Maximal points implementation report

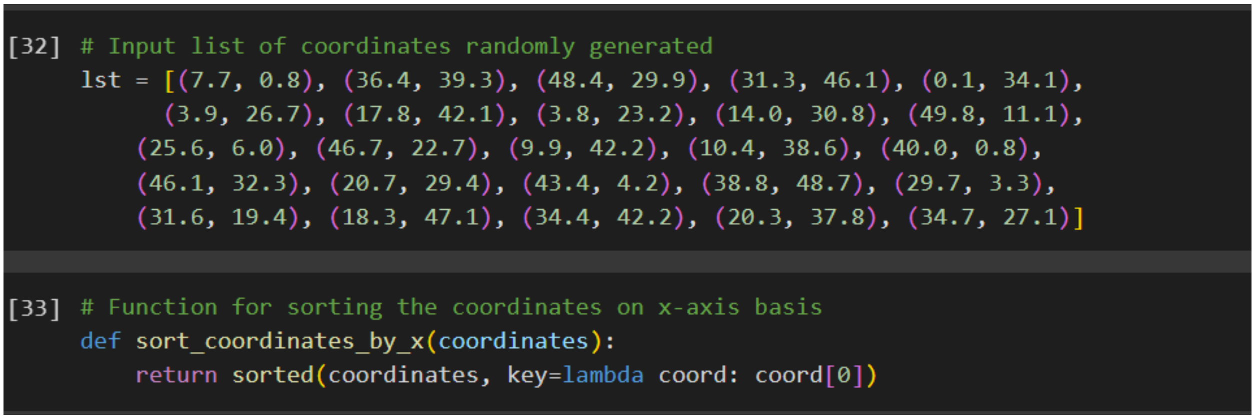
Name: Shravan Ram **Roll no: 202210101200003** **Date: 26-01-2024**

Screen Recording URL: Watch video
Implementation file: View code

Observation:

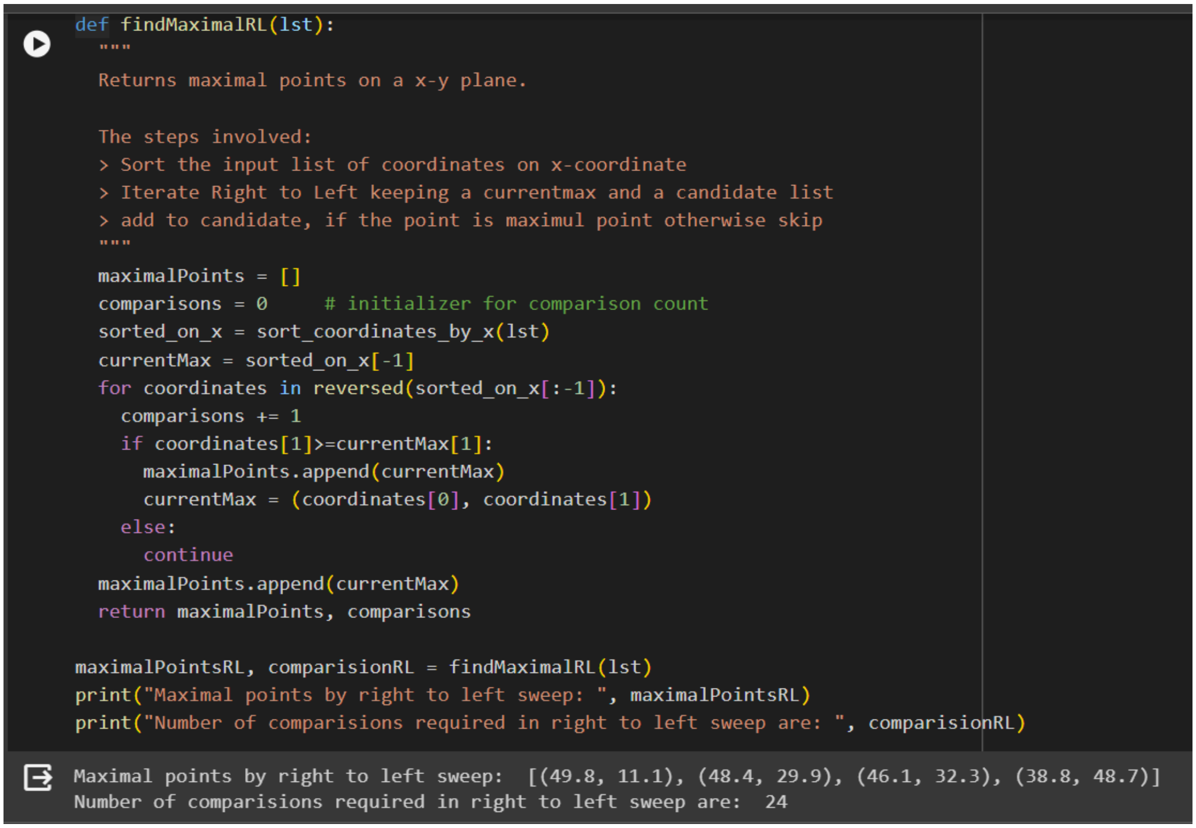
In this implementation, there are broadly two steps involved i.e. preprocessing to sort the coordinates by x-axis and finding the maximal points. Further, there are two ways to find the maximal points after sorting . Sorting and sweep is required in both approaches. And we need to maintain a candidate list in the left to right approach which takes O(n) additional space whereas in the right to left approach we only need to keep a variable which takes O(1)space. Although in the interest of providing a combined list of all maximal points I have used a list in both approaches. Another point I observed is that in LR, every coordinate is added to the list and every non-maximal point is deleted from the once. Whereas in the RL approach, we only add to the list if it is a maximal point.

