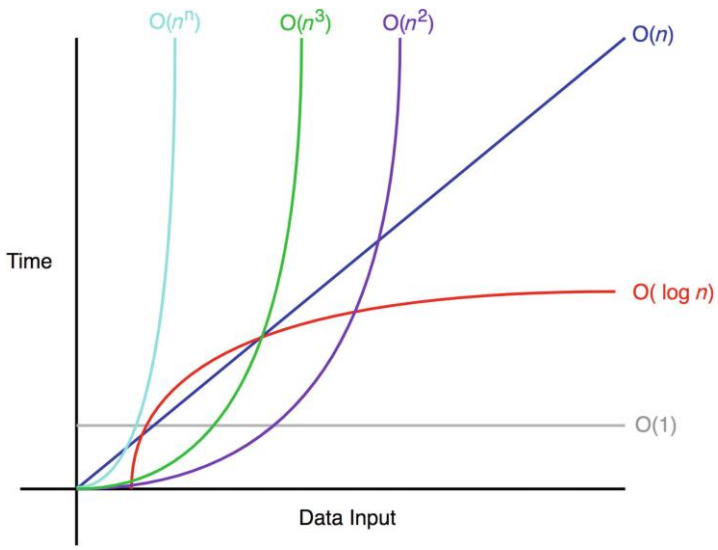
3Δ ΥΠΟΛΟΓΙΣΤΙΚΗ ΓΕΩΜΕΤΡΙΑ ΚΑΙ ΟΡΑΣΗ

2η Εργαστηριακή Άσκηση



up1092813
Νικόλαος Γέροντας

Πίνακας περιεχομένων

Ασκήσεις 1 → 3 2
Άσκηση 4 2
Άσκηση 5 3
Άσκηση 6 5

Ασκήσεις 1 → 3

Συμπληρώστε τη συνάρτηση **CH_brute_force**.

a. Κατανοήστε τον απλό αλγόριθμο *εξαντλητικής αναζήτησης* για την εύρεση του κυρτού περιβλήματος ενός συνόλου σημείων.
b. Για κάθε ζεύγος σημείων, ελέγξτε αν το ευθύγραμμο τμήμα που ορίζουν ανήκει στο κυρτό περίβλημα.

Συμπληρώστε την συνάρτηση **CH_graham_scan**.

a. Κατανοήστε τον αλγόριθμο *graham scan* για την εύρεση του κυρτού περιβλήματος ενός συνόλου σημείων.
b. Υλοποιήστε την συνάρτηση **orientation**.
c. Ταξινομήστε τα σημεία ανάλογα με την γωνία τους, αριστερόστροφα (αντίστροφα των δεικτών του ρολογιού).
d. Υλοποιήστε την προσθαφαίρεση σημείων στην στοίβα.

Συμπληρώστε την συνάρτηση **CH_quickhull**.

a. Κατανοήστε τον greedy αλγόριθμο *quickhull* για την εύρεση του κυρτού πολυγώνου ενός συνόλου σημείων.
b. Γράψτε κώδικα για την εύρεση του μακρινότερου σημείου από μια γραμμή.
c. Υλοποιήστε την συνάρτηση **separate_points_by_line** για τον διαχωρισμό ενός συνόλου σημείων σε αριστερά και δεξιά μιας γραμμής.

Απάντηση

Οι αλγόριθμοι αναλύθηκαν και υλοποιήθηκαν στο πλαίσιο του εργαστηρίου. Για την καλύτερη κατανόηση της διαδικασίας, αναπτύχθηκε επιπλέον κώδικας για τη γραφική απεικόνιση και δημιουργία του κυρτού περιβλήματος με τη μέθοδο brute force (βλ. αρχεία: test_lab2_animation.py & lab2_animation_CH_bf.mp4). Παράλληλα, οι αρχικές δοκιμές σχετικά με τις animation δυνατότητες της βιβλιοθήκης vvrpywork πραγματοποιήθηκαν στο αρχείο test_animation.py.

