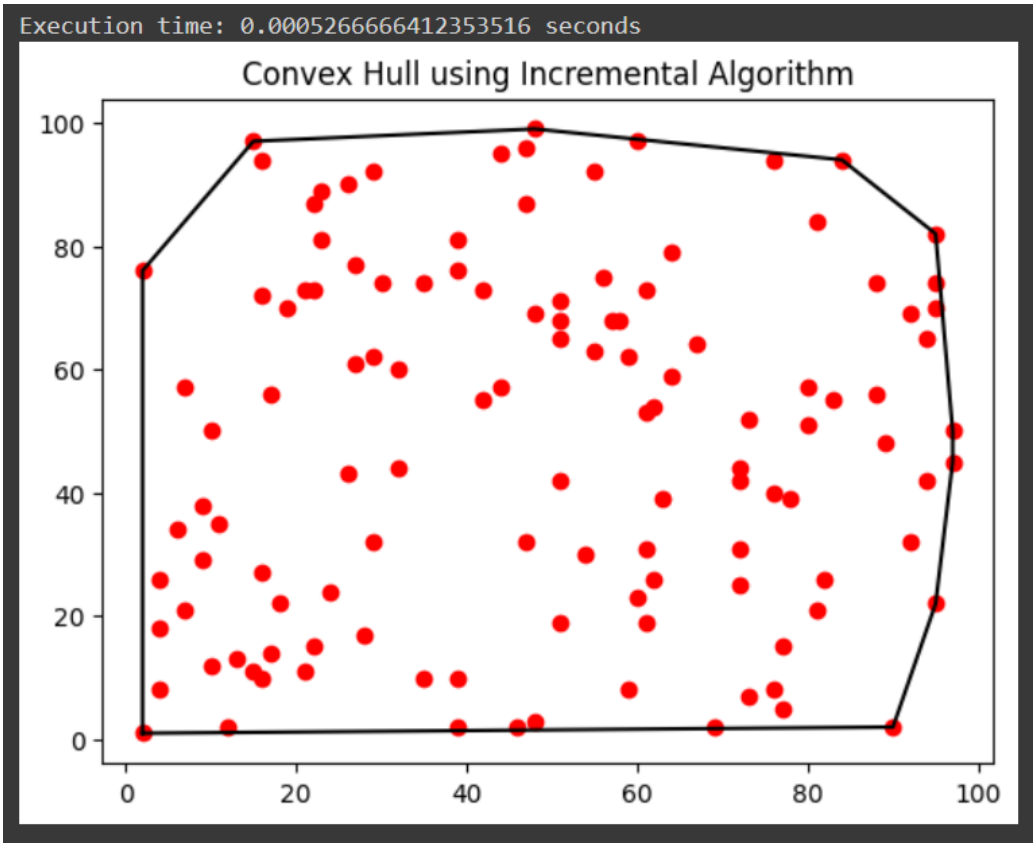
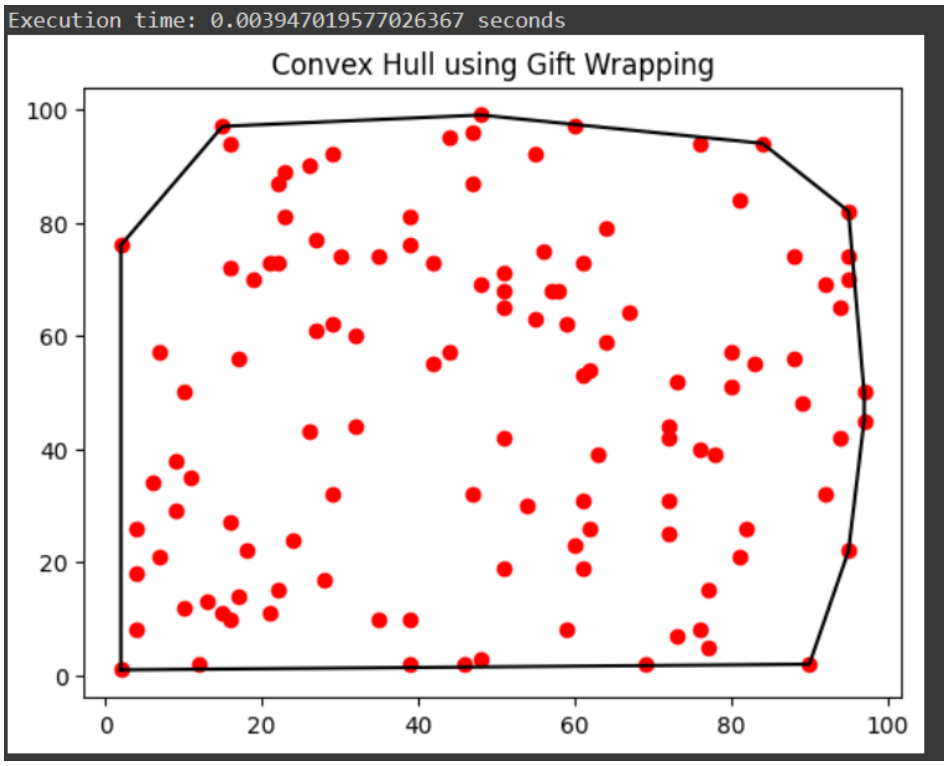
Part_1

