[](http://www.teilar.gr/)

**Αλγόριθμοι και Στοιχεία Πολυπλοκότητας.**

**Ενότητα 2:** Πολυπλοκότητα.

Διδάσκων: Ηλίας Κ Σάββας, Αναπληρωτής Καθηγητής.

Τμήμα Μηχανικών Πληροφορικής, Τεχνολογικής Εκπαίδευσης.

**Άδειες χρήσης.**

* Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons (C C). **Αναφορά δημιουργού (B Y), Μη εμπορική χρήση (N C), Μη τροποποίηση (N D), 3.0, Μη εισαγόμενο.**
* Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

[](http://creativecommons.org/licenses/by-nc-nd/3.0/deed.el)

**Χρηματοδότηση.**

* Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
* Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

[](http://www.edulll.gr/)

# Σκοποί ενότητας.

Ο αναγνώστης να μπορεί να:

1) αντιλαμβάνεται πλήρως τις συναρτήσεις πολυπλοκότητας,

2) επιλύει προβλήματα σύνθετα, χρησιμοποιώντας απλές μεθόδους (brute force).

ΠΕΡΙΕΧΌΜΕΝΑ ΕΝΌΤΗΤΑΣ.

[Σκοποί ενότητας. 2](#_Toc368172497)

[ΑΛΓΌΡΙΘΜΟΙ 3](#_Toc368172498)

[ΠΡΟΓΡΆΜΜΑΤΑ 3](#_Toc368172499)

[ΣΧΉΜΑΤΑ 4](#_Toc368172500)

[2. ΠΟΛΥΠΛΌΚΟΤΗΤΑ. 4](#_Toc368172501)

[2.1 Ορισμοί. 4](#_Toc368172502)

[2.2 Αλγόριθμοι Brute-Force. 7](#_Toc368172503)

[2.3 Δισδιάστατα μέγιστα. 8](#_Toc368172504)

[2.4 Το πρόβλημα των ελαχίστων αποστάσεων (closest-pair problem). 12](#_Toc368172505)

[Λύση. 12](#_Toc368172506)

[2.5 Εύρεση αλφαριθμητικών. 14](#_Toc368172507)

[2.6 Υπολογιστική γεωμετρία. 16](#_Toc368172508)

# ΑΛΓΌΡΙΘΜΟΙ

|  |  |
| --- | --- |
| ***Αλγόριθμος 1:*** *Δισδιάστατα μέγιστα – Αλγόριθμος Brute-Force.* | 9 |
| ***Αλγόριθμος 2****: Δισδιάστατα μέγιστα – Δεύτερος αλγόριθμος.* | 10 |
| ***Αλγόριθμος 3****: Πρόβλημα ελαχίστων αποστάσεων (Brute-Force).* | 12 |
| ***Αλγόριθμος 4****: Αναζήτηση συμβολοσειράς (αλγόριθμος brute-force).* | 15 |
| ***Αλγόριθμος 5:*** *Αναφοράς γειτονικών σημείων σε σχέση με σταθερή απόσταση: Πρώτη προσέγγιση.* | 17 |
| ***Αλγόριθμος 6:*** *Αναφοράς γειτονικών σημείων σε σχέση με σταθερή απόσταση: Βελτιωμένος αλγόριθμος.* | 18 |

# ΠΡΟΓΡΆΜΜΑΤΑ

|  |  |
| --- | --- |
| ***Πρόγραμμα 1****: Υπολογισμός δισδιάστατων μεγίστων.* | 11 |
| ***Πρόγραμμα 2****: Πρόβλημα ελαχίστων αποστάσεων.* | 13 |
| ***Πρόγραμμα 3****: Αναζήτηση συμβολοσειράς (αλγόριθμος brute-force).* | 15 |

# 

# ΣΧΉΜΑΤΑ

|  |  |
| --- | --- |
| ***Σχήμα 2.1****: Ρυθμός αύξησης συναρτήσεων.* | 7 |
| ***Σχήμα 2.2****: Δισδιάστατα μέγιστα.* | 9 |
| ***Σχήμα 2.3****: Συνάρτηση πολυπλοκότητας αλγόριθμου δισδιάστατων μεγίστων.* | 11 |
| ***Σχήμα 2.4****: Σημεία πάνω σε ευθεία σε μονοδιάστατο χώρο.* | 17 |