Code:



```
[32] # Input list of coordinates randomly generated
     lst = [(7.7, 0.8), (36.4, 39.3), (48.4, 29.9), (31.3, 46.1), (0.1, 34.1),
           (3.9, 26.7), (17.8, 42.1), (3.8, 23.2), (14.0, 30.8), (49.8, 11.1),
         (25.6, 6.0), (46.7, 22.7), (9.9, 42.2), (10.4, 38.6), (40.0, 0.8),
         (46.1, 32.3), (20.7, 29.4), (43.4, 4.2), (38.8, 48.7), (29.7, 3.3),
         (31.6, 19.4), (18.3, 47.1), (34.4, 42.2), (20.3, 37.8), (34.7, 27.1)]
```

```
[33] # Function for sorting the coordinates on x-axis basis
     def sort_coordinates_by_x(coordinates):
         return sorted(coordinates, key=lambda coord: coord[0])
```

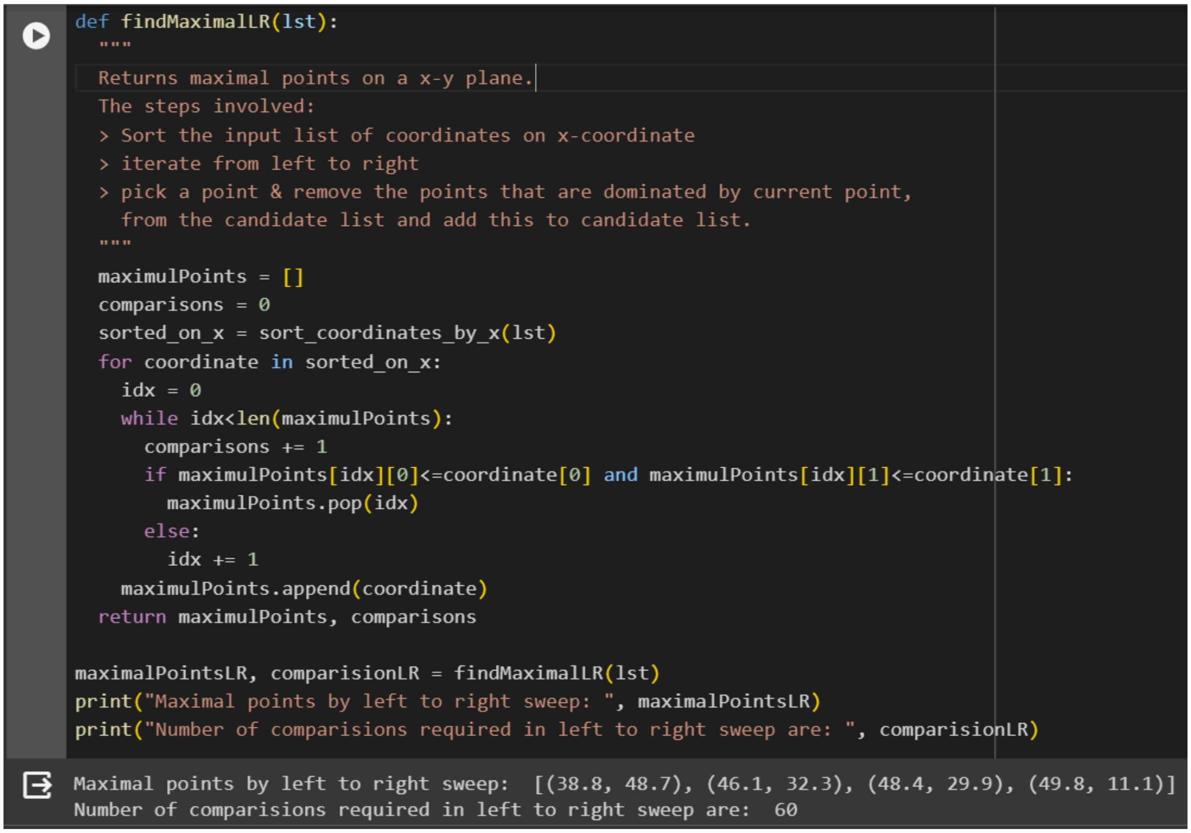


```
def findMaximalRL(lst):
  """
  Returns maximal points on a x-y plane.

  The steps involved:
  > Sort the input list of coordinates on x-coordinate
  > Iterate Right to Left keeping a currentmax and a candidate list
  > add to candidate, if the point is maximul point otherwise skip
  """
  maximalPoints = []
  comparisons = 0     # initializer for comparison count
  sorted_on_x = sort_coordinates_by_x(lst)
  currentMax = sorted_on_x[-1]
  for coordinates in reversed(sorted_on_x[:-1]):
    comparisons += 1
    if coordinates[1]>=currentMax[1]:
      maximalPoints.append(currentMax)
      currentMax = (coordinates[0], coordinates[1])
    else:
      continue
  maximalPoints.append(currentMax)
  return maximalPoints, comparisons

maximalPointsRL, comparisionRL = findMaximalRL(lst)
print("Maximal points by right to left sweep: ", maximalPointsRL)
print("Number of comparisions required in right to left sweep are: ", comparisionRL)
```