1. Icremental Algorithm
2. Gift Wrapping
3. Divide and Conquer
4. Quickhull

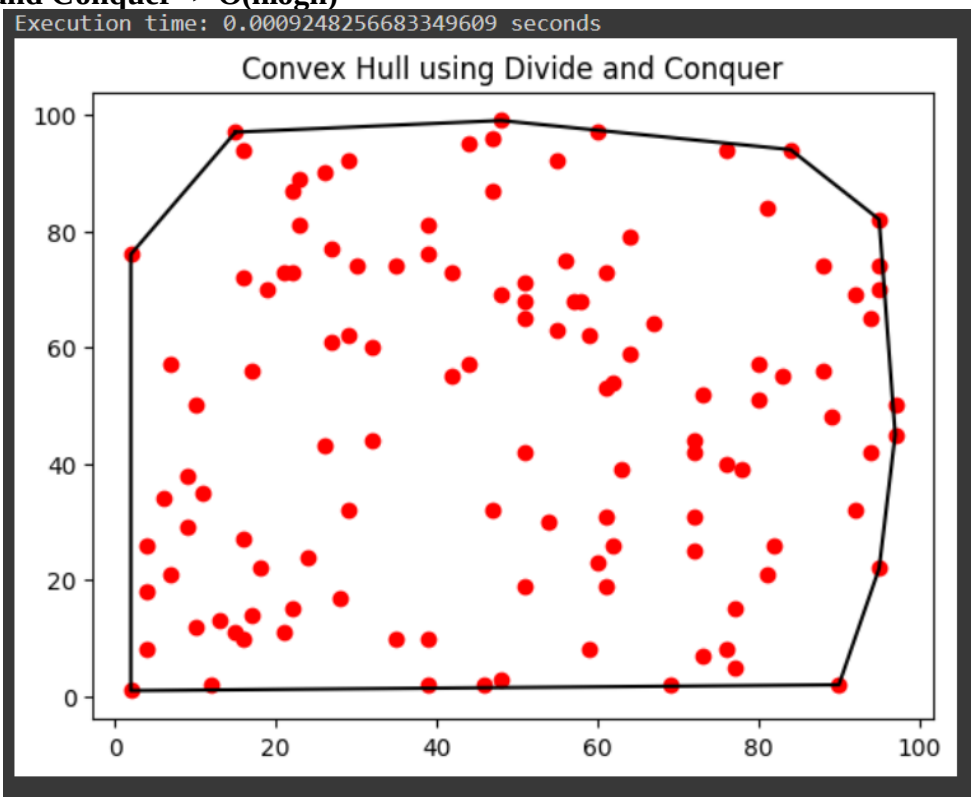
1. O(nlogn)-> Icremental Algorithm



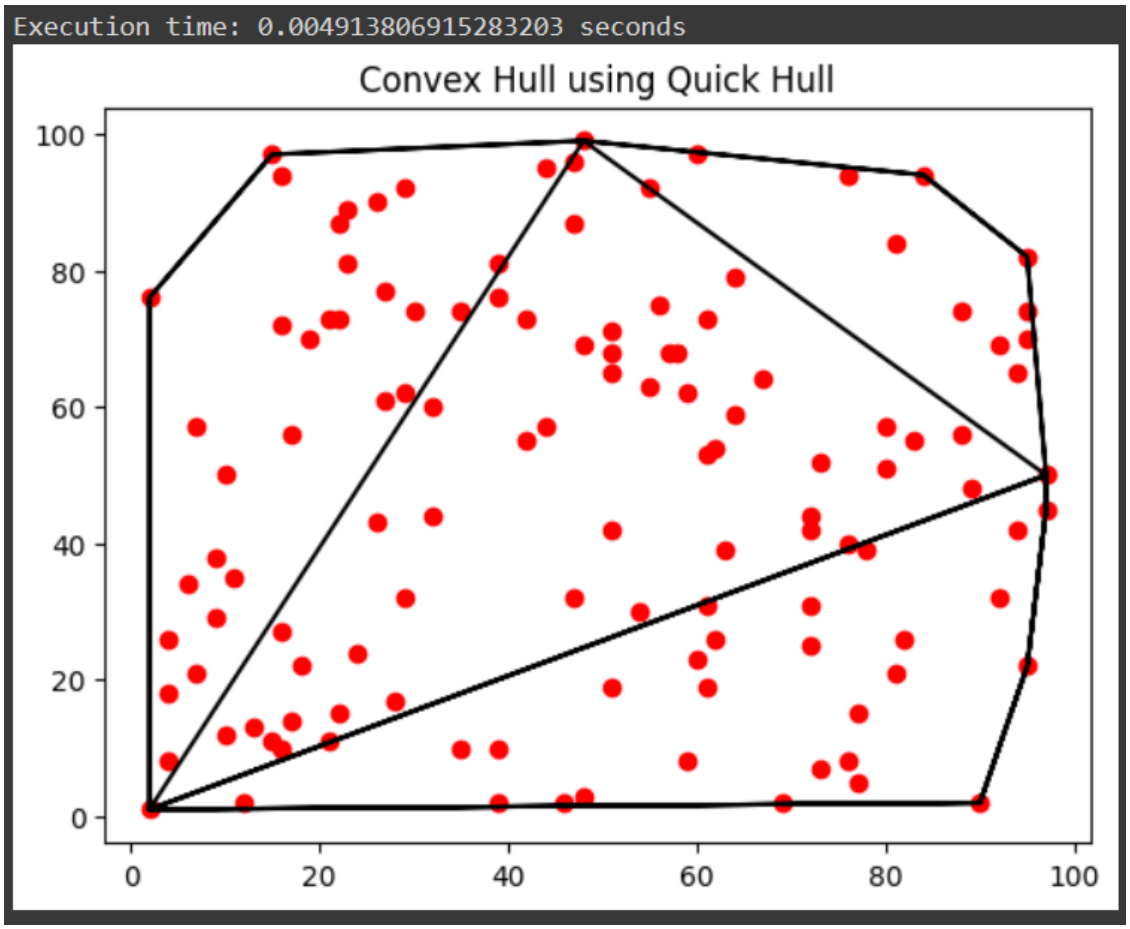
2. **Gift Wrapping -> O(nh)**



3. **Divide and Conquer -> O(nlogn)**



5. **Quickhull ->O(nlogn)**



Algorithm	**Execution time for 120 points**
Incremental O(nlogn)	c09545898438 sec
Gift Wrapping O(nh)	0.003947019577026367 sec
Divide and Conquer O(nlogn)	0.0009248256683349609 sec
Quickhull O(nlogn)	0.004913806915283203 sec

	30 points	60 points	90 points	120 points
Incremental	0.000224828	0.00044608	0.00044894	0.00044894
Gift Wrapping	0.00223708	0.00355291	0.004140138	0.003947019
Divide and Conquer	0.000459194	0.000459194	0.000621080	0.00092482
Quickhull	0.002568483	0.002134799	0.003542184	0.0049138069

- Incremental:Έχει πολύ χαμηλό και σταθερό χρόνο εκτέλεσης, με μικρή αύξηση καθώς αυξάνονται τα σημεία, παραμένοντας σημαντικά κάτω από τους άλλους αλγόριθμους.

- Gift Wrapping:Δείχνει σαφή αύξηση στον χρόνο εκτέλεσης με περισσότερα σημεία, υποδεικνύοντας ότι έχει υψηλότερη υπολογιστική πολυπλοκότητα.

- Divide and Conquer:Ξεκινά με καλή απόδοση, αλλά ο χρόνος αυξάνεται πιο απότομα με την αύξηση των σημείων.

- Quickhull:Έχει αρχικά χρόνο εκτέλεσης παρόμοιο με τον Gift Wrapping, αλλά οι χρόνοι εκτέλεσης παρουσιάζουν μεγαλύτερες διακυμάνσεις και αυξάνονται σημαντικά με τα 120 σημεία.

Άρα αν συγκρίνουμε τα παραπάνω αποτελέσματα έχουμε:

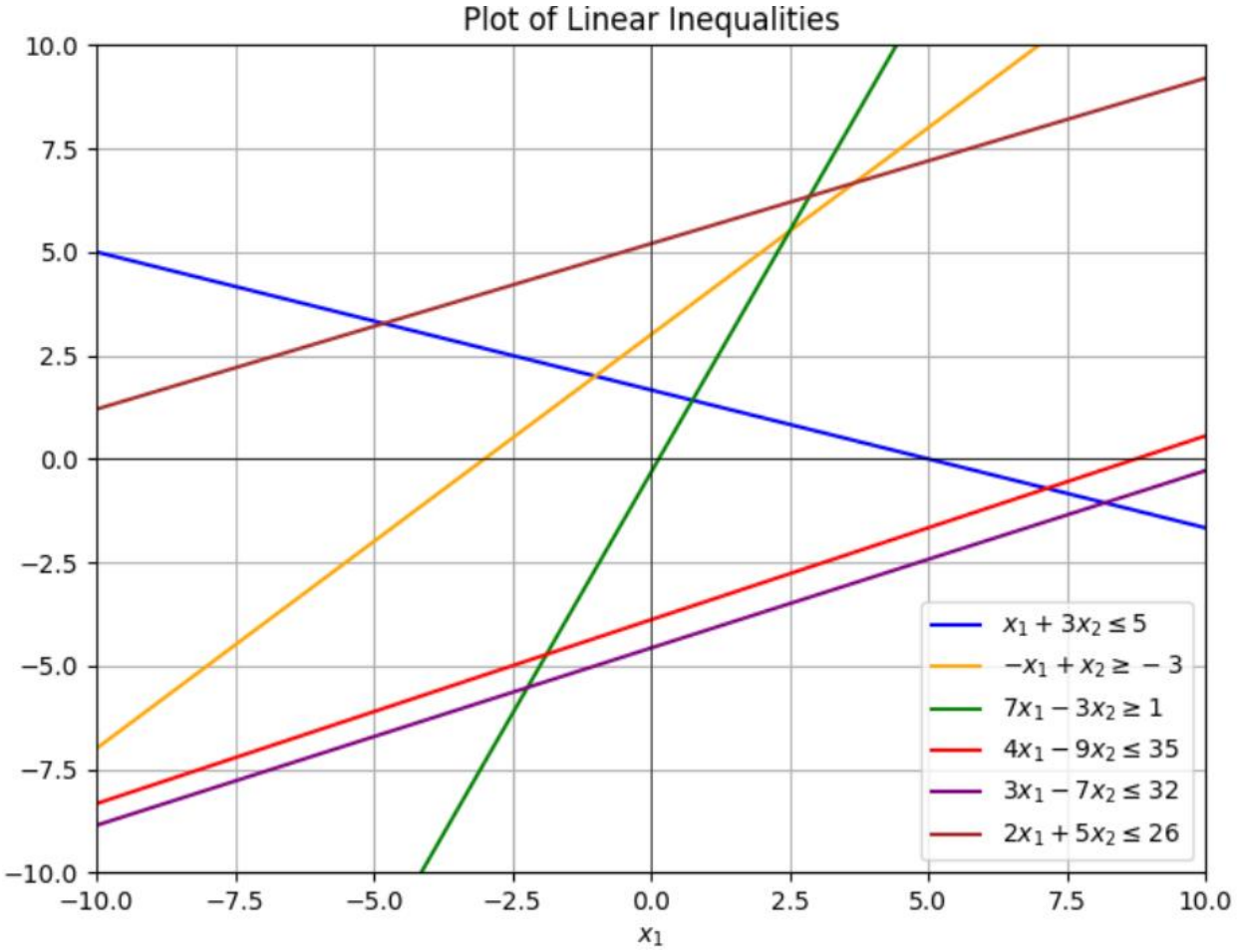
1. Ο Incremental είναι ο πιο αποδοτικός αλγόριθμος σε όλα τα σημεία, με τον χαμηλότερο χρόνο εκτέλεσης.
2.
3. Ο Divide and Conquer είναι αρχικά ισχυρός, αλλά η αύξηση του χρόνου είναι πιο απότομη σε σύγκριση με τον Incremental.
4. Οι Gift Wrapping και Quickhull δείχνουν υψηλότερους χρόνους εκτέλεσης, με τον Quickhull να έχει τη χειρότερη απόδοση στα 120 σημεία.

Ο Incremental φαίνεται να είναι ο πιο κατάλληλος αλγόριθμος για αυξανόμενο αριθμό σημείων, κάνοντάς τον ιδανικό για περιπτώσεις όπου οι αριθμοί σημείων μπορεί να διαφέρουν σημαντικά.Οι Gift Wrapping και Quickhull μπορεί να γίνουν «στενωπή» σε εφαρμογές που απαιτούν υψηλή απόδοση καθώς αυξάνονται τα σημεία.

Ακόμα σχηματικά οι αλγόριθμοι δεν διαφέρουν σχηματικά μεταξύ τους, αφού όλοι έχουν το ίδιο σκοπό.