# 2. ΠΟΛΥΠΛΌΚΟΤΗΤΑ.

Πριν την σχεδίαση ενός αλγορίθμου, είναι στοιχειωδώς απαραίτητο να ορισθεί το κριτήριο ή κριτήρια, τα οποία πρέπει να εκπληρώνει, ώστε να μπορεί να χαρακτηρισθεί εάν είναι «καλός» αλγόριθμος ή όχι. Το βασικό στοιχείο ενός καλού αλγόριθμου λοιπόν, είναι το κατά πόσον είναι ή όχι αποδοτικός. Και απόδοση σημαίνει η χρήση υπολογιστικών πόρων που απαιτούνται για την επίλυση του προβλήματος. Οι υπολογιστικοί πόροι (ανάλογα και με την φύση του αλγόριθμου), μπορεί να είναι η CPU, η μνήμη ή ακόμη και πιθανούς δικτυακούς πόρους (πχ bandwidth) που χρησιμοποιούνται. Ουσιαστικά όμως, στα περισσότερα προβλήματα, αυτό που εξετάζεται είναι η χρήση της CPU δηλαδή, πόσος χρόνος απαιτείται για την εκτέλεση του αλγόριθμου. Αυτό όμως προϋποθέτει να δοθεί και τυποποιηθεί ένα υπολογιστικό μοντέλο, ώστε όλοι να αναφέρονται σε αυτό. Αυτή η υπολογιστική μηχανή θεωρείται όσο πιο απλή όσο είναι δυνατόν. Δηλαδή, είναι σε θέση να εκτελεί μόνο στοιχειώδεις αριθμητικές πράξεις (πχ πρόσθεση, αφαίρεση, και τα λοιπά), και επίσης στοιχειώδεις συγκρίσεις (πχ είναι το a > 4;). Η υπολογιστική αυτή μηχανή ονομάζεται ***RAM*** από το ακρωνύμιο *Random Access Machine*. Η RAM, αποτελείται από ένα μόνο επεξεργαστή (επομένως δεν είναι δυνατή η παράλληλη επεξεργασία), και εκτελεί τις οδηγίες του αλγόριθμου σειριακά, δηλαδή μία την φορά. Τέλος, θεωρείται ότι η μνήμη της είναι μεγάλη ή τουλάχιστον τόσο μεγάλη, όσο απαιτεί ο οποιοσδήποτε αλγόριθμος.

## 2.1 Ορισμοί.

Η έννοια της ***Πολυπλοκότητα και Απόδοσης Αλγόριθμου,*** αντιπροσωπεύει το κόστος χρήσης του αλγόριθμου για την επίλυση ενός προβλήματος. Το πρόβλημα χαρακτηρίζεται από τα δεδομένα εισόδου, ενώ η συνάρτηση της πολυπλοκότητας *f(n)*  εκφράζει την απαίτηση του αλγόριθμου σε χρόνο εκτέλεσης, σε σχέση με το μέγεθος των δεδομένων εισόδου *n*. Επειδή είναι ιδιαίτερα δύσκολο ή και αδύνατο σε πάρα πολλές περιπτώσεις, να βρεθεί μία ακριβής συνάρτηση, συνήθως σε αυτήν την περίπτωση, ενδιαφέρει η εύρεση της τιμής της *f(n)* στις εξής περιπτώσεις:

* Καλύτερη Περίπτωση, δηλαδή η ελάχιστη τιμή της *f(n),*
* Χειρότερη Περίπτωση, δηλαδή η μέγιστη τιμή της *f(n),*
* Μέση Περίπτωση, δηλαδή η αναμενόμενη τιμή της *f(n),*

με πλέον χρήσιμη την χειρότερη περίπτωση.

Όμως το πρόβλημα της ανάλυσης ενός αλγόριθμου, παραμένει όταν τα δεδομένα εισόδου *n*, αυξάνονται πάρα πολύ. Για αυτό είναι απαραίτητη η χρήση και επίδειξη ασυμπτωτικών συναρτήσεων, οι οποίες να δείχνουν την τάξη μεγέθους της αύξησης του υπολογιστικού χρόνου, χωρίς να είναι απαραίτητο να δείχνουν την ακριβή συνάρτηση της πολυπλοκότητάς του. Έτσι, είναι ιδιαίτερα χρήσιμοι οι ορισμοί που ακολουθούν.

* + ***Ορισμός 1***. Θεωρούμε ότι *f(n)=o(g(n))* για (μικρό όμικρον – little oh) εάν . Δηλαδή, η συνάρτηση *f* αυξάνει πιο αργά από την συνάρτηση *g,* όταν το *n* είναι πολύ μεγάλο.



*Παραδείγματα*:

* + - ,



* + - .



