array into smaller, equal, and larger elements relative to MoM.
5. Determine the Kth largest element based on the partitioning.

Time Complexity Analysis

- The Median of Medians algorithm operates in linear time, O(n), where n is the input size.

Observations and Insights

- Efficiently finds the Kth largest element with linear time complexity.
- Handles cases with multiple elements of the same value, returning the smallest rank.

References

- https://iq.opengenus.org/median-of-medians/