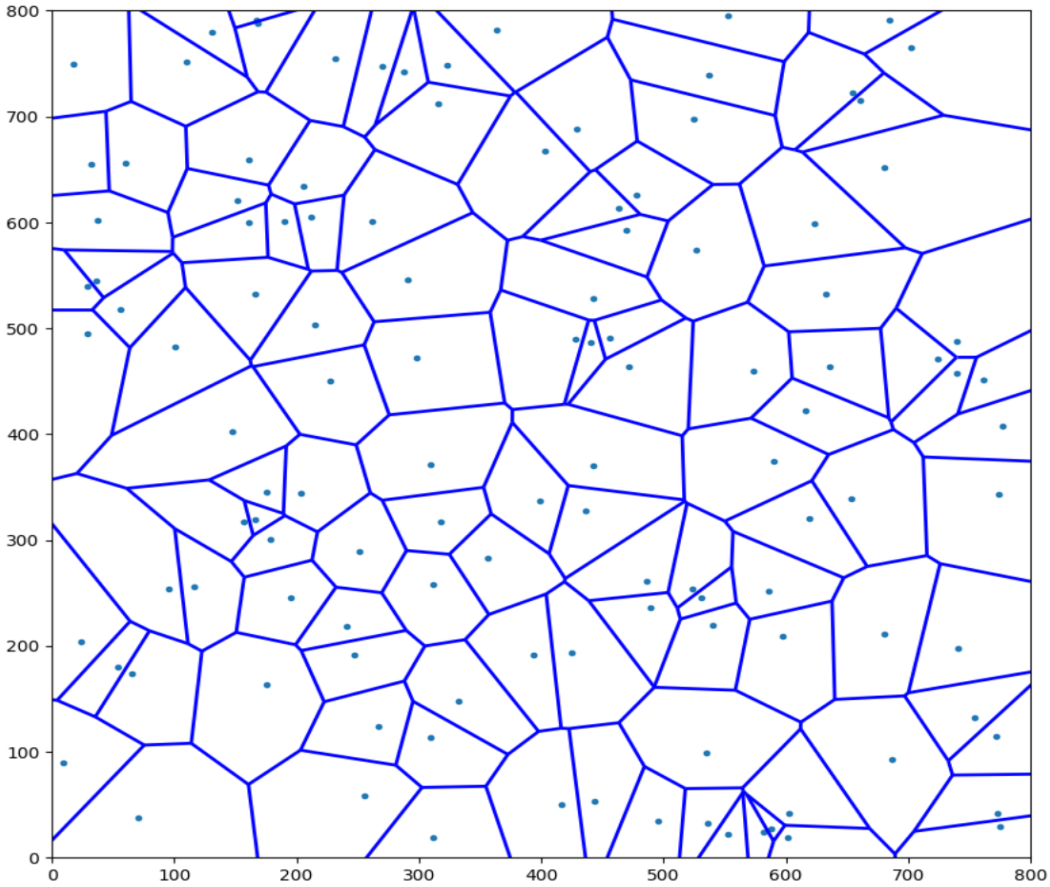
Part_2

Με τους συγκεκριμένους υπολογισμούς δεν υπάρχει εφικτή περιοχή λύσεων



Part_3

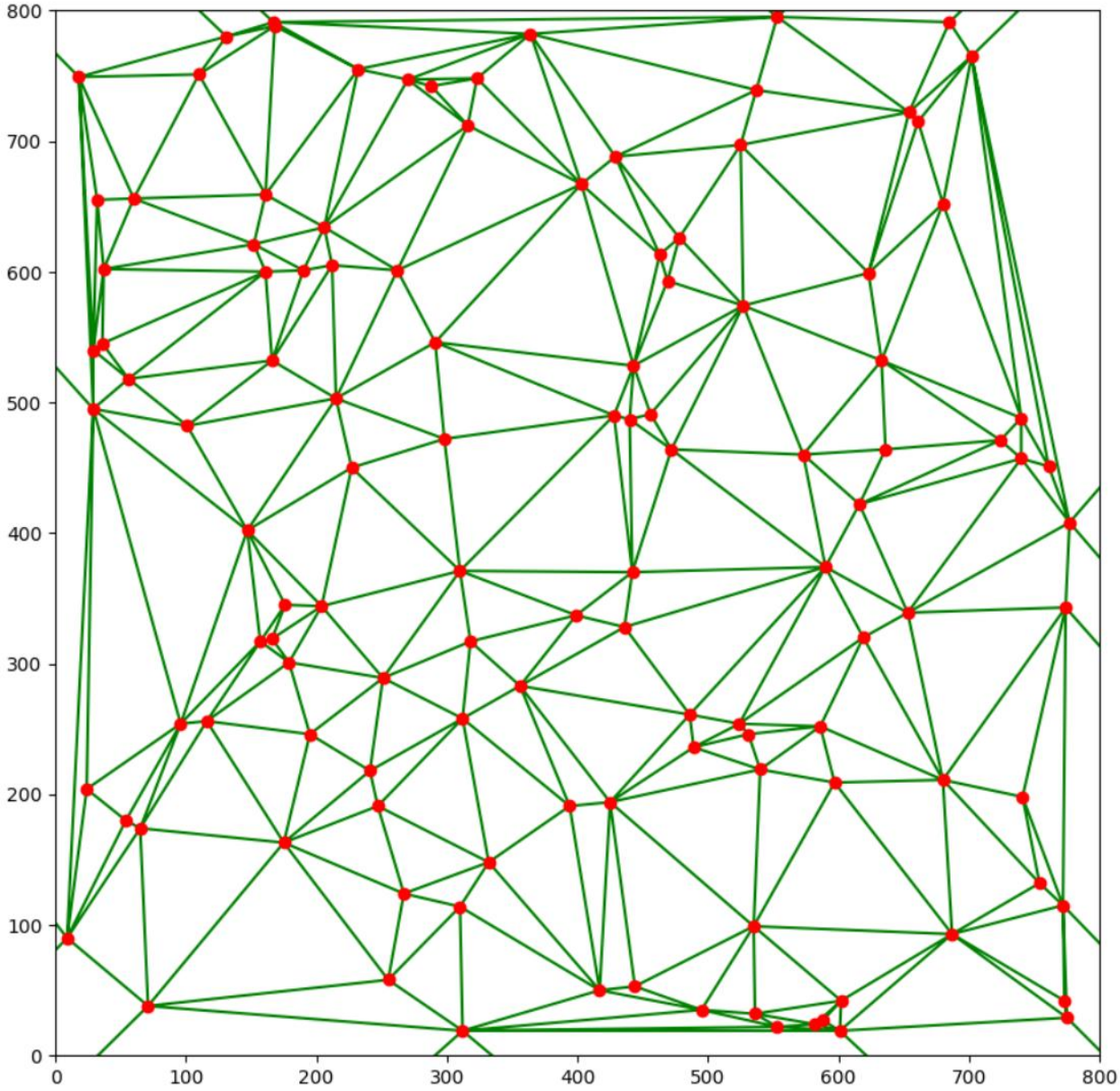
- **Voronoi Diagram**



<u>Complexity of Voronoi Diagram -> O(nlogn)</u>

Όσο αυξάνεται το n, η πολυπλοκότητα του αλγορίθμου μεγαλώνει λογαριθμικά, πράγμα που σημαίνει ότι ο χρόνος εκτέλεσης αυξάνεται, αλλά όχι γραμμικά. Ωστόσο, η