* + ***Ορισμός 2***. Θεωρούμε ότι *f(n) = O(g(n))* για (μεγάλο όμικρον – big oh) εάν , τέτοια ώστε . Δηλαδή, η συνάρτηση *f,* δεν αυξάνει πιο γρήγορα από την συνάρτηση *g*.



*Παραδείγματα*:

* + - ,



* + - .



* + - * *Απόδειξη*: με επικρατούντα όρο τον . Επομένως .



* + - .



* + - * *Απόδειξη*: . Ο επικρατών όρος είναι ο δεύτερος γιατί και επομένως .



* + - Γενικά, οι σημαντικότερες συναρτήσεις τάξης μεγέθους παρουσιάζουν την ακόλουθη διάταξη (αύξουσα):



* + ***Ορισμός 3***. Θεωρούμε ότι , (θήτα) εάν υπάρχουν σταθερές τέτοιες ώστε . Δηλαδή οι δύο συναρτήσεις, *f* και *g*, αυξάνουν με τον ίδιο περίπου ρυθμό.



*Παραδείγματα*:



* + ***Ορισμός 4***. Θεωρούμε ότι , (ασυμπτωτικά) εάν . Δηλαδή όχι μόνον ότι οι δύο συναρτήσεις *f* και *g,* αυξάνουν με τον ίδιο περίπου ρυθμό, αλλά και ότι το πηλίκο τους προσεγγίζει την μονάδα όσο .



*Παραδείγματα*:

* + - ,



* + - .



* + ***Ορισμός 5***. Θεωρούμε ότι , (ωμέγα) εάν και μία ακολουθία τέτοια ώστε . Στην πραγματικότητα αυτό δηλώνει την άρνηση του ‘*ο*’. Δηλαδή, εάν . Για παράδειγμα, είναι γνωστό ότι ο πολλαπλασιασμός δύο πινάκων, δεν μπορεί με κανένα τρόπο να επιτευχθεί με λιγότερους από πολλαπλασιασμούς. Αυτό σημαίνει ότι υπάρχει κατώτερο όριο, και ότι η πολυπλοκότητα του γινομένου, εκπεφρασμένου με την συνάρτηση *Ω* είναι .



Μερικές χρήσιμες μαθηματικές σειρές:

*Αριθμητική σειρά*: = Θ(n2).



*Γεωμετρική σειρά*: = Θ(*xn*).

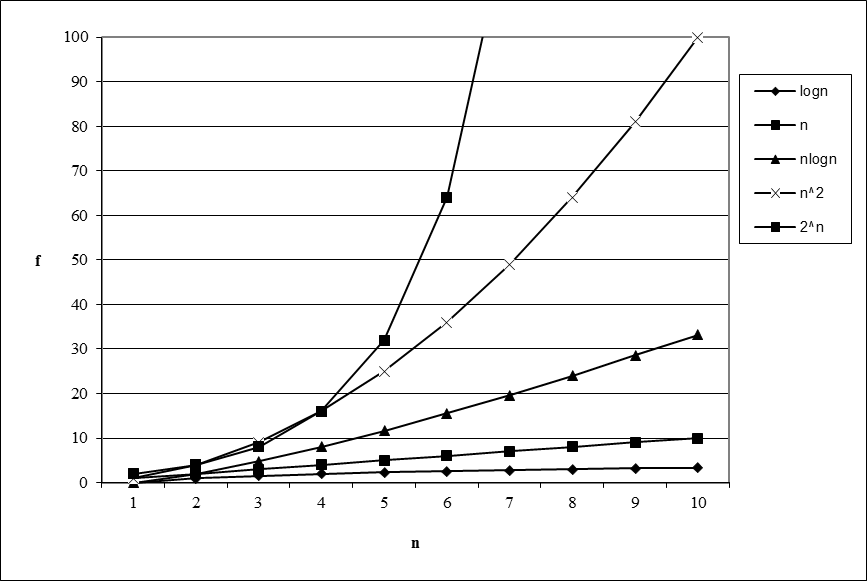


*Αρμονική σειρά*: = Θ(ln(*n*)).



Τέλος, στο επόμενο σχήμα (*Σχήμα 2.1*) φαίνεται ο ρυθμός αύξησης των συναρτήσεων , , ,, και .