Άσκηση 4

Υλοποιήστε την συνάρτηση **CH_jarvis_march**.

a. Κατανοήστε τον αλγόριθμο *jarvis march/gift wrapping* για την εύρεση του κυρτού πολυγώνου ενός συνόλου σημείων.
b. Υλοποιήστε το σώμα της συνάρτησης χρησιμοποιώντας τη συνάρτηση **orientation**.

Απάντηση

a. Ο αλγόριθμος (στην υλοποίησή μου) ξεκινά από το αριστερότερο σημείο (το σημείο με τη μικρότερη x-συντεταγμένη) και **επαναλαμβανόμενα επιλέγει το επόμενο σημείο του περιβλήματος**, δηλαδή το σημείο που σχηματίζει τη μικρότερη δυνατή αριστερόστροφη γωνία με τη γραμμή από το τρέχον σημείο. Αυτό επαναλαμβάνεται **μέχρι να γυρίσουμε πίσω στο αρχικό σημείο**.

b. Χρησιμοποιώντας το εξωτερικό γινόμενο [δηλαδή, την εντολή numpy.cross()], υλοποιούμε την συνάρτηση orientation.

Διανυσματικό γινόμενο - Βικιπαίδεια (η χρήση του σε 2D, δίνει βαθμωτό αποτέλεσμα)

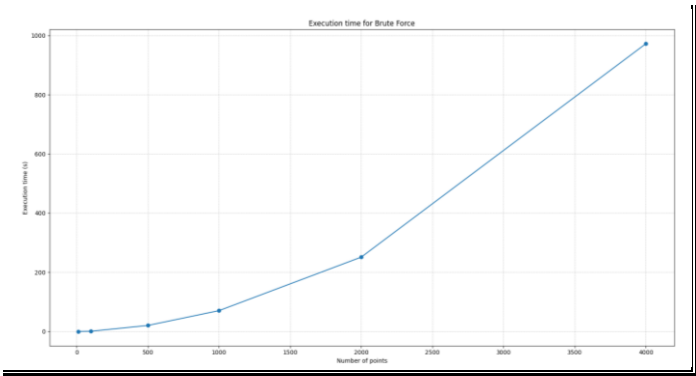
Άσκηση 5

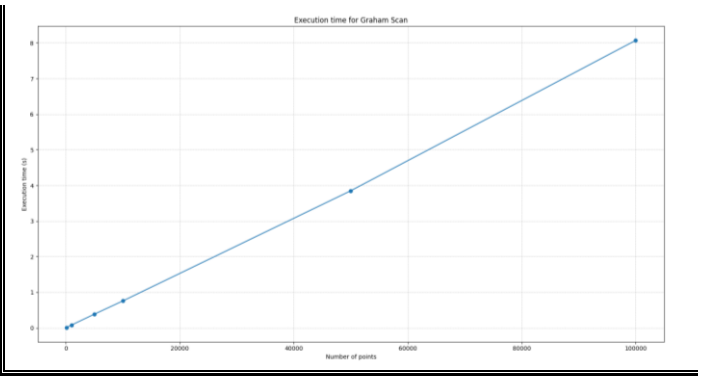
Μετρήστε το χρόνο εκτέλεσης των τεσσάρων αλγορίθμων υπολογισμού κυρτού πολυγώνου, για διαφορετικά πλήθη σημείων.

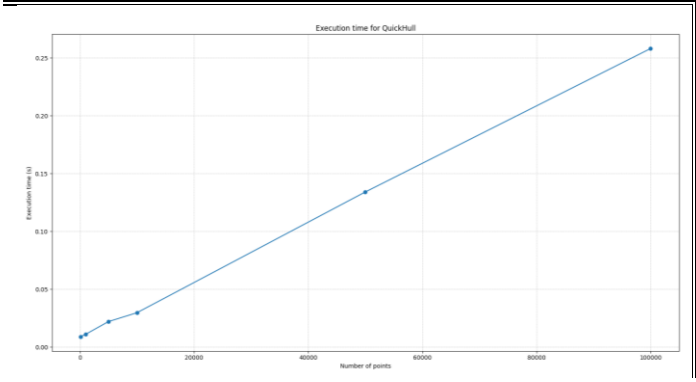
a. Απεικονίστε σε γραφική παράσταση το χρόνο εκτέλεσης συναρτήσει του πλήθους των σημείων για όλους τους αλγορίθμους. Χρησιμοποιήστε τη βιβλιοθήκη matplotlib.pyplot, ή οποιοδήποτε άλλο εργαλείο της επιλογής σας (Excel, MATLAB, κλπ.).
b. Τι παρατηρείτε; Είναι τα αποτελέσματα όπως τα περιμένατε; Σχολιάστε.