αύξηση του n μπορεί να προκαλέσει επίσης αύξηση του αριθμού των περιοχών στο διάγραμμα, κάνοντάς το πιο περίπλοκο. Καθώς κάθε σημείο στο διάγραμμα αντιστοιχεί σε μία περιοχή Voronoi, που περιλαμβάνει όλα τα σημεία του χώρου που είναι πιο κοντά σε αυτό το σημείο από οποιοδήποτε άλλο.

- **Delaunay Triangulation**



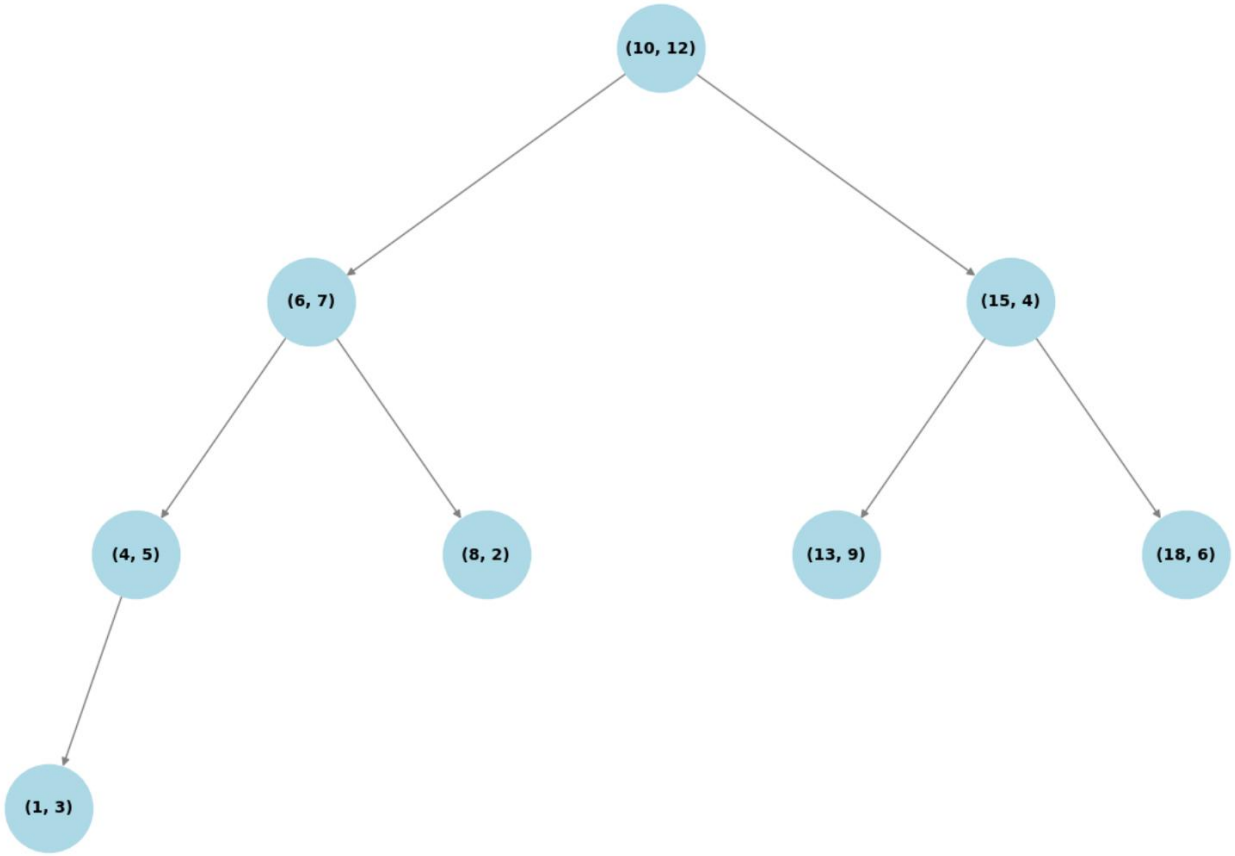
Complexity of Delaunay Triangulation -> O(nlogn)