[](#Σχήμα_2_1)

***Σχήμα 2.1*** *Ρυθμός αύξησης συναρτήσεων.*

## 2.2 Αλγόριθμοι Brute-Force.

Μία κατηγορία αλγορίθμων είναι οι λεγόμενοι brute-force αλγόριθμοι. Με βάση αυτή τη τεχνική, το πρόβλημα επιλύεται με τον πιο απλό και προφανή τρόπο, ο οποίος όμως δεν είναι πάντα (σχεδόν σε όλες τις περιπτώσεις) και ο πιο καλός. Επειδή ακριβώς επιλύει τα προβλήματα με τον πιο απλό και προφανή δυνατό τρόπο, είναι συνήθως εύκολο να υλοποιηθεί και να γίνει κατανοητός. Συνήθως αποτελούν και τον πιο «κουτό» τρόπο επίλυσης ενός προβλήματος. Ένα χαρακτηριστικό παράδειγμα τέτοιας τεχνικής, είναι η ταξινόμηση ενός πίνακα με την μέθοδο της φυσαλίδας, όπως επίσης και η σειριακή αναζήτηση. Σαν απόδειξη του πόσο αργοί είναι οι αλγόριθμοι αυτού του είδους, η μέθοδος της φυσαλίδας έχει πολυπλοκότητα της τάξης , ενώ άλλες τεχνικές ταξινόμησης (πχ quick sort), έχουν πολυπλοκότητα της τάξης .



Ένα χαρακτηριστικό παράδειγμα για την διευκρίνιση της τεχνικής Brute-Force είναι το πώς θα υπολόγιζε μία δύναμη, για παράδειγμα την δύναμη :



Brute-Force:



Ενώ με κάποια άλλη τεχνική (πολύ καλύτερη) θα μπορούσε να υπολογισθεί σαν:

Καλύτερη τεχνική: , δηλαδή 4 πολλαπλασιασμοί αντί για 16.



## 2.3 Δισδιάστατα μέγιστα.

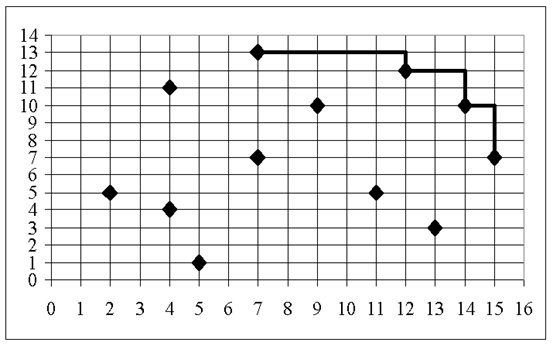
Ένα πολύ ενδιαφέρον και χρήσιμο πρόβλημα, είναι το πρόβλημα της εύρεσης μεγαλύτερων στοιχείων σε δισδιάστατο χώρο (2-dimension maxima). Ένα απλό παράδειγμα που το περιγράφει αρκετά καλά είναι το ακόλουθο: Εάν υποτεθεί ότι κάποιος θέλει να αγοράσει ένα αυτοκίνητο, και ότι αυτό που τον συγκινεί περισσότερο είναι η τελική ταχύτητα του αυτοκινήτου. Βέβαια επειδή δεν διαθέτει πολλά χρήματα, τον ενδιαφέρει επίσης να είναι όσο πιο φτηνό γίνεται. Φυσικά τα πιο γρήγορα αυτοκίνητα δεν είναι και φτηνά. Ο υποψήφιος αγοραστής δεν μπορεί να αποφασίσει αν τον ενδιαφέρει πιο πολύ η ταχύτητα ή η τιμή αλλά αυτό που ξέρει σίγουρα, είναι ότι δεν θα ήθελε να αγοράσει ένα αυτοκίνητο, εάν υπάρχει κάποιο άλλο το οποίο είναι και φτηνότερο και ταχύτερο.

Το πρόβλημα αυτό ανάγεται στην εύρεση μέγιστου ή και μέγιστων, σε ένα δισδιάστατο χώρο όπου οι συντεταγμένες αναπαριστούν (στην προκείμενη περίπτωση), η μία τιμή και η άλλη ταχύτητα. Στο προκείμενο πρόβλημα το αυτοκίνητο που ενδιαφέρει είναι αυτό για το οποίο ο συνδυασμός ταχύτητας ⁄ τιμής είναι βέλτιστος. Το πρόβλημα δεν είναι δυνατόν να λυθεί εάν δεν τυποποιηθεί πρώτα μαθηματικά. Αυτή η τυποποίηση θα βοηθήσει τόσο στην καλύτερη κατανόηση του προβλήματος, όσο και στην γενίκευση της λύσης του.