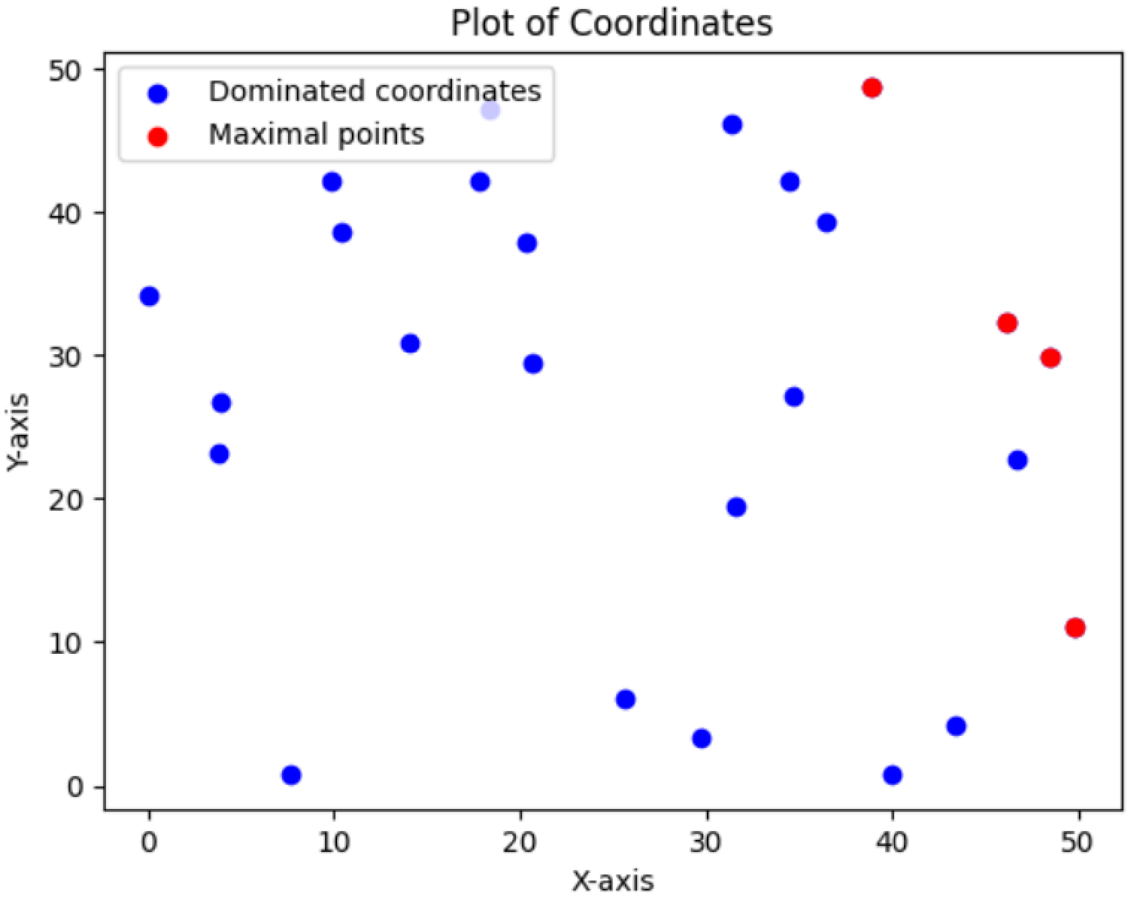
```
Maximal points by right to left sweep:  [(49.8, 11.1), (48.4, 29.9), (46.1, 32.3), (38.8, 48.7)]
Number of comparisions required in right to left sweep are:  24
```



```
def findMaximalLR(lst):
  """
  Returns maximal points on a x-y plane.
  The steps involved:
  > Sort the input list of coordinates on x-coordinate
  > iterate from left to right
  > pick a point & remove the points that are dominated by current point,
    from the candidate list and add this to candidate list.
  """
  maximulPoints = []
  comparisons = 0
  sorted_on_x = sort_coordinates_by_x(lst)
  for coordinate in sorted_on_x:
    idx = 0
    while idx<len(maximulPoints):
      comparisons += 1
      if maximulPoints[idx][0]<=coordinate[0] and maximulPoints[idx][1]<=coordinate[1]:
        maximulPoints.pop(idx)
      else:
        idx += 1
    maximulPoints.append(coordinate)
  return maximulPoints, comparisons

maximalPointsLR, comparisionLR = findMaximalLR(lst)
print("Maximal points by left to right sweep: ", maximalPointsLR)
print("Number of comparisions required in left to right sweep are: ", comparisionLR)
```

```
Maximal points by left to right sweep:  [(38.8, 48.7), (46.1, 32.3), (48.4, 29.9), (49.8, 11.1)]
Number of comparisions required in left to right sweep are:  60
```


Plot of Coordinates
● Dominated coordinates
● Maximal points
Y-axis
X-axis
0
10
20
30
40
50
0
10
20
30
40
50