Η πολυπλοκότητα αυξάνεται λογαριθμικά με την αύξηση του n. Ωστόσο, καθώς τα σημεία αυξάνονται, ο αριθμός των τριγώνων που παράγονται στην τριγωνοποίηση, μπορεί να αυξηθεί πιο γρήγορα, επηρεάζοντας τον χρόνο επεξεργασίας. Όταν προστίθονται σημεία, η τριγωνοποίηση αναδιαρθρώνεται για να διατηρήσει τις ιδιότητες της Delaunay, όπως η

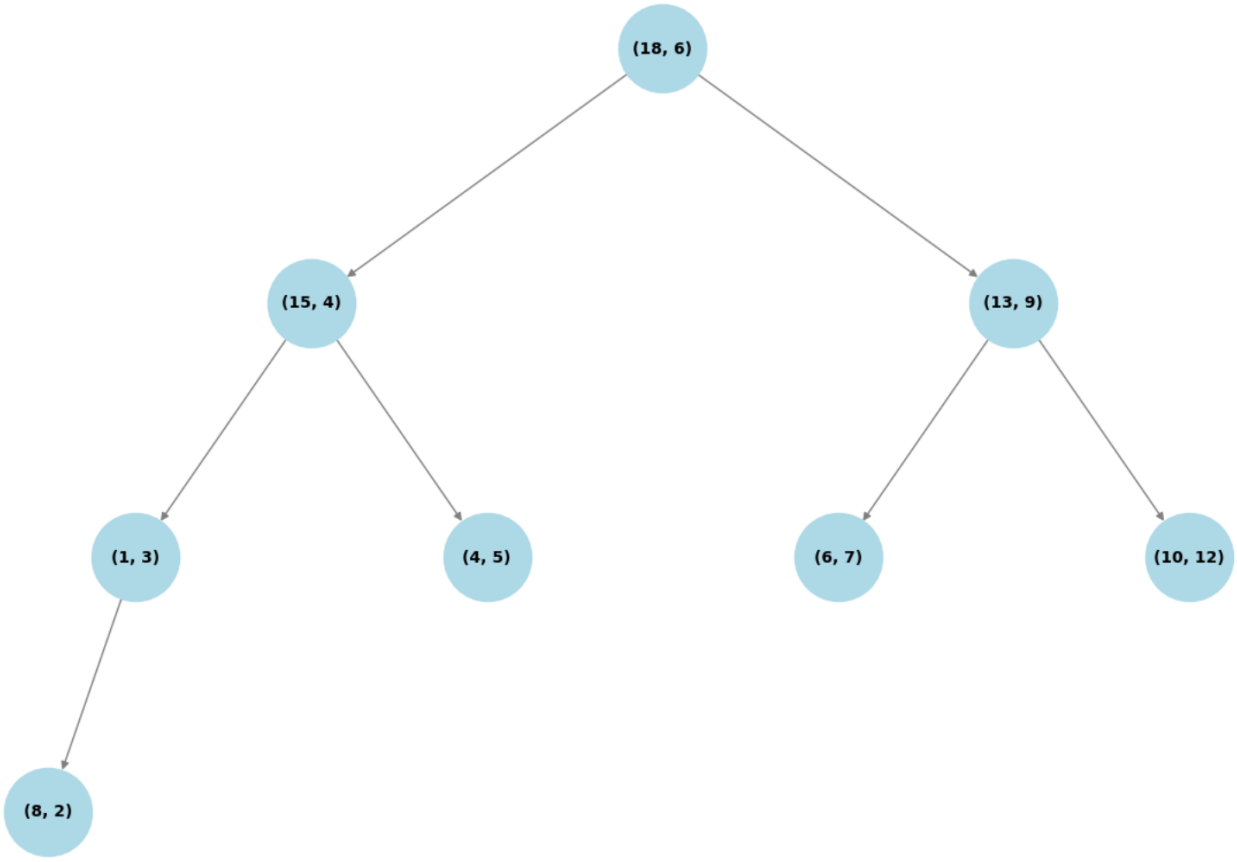
μεγιστοποίηση των γωνιών των τριγώνων. Αυτό σημαίνει ότι μπορεί να χρειαστεί να απορριφθούν ή να αναδιαμορφωθούν υπάρχοντα τρίγωνα.