Η μαθηματική τυποποίηση του προβλήματος είναι η ακόλουθη:

* + Εάν p αναπαριστά ένα σημείο στον δισδιάστατο χώρο με συντεταγμένες p = (p.x, p.y) τότε εάν δεν υπάρχει άλλο σημείο q τέτοιο ώστε p.x<q.x ΚΑΊ p.y<q.y τότε το σημείο p θεωρείται μέγιστο (δεν καλύπτεται από κανένα άλλο).
  + Στο πρόβλημα της αγοράς αυτοκινήτου, εάν στον άξονα x απεικονισθούν οι ταχύτητες των διαθέσιμων αυτοκινήτων, και στον άξονα y οι αρνητικές τιμές των τιμών (ώστε όσο αυξάνει ο y να μειώνονται στην πραγματικότητα οι τιμές), τότε η εύρεση των μέγιστων αποτελούν τις καλύτερες περιπτώσεις αγοράς αυτοκινήτων.

*ΠΡΟΣΟΧΉ*: Είναι δυνατόν να υπάρχουν πολλά μέγιστα όπως φαίνεται στο παρακάτω *Σχήμα 2.2*.



***Σχήμα 2.2*** *Δισδιάστατα μέγιστα.*

Ένας σχετικά απλός αλλά καθόλου «έξυπνος» αλγόριθμος, είναι το να ελέγχει όλα τα σημεία, σε σχέση με όλα τα υπόλοιπα (brute-force algorithm). Δηλαδή για όλα τα σημεία να ελέγχει εάν όλα τα υπόλοιπα, έχουν και τις δύο συντεταγμένες τους μικρότερες από το ελεγχόμενο. Η αναλυτική περιγραφή του αλγόριθμου δίνεται στον *Αλγόριθμο 1*.

***Αλγόριθμος 1****: Δισδιάστατα μέγιστα – Αλγόριθμος Brute-Force.*

1: Δεδομένα ⁄ Είσοδος: Πίνακας Πn, n = πλήθος σημείων.

2: Διαδικασία Δισδιάστατα Μέγιστα1.

3: Αρχή,

4: Ακέραιοι: i, j,

5: Boolean: Μέγιστο.

6: Για i από 1 μέχρι n,

7: Έστω Μέγιστο 🡨 Αληθές,

8: Για j από 1 μέχρι n,

9: Εάν (i≠j) ΚΑΊ (Πi.x ≤ Πj.x) KAI (Πi.y ≤ Πj.y). Τότε,

10: Μέγιστο 🡨 Ψευδές,

11: Τέλος Εάν,

12: Τέλος Για (j).

13: Εάν (Μέγιστο) Τότε Επέστρεψε Πι,

14: Τέλος Για (i).

15: Τέλος Αλγόριθμου «Δισδιάστατα Μέγιστα1».

(*Παρατήρηση*: Αποτελεί στοιχειώδης διαδικασία η απόδειξη (και μαθηματική εάν χρειασθεί), της ***ορθότητας ενός αλγόριθμου***, δηλαδή για κάθε αλγόριθμο πρέπει να αποδεικνύεται μαθηματικά ότι είναι ορθός. Βέβαια στην προκείμενη περίπτωση είναι προφανές, μιας και εξετάζονται όλα τα στοιχεία σε σχέση με όλα τα υπόλοιπα.)

Ο παραπάνω αλγόριθμος φυσικά είναι σωστός, αλλά το πρόβλημα που τίθεται είναι ***πόσο γρήγορος είναι***. Δηλαδή ποια είναι η πολυπλοκότητά του. Είναι φανερό ότι για κάθε τιμή του εξωτερικού βρόγχου, ο εσωτερικός εκτελείται n φορές. Επίσης, εάν υποθέσουμε ότι μετράμε το πόσες φορές θα προσπελασθεί κάποιο σημείο του πίνακα Π, τότε ο εσωτερικός βρόγχος θα προσπελάσει 4 φορές κάποιο σημείο του πίνακα, ενώ ο εξωτερικός 2 (είναι το σημείο που θα επιστραφεί, και στην χειρότερη περίπτωση που όλα τα σημεία είναι μέγιστα). Επομένως, η πολυπλοκότητα του αλγόριθμου, εκφρασμένη ανάλογα με τα δεδομένα εισόδου είναι:

.



Επομένως, η πολυπλοκότητα της μεθόδου είναι της τάξης .