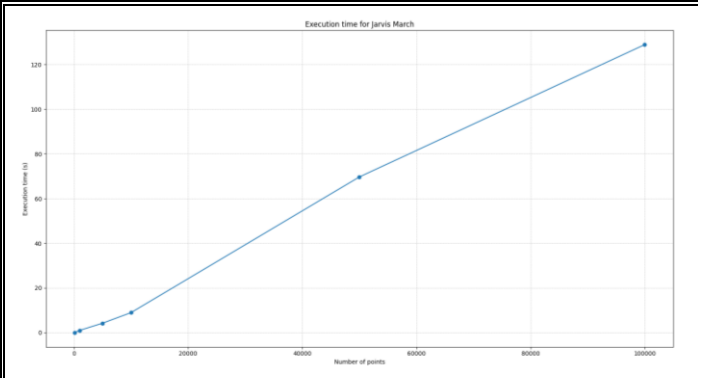
Απάντηση

a. Χρησιμοποιώντας το πρότυπο της συνάρτησης/λογική που είχα υλοποιήσει στο μάθημα των αλγορίθμων και την βιβλιοθήκη: Matplotlib, έχουμε:









Αναλυτικότερα:

```
Function's name:  CH_brute_force
Number of points: 10
Execution time:   0.009998083114624023
Function's name:  CH_brute_force
Number of points: 100
Execution time:   0.704002857208252
Function's name:  CH_brute_force
Number of points: 500
Execution time:   20.25030779838562
Function's name:  CH_brute_force
Number of points: 1000
Execution time:   70.30099892616272
Function's name:  CH_brute_force
Number of points: 2000
Execution time:   251.211177110672
Function's name:  CH_brute_force
Number of points: 4000
Execution time:   972.4416890144348
```

```
Function's name:  CH_graham_scan
Number of points: 100
Execution time:   0.012002706527709961
Function's name:  CH_graham_scan
Number of points: 1000
Execution time:   0.082000732421875
Function's name:  CH_graham_scan
Number of points: 5000
Execution time:   0.3859882354736328
Function's name:  CH_graham_scan
Number of points: 10000
Execution time:   0.7580149173736572
Function's name:  CH_graham_scan
Number of points: 50000
Execution time:   3.851616144180298
Function's name:  CH_graham_scan
Number of points: 100000
Execution time:   8.073997020721436
```

```
Function's name:  CH_quickhull
Number of points: 100
Execution time:   0.009002208709716797
Function's name:  CH_quickhull
Number of points: 1000
Execution time:   0.0110015869140625
Function's name:  CH_quickhull
Number of points: 5000
Execution time:   0.022002458572387695
Function's name:  CH_quickhull
Number of points: 10000
Execution time:   0.02967691421508789
Function's name:  CH_quickhull
Number of points: 50000
Execution time:   0.1340022087097168
Function's name:  CH_quickhull
Number of points: 100000
Execution time:   0.2579977512359619
```

```
Function's name:  CH_jarvis_march
Number of points: 100
Execution time:   0.04899787902832031
Function's name:  CH_jarvis_march
Number of points: 1000
Execution time:   1.0130057334899902
Function's name:  CH_jarvis_march
Number of points: 5000
Execution time:   4.219999551773071
Function's name:  CH_jarvis_march
Number of points: 10000
Execution time:   8.962952375411987
Function's name:  CH_jarvis_march
Number of points: 50000
Execution time:   69.686776638031
Function's name:  CH_jarvis_march
Number of points: 100000
Execution time:   128.92703914642334
```

b. Βάση των παραπάνω αποτελεσμάτων, παρατηρούμε ότι:

Ο αλγόριθμος Brute Force εμφανίζει **μέση πολυπλοκότητα O(n^2)**, ενώ η **θεωρητική του πολυπλοκότητα {worst case} είναι O(n^3)**! Η απόκλιση αυτή ευθύνεται στην χρήση του *for ... else construct* της Python κατά την υλοποίηση, η οποία βελτιώνει την απόδοση σε ορισμένες περιπτώσεις. Αναλυτικότερα, ισχύει ο τύπος:

$$\boxed{\frac{n_1^2}{t_1} = \frac{n_2^2}{t_2}} \; [O(n^2)]$$

όπου n_i: το μέγεθος του point cloud &
t_i: ο χρόνος εκτέλεσης/διεκπεραίωσης του αλγορίθμου

✍ **π.χ.:**

$$\frac{500^2}{20.25} = \frac{1000^2}{t_2} \Longrightarrow t_2 = 81 \text{ sec } \approx 70.3 \text{ sec } [\text{πραγματική τιμή}]$$

Παράλληλα, για τον αλγόριθμο Graham Scan, παρατηρούμε ότι η **θεωρητική του πολυπλοκότητα {O(n logn)} συμφωνεί με την πραγματική κατά την εκτέλεση**! Αντίστοιχα, ισχύει ένας παρόμοιος τύπος με τον προηγούμενο, οδηγώντας στο ακόλουθο παράδειγμα:

$$\frac{50000 \cdot \log(50000)}{3.85} = \frac{100000 \cdot \log(100000)}{t_2} \Longrightarrow t_2 = 8.19 \text{ sec} \approx 8.07 \text{ sec } [\text{πραγματική τιμή}]$$

Επίσης, για τον αλγόριθμο Quickhull, παρατηρούμε ότι, αν και η **θεωρητική του πολυπλοκότητα {worst case} είναι O(n^2), στην πράξη εμφανίζεται να ακολουθεί πολυπλοκότητα O(n logr)**! Συγκεκριμένα, αυτό συμβαίνει επειδή η ακρίβεια της εισόδου είναι περιορισμένη σε O(log n) bits, καθώς χρησιμοποιούμε την βιβλιοθήκη NumPy που ακολουθεί τα data types της C, με προεπιλεγμένη επιλογή να είναι το float64 (καθώς χρησιμοποιώ 64-bit υπολογιστικό σύστημα), δηλαδή, 64 bits περιορισμός!

Quickhull - Wikipedia

✍ **π.χ.:**

$$\frac{50000 \cdot \log(\sim \log(50000))}{0.134} = \frac{100000 \cdot \log(\sim \log(100000))}{t_2} \Longrightarrow t_2 = 0.28 \text{ sec } \approx 0.26 \text{ sec } [\text{πραγματική τιμή}]$$

Τέλος, για τον αλγόριθμο Jarvis march, παρατηρούμε ότι η **θεωρητική του πολυπλοκότητα {O(nh)} συμφωνεί με την πραγματική κατά την εκτέλεση**!

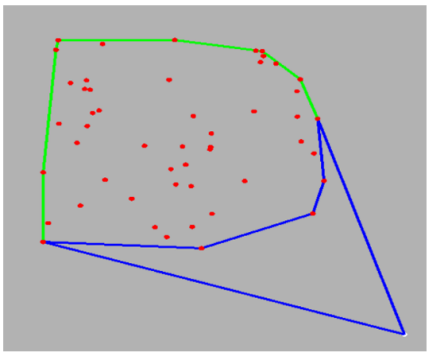
Gift wrapping algorithm - Wikipedia

Άσκηση 6

Σχεδιάστε σημείο που ακολουθεί την κίνηση του ποντικιού. *(Υλοποιημένο)*

a. Αλλάξτε το χρώμα του σημείου ώστε να διαφέρει εάν το ποντίκι βρίσκεται εντός ή εκτός του κυρτού περιβλήματος.

b. Στην περίπτωση σημείου εκτός του κυρτού περιβλήματος: Σχεδιάστε τα ευθύγραμμα τμήματα από το σημείο έως τις ακραίες κορυφές του πολυγώνου που είναι ορατές και επιπλέον χρωματίστε τις ακμές του πολυγώνου που είναι ορατές από το σημείο (βλ. εικόνα δίπλα).



c. Η λύση που δώσατε λειτουργεί σωστά για όλους τους αλγορίθμους σχεδιασμού του κυρτού περιβλήματος; Προσπαθήστε να το ερμηνεύσετε.