Part_4

For a dataset of certain points : [(1, 3), (4, 5), (6, 7), (8, 2), (10, 12), (13, 9), (15, 4), (18, 6)]
The following tree is arranged base on x



- Each of the nodes has its own y-subtree
- This is the y-subtree of the root- node (Secondary Tree)

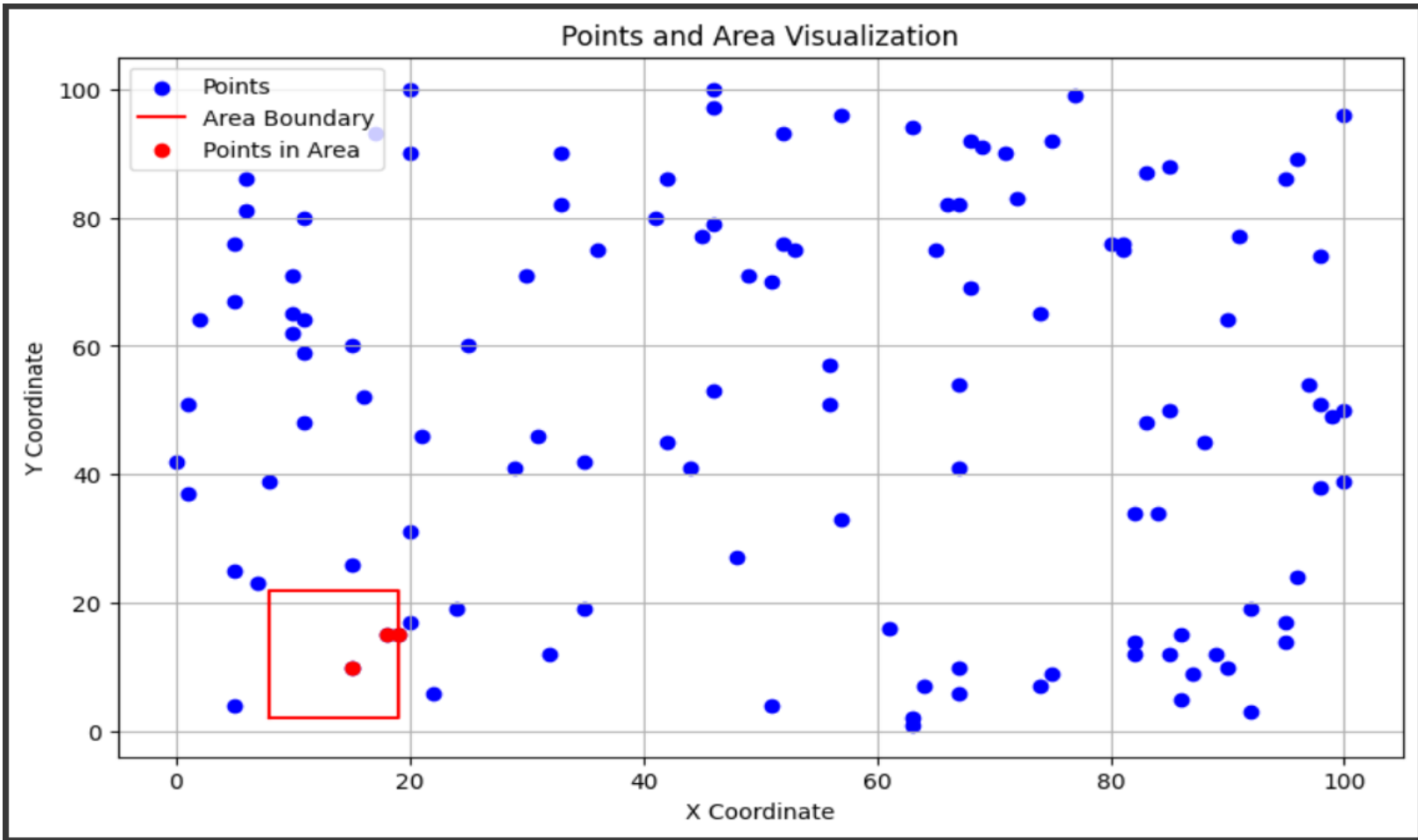


- For this dataset: [(81, 76), (35, 42), (88, 45), (29, 41), (32, 12), (77, 99), (21, 46), (15, 60), (96, 24), (49, 71)]

- Using this search range function on the x tree ,with these restrictions:
points_in_range = range_tree.range_search(range_tree.root, xmin=8, xmax=19, ymin=2, ymax=22)

- We get the following results:
[(15, 10), (19, 15), (18, 15)]

- With the plot we get the same results




```
Points within the red area:
(15, 10)
(18, 15)
(19, 15)
```