Βέβαια αυτό που πρώτιστα ενδιαφέρει, είναι η απόδοση του αλγόριθμου όταν το *n* γίνει μεγάλο (για μικρό n όλοι οι αλγόριθμοι είναι γρήγοροι). Επομένως με κάποιο τρόπο, πρέπει να υπολογισθεί η αύξηση της βραδύτητας του αλγόριθμου για μεγάλα *n*, δηλαδή ο ρυθμός αύξησης του χρόνου όσο το n αυξάνει. Εάν ο παραπάνω αλγόριθμος, βελτιωθεί όπως φαίνεται στην *Αλγόριθμο 2,* τότε αυτό που μόλις υπολογίσθηκε, αποτελεί την χειρότερη περίπτωση.

***Αλγόριθμος 2****: Δισδιάστατα μέγιστα – Δεύτερος αλγόριθμος.*

1: Δεδομένα ⁄ Είσοδος: Πίνακας Πn, n = πλήθος σημείων.

2: Διαδικασία Δισδιάστατα Μέγιστα2.

3: Αρχή,

4: Ακέραιοι: i, j,

5: Boolean: Μέγιστο.

6: Για i από 1 μέχρι n,

7: Έστω Μέγιστο 🡨 Αληθές,

8: Για j από 1 μέχρι n,

9: Εάν (i≠j) ΚΑΙ (Πi.x ≤ Πj.x) KAI (Πi.y ≤ Πj.y). Τότε,

10: Μέγιστο 🡨 Ψευδές,

11: **Τέλος εσωτερικού βρόγχου.**

12: Τέλος Εάν,

13: Τέλος Για (j).

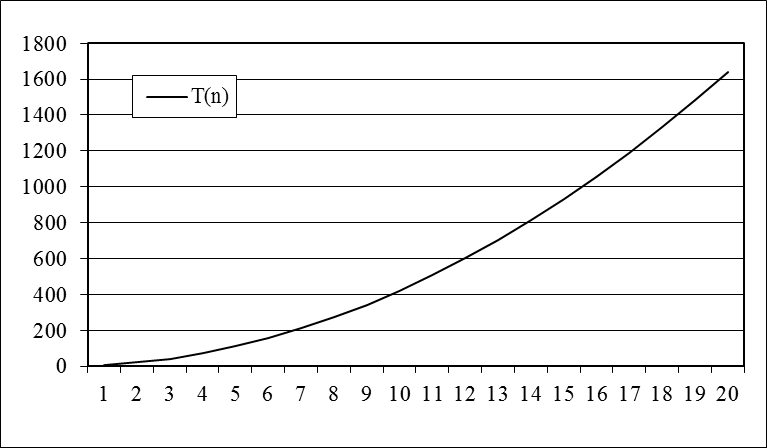
14: Εάν (Μέγιστο) Τότε Επέστρεψε Πι,

15: Τέλος Για (i).

16: Τέλος Αλγόριθμου «Δισδιάστατα Μέγιστα2».

Όπως έχει ήδη αποδειχθεί, ο ρυθμός αύξησης του αλγόριθμου είναι: . Δηλαδή, όπως φαίνεται στο παρακάτω *Σχήμα 2.3*:





***Σχήμα 2.3*** *Συνάρτηση πολυπλοκότητας αλγόριθμου δισδιάστατων μεγίστων.*

Στο *Πρόγραμμα 1,* δίνεται ο υπολογισμός δισδιάστατων μεγίστων (*Αλγόριθμος 2*), στην γλώσσα προγραμματισμού C.

***Πρόγραμμα 1****: Υπολογισμός δισδιάστατων μεγίστων.*

#include <stdio.h>.

#define N 12 /\* πλήθος σημείων \*/,

/\* Περιγραφή της δομής «Σημείο» \*/,

struct Simio {

int x;

int y;

};

struct Simio P[N]; /\* Πίνακας σημείων \*/,

main()

{

int i, j, k=0, l=0;

int megisto;

/\* Εισαγωγή σημείων \*/,

for (i=0; i<N; i++) {

printf("\ Εισαγωγή των συντεταγμένων του %3d στοιχείου:", i);

scanf("%d %d", &P[i].x, &P[i].y);

}

printf("\n\n ΥΠΟΛΟΓΙΣΜΌΣ ΜΕΓΊΣΤΩΝ \n\n");

for (i=0; i<N; i++) {

megisto = 0;

for (j=0; j<N; j++) {

l++; /\* μετρητής αριθμού επαναλήψεων \*/,

if ((i!=j) && (P[i].x<=P[j].x) && (P[i].y<=P[j].y)) {

megisto = 1; break;

}

}

if (megisto == 0) {

k++;

printf("\n %d-->%5d, %5d", k, P[i].x, P[i].y);