Απάντηση

Στην κλάση Polygon2D έχουν προστεθεί 2 νέα γνωρίσματα:

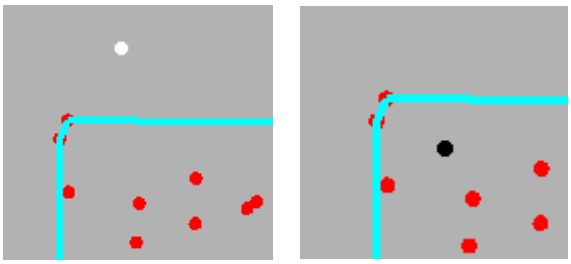
- ✓ **convex_hull_edges**: Λίστα της Python που περιέχει όλες τις πλευρές του πολυγώνου (ως αντικείμενα τύπου Line2D).
- ✓ **convex_hull_vertices**: Λίστα της Python που περιέχει όλες τις κορυφές του πολυγώνου (ως αντικείμενα τύπου Point2D).

a. Η απόφαση για το αν ο κέρσορας βρίσκεται εντός ή εκτός του κυρτού περιβλήματος, υλοποιήθηκε με 2 τρόπους. Ο 1ος τρόπος βασίζεται στον προσανατολισμό των πλευρών του πολυγώνου:

i. Ο αλγόριθμος ελέγχει αν το σημείο του κέρσορα βρίσκεται στα «δεξιά/αριστερά» {μία πλευρά} όλων των πλευρών του πολυγώνου. Για να είναι ο έλεγχος έγκυρος, πρέπει οι πλευρές να έχουν σχηματιστεί με «συνεπή» κυκλική φορά (π.χ. όλες με δεξιόστροφη ή όλες με αριστερόστροφη φορά)!

ii. Στον 2ο τρόπο, θεωρούμε ότι από το σημείο του κέρσορα σχηματίζονται ευθείες προς κάθε κορυφή του πολυγώνου. **Αν οποιαδήποτε από αυτές τις ευθείες τέμνει μία πλευρά του πολυγώνου** (χωρίς να έχει κοινό σημείο κορυφής), τότε θεωρούμε ότι **το σημείο βρίσκεται εκτός του πολυγώνου**. Αν καμία από τις ευθείες δεν τέμνει το περίβλημα, τότε το σημείο θεωρείται εντός του πολυγώνου. **Δεν απαιτεί κάποια ειδική προϋπόθεση (σε αντίθεση με τον 1ο τρόπο) και επομένως λειτουργεί ανεξάρτητα από τον αλγόριθμο που χρησιμοποιήθηκε για την εύρεση του κυρτού περιβλήματος!**

Έτσι, με τη χρήση είτε της μεθόδου is_point_inside_polygon είτε της is_point_inside_polygon_intersection, σε συνδυασμό με τις συναρτήσεις on_mouse_press και on_mouse_release, μπορούμε να αλλάζουμε το χρώμα του σημείου του κέρσορα ανάλογα με τη σχετική του θέση ως προς το κυρτό περίβλημα.



Με χρήση του αλγορίθμου Brute Force

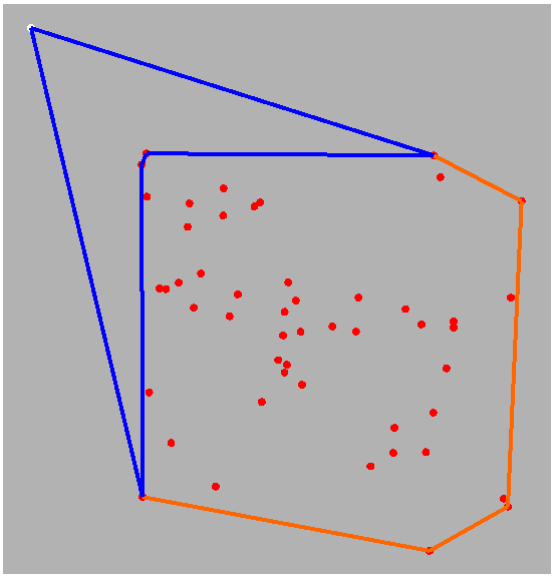
Για τους τέσσερις αλγορίθμους εύρεσης κυρτού περιβλήματος που υλοποιήθηκαν, οι 2 προσεγγίσεις (με βάση τη θέση του σημείου) καταλήγουν στο ίδιο αποτέλεσμα!

b. Η υλοποίηση της άσκησης 6 b πραγματοποιήθηκε με 2 διαφορετικούς τρόπους, οι οποίοι βασίζονται στις μεθόδους is_point_inside_polygon και is_point_inside_polygon_intersection, αντίστοιχα:

i. Η **1η προσέγγιση** (**exercise6b_solution2**) **επεκτείνει τη λογική της μεθόδου is_point_inside_polygon**, η οποία βασίζεται στη γεωμετρική σχέση «δεξιά/αριστερά» ως προς τις πλευρές του πολυγώνου. Συγκεκριμένα, για κάθε πλευρά του κυρτού περιβλήματος, ελέγχεται αν το εξωτερικό σημείο βρίσκεται δεξιά της πλευράς (με χρήση της μεθόδου isOnRight). Μετά, οι πλευρές που δεν έχουν το σημείο στα δεξιά τους θεωρούνται ορατές, και χρωματίζονται με διαφορετικό χρώμα. Στη συνέχεια, προσδιορίζονται οι 2 ακραίες πλευρές (πρώτη και τελευταία στη λίστα ορατών πλευρών), και σχεδιάζονται τα ευθύγραμμα τμήματα από το εξωτερικό σημείο προς τις αντίστοιχες κορυφές.

ii. Η **2η υλοποίηση** **επεκτείνει τη λογική της is_point_inside_polygon_intersection**, η οποία δεν βασίζεται σε προσανατολισμό των πλευρών, αλλά σε καθαρά γεωμετρικό έλεγχο τομής ευθειών. Συγκεκριμένα, για κάθε κορυφή του κυρτού περιβλήματος, δημιουργείται 1 ευθύγραμμο τμήμα από το σημείο του κέρσορα προς την κορυφή. Αν το τμήμα αυτό τέμνει κάποια πλευρά του πολυγώνου (χωρίς κοινό άκρο), η αντίστοιχη κορυφή θεωρείται μη ορατή. Όσες κορυφές δεν έχουν εμπόδια θεωρούνται ορατές, και βάσει αυτών:

- Σχεδιάζονται οι ακμές που ενώνονται μόνο μεταξύ ορατών κορυφών.
- Υπολογίζεται το ζεύγος ορατών κορυφών που σχηματίζει τη μέγιστη γωνία ως προς το εξωτερικό σημείο, και σχεδιάζονται τα αντίστοιχα ευθύγραμμα τμήματα.



c. Παρατηρούμε ότι ο **2ος τρόπος υλοποίησης**, λειτουργεί σωστά ανεξάρτητα από τον τρόπο εύρεσης του κυρτού περιβλήματος! Αντίθετα, ο **1ος τρόπος**, εμφανίζει πρόβλημα λόγο του παρακάτω σημείου κώδικα:

```
# Πρόσθεσε τις γραμμές που συνδέουν το σημείο με τα άκρα του ορατού πολυγώνου
temp = [np.where(is_right)[0], np.where(~is_right)[0]]
for visible_indices in temp:
    first_edge = self.poly.convex_hull_edges[visible_indices[0]]
    last_edge = self.poly.convex_hull_edges[visible_indices[-1]]

    first_visible_vertex = (first_edge.x1, first_edge.y1)
    last_visible_vertex = (last_edge.x2, last_edge.y2)

    if first_visible_vertex != last_visible_vertex:
        break;
```

Αναλυτικότερα, το συγκεκριμένο σημείο του κώδικα **βασίζεται στη φορά των ακμών του κυρτού περιβλήματος**, και άρα η σωστή λειτουργία εξαρτάται από το αν έχει γίνει reorderIfNecessary=True λογική κατά την κατασκευή του!