}

}

printf("\n\n Αριθμός επαναλήψεων: %5d \n\n", l);

}

## 2.4 Το πρόβλημα των ελαχίστων αποστάσεων (closest-pair problem).

Ένα ενδιαφέρον πρόβλημα είναι το εξής: Δοθέντων n σημείων στον δισδιάστατο χώρο, να βρεθούν τα δύο εκείνα σημεία, τα οποία η απόσταση μεταξύ τους είναι η πιο μικρή δυνατή (closest-pair problem). Μία brute-force τεχνική, είναι να υπολογισθούν όλες οι αποστάσεις, και στη συνέχεια να επιλεγεί η πιο μικρή. Τα σημεία που την δημιουργούν, είναι και τα πιο «κοντινά» σημεία, α) να αναπτυχθεί αυτός ο αλγόριθμος και στη συνέχεια, β) να υλοποιηθεί σε κάποια γλώσσα προγραμματισμού.

### Λύση.

***Αλγόριθμος 3****: Πρόβλημα ελαχίστων αποστάσεων (Brute-Force).*

1: Δεδομένα ⁄ Είσοδος: Πίνακας Πn, n = πλήθος σημείων.

2: Διαδικασία Υπολογισμού Ελαχίστων Αποστάσεων.

3: Ακέραιοι: i, j, k, d1, d2, k=0,

4: Ακέραιος M = n\*(n-1),

5: Πίνακας πραγματικών AM,

6: Πραγματικός: el.

7: Για i από 1 μέχρι n,

8: Για j από 1 μέχρι n,

9: *Ak*🡨,



10: Εάν ( k = 0). Τότε *(αρχικοποίηση τιμών),*

11: el 🡨 Ak,

12: d1 🡨 0,

13: d2 🡨 1.

14: Αλλιώς,

15: Εάν (el > Αk),

16: el 🡨 Αk,

17: d1 🡨 I,

18: d2 🡨 j,

19: Τέλος Εάν,

20: Τέλος Εάν.

21: k 🡨 k+1.

22: Τέλος Για (j),

23: Τέλος Για (i).

24: Εκτύπωσε «Τα σημεία που απέχουν ελάχιστα είναι τα: d1 και d2»,

25: Εκτύπωσε «και η απόστασή τους είναι: el».

26: Τέλος Αλγόριθμου «Υπολογισμού Ελαχίστων Αποστάσεων».

***Πρόγραμμα 2****: Πρόβλημα ελαχίστων αποστάσεων.*

#include <stdio.h>

#include <math.h>

#define N 3.

struct Simio {

float x;

float y;

};

struct Simio P[N] ;

float apostasi(struct Simio, struct Simio);

main()

{

int i, j, k=0, M = N\*(N-1), d1, d2;

float el; /\* Ελάχιστη τιμή \*/,

float a[M];

/\* Εισαγωγή σημείων \*/,

for (i=0; i<N; i++) {

printf("\n Εισαγωγή των συντεταγμένων του %3d σημείου:", i+1);

scanf("%f %f", &P[i].x, &P[i].y);

}

for (i=0; i<N; i++)

for (j=i+1; j<N; j++) {

a[k] = apostasi(P[i], P[j]);

if (k == 0) { /\* αρχικοποίηση τιμών \*/,

el = a[k];

d1 = 0;

d2 = 1;

}

else

if (el > a[k]) {

el = a[k];

d1 = i;

d2 = j;

}

k++;

}

printf("\n\n Τα σημεία που απέχουν ελάχιστα είναι τα: %3d και %3d", d1+1, d2+1);

printf("\n\n και η απόστασή τους είναι: %.2f \n\n", el);

}

/\* ΣΥΝΆΡΤΗΣΗ υπολογισμού απόστασης \*/,

float apostasi(struct Simio A, struct Simio B)

{

float d;

d = (A.x \* A.x -B.x \* B.x) + (A.y \* A.y -B.y \* B.y);

if (d < 0)

d = -d;

return sqrt(d);

}

## 2.5 Εύρεση αλφαριθμητικών.

Ένα ιδιαίτερα σημαντικό πρόβλημα, είναι η αναζήτηση ενός αλφαριθμητικού μέσα σε ένα κείμενο (ότι ακριβώς η λειτουργία grep στα λειτουργικά συστήματα τύπου UNIX, ή η λειτουργία «Εύρεση» των περισσοτέρων επεξεργαστών κειμένου). Εδώ υπάρχουν δύο ενδεχόμενα: 1) να αναζητά ακριβώς το ίδιο αλφαριθμητικό, και 2) να αναζητά όσο το δυνατόν πλησιέστερα σχήματα, με το υπό αναζήτηση αλφαριθμητικό (δηλαδή να βρει κάτι παραπλήσιο). Στην δεύτερη περίπτωση ανάγεται και η αναζήτηση γενετικών κωδικών. Η μοριακή αλυσίδα του DNA, μπορεί να διασπασθεί σε μεγάλες σειρές, οι οποίες αποτελούνται από τέσσερις βασικούς τύπους, τους C, G, T, και A. Το να βρεθεί στη βιολογία ακριβές ταίριασμα, είναι μάλλον απίθανο, και αυτό που είναι το ζητούμενο, είναι να βρεθεί ένας βαθμός ομοιότητας. Ένας τρόπος μέτρησης της ομοιότητας, είναι το μέγεθος της σειράς που είναι ακριβώς ίδιες.

Ένα απλός brute-force αλγόριθμος, δίνεται στον *Αλγόριθμο 4*  (για απόλυτο ταίριασμα συμβολοσειρών), και η υλοποίησή του στο *Πρόγραμμα 3*. Η απλοϊκή λύση του προβλήματος, στηρίζεται στο ψάξιμο όλων των συμβολοσειρών ένα προς ένα χαρακτήρων τους, μέχρι να βρεθεί ή όχι η προς αναζήτηση συμβολοσειρά. Έστω ότι είναι η συμβολοσειρά που αναζητάτε, για το εάν εμφανίζεται στην συμβολοσειρά , . Ο brute-force αλγόριθμος, ψάχνει εάν υπάρχει μία ή και περισσότερες *Μ*-άδες χαρακτήρων, ακριβώς ίδιες στην *S*. Για να το επιτύχει, ξεκινάει από τον πρώτο χαρακτήρα, και συγκρίνει τον χαρακτήρα του *T* με τον αντίστοιχο του *S*. Εάν είναι ίδιοι, προχωράει με τους επόμενους *M* -1 του *Τ*. Εάν όλοι είναι ίδιοι, τότε το πρώτο «ταίριασμα» βρέθηκε, και συνεχίζει με τον ίδιο τρόπο μέχρι το τέλος του *S,* το οποίο βέβαια είναι ο χαρακτήρας *N* -*M*. Εάν κάποιος από τους ενδιάμεσους χαρακτήρες δεν είναι ίδιος με τον αντίστοιχο του *S*, τότε πάλι αρχίζει από την αρχή την αναζήτηση με τον πρώτο χαρακτήρα του *T,* και με τον χαρακτήρα του *S* που σταμάτησε το προηγούμενο ταίριασμα.



***Αλγόριθμος 4****: Αναζήτηση συμβολοσειράς (αλγόριθμος brute-force).*

1: Είσοδος ⁄ Δεδομένα: Συμβολοσειρά αναζήτησης Τ μεγέθους Μ, και συμβολοσειρά που θα αναζητηθεί S μεγέθους Ν.

2: Ακέραιοι: i🡨0, j 🡨0, k🡨0,

3: Εφόσον (i < N -M+1), /\* H συμβολοσειρά που αναζητάτε, να μην ξεπερνάει σε μήκος την συμβολοσειρά στην οποία γίνεται η αναζήτηση \*/,

4: Εφόσον (Τj = Si),

5: i 🡨 i +1, /\* προχώρησε στο επόμενο γράμμα \*/,

6: i 🡨 j +1, /\* προχώρησε στο επόμενο γράμμα \*/,

7: Τέλος Εφόσον.

8: Εάν (j = M). Τότε, /\* βρέθηκε \*/,

8: k 🡨 k+1, /\* αυξάνεται ο αριθμός των φορών που βρέθηκε \*/,

10: Τέλος Εάν.

11: Εάν (j > 0 ΚΑΊ j < Μ), /\* Εάν δεν βρέθηκε το i, πρέπει να πάρει την τιμή της τελευταίας αναζήτησης \*/,

12: i 🡨 i –1,

13: Τέλος Εάν.

14: j 🡨 0, /\* Αρχίζει η αναζήτηση από την αρχή \*/,

15: i 🡨 i +1,

16: Τέλος Εφόσον.

17: Τέλος Αλγόριθμου «Αναζήτηση συμβολοσειράς (αλγόριθμος brute-force)».

***Πρόγραμμα 3****: Αναζήτηση συμβολοσειράς (αλγόριθμος brute-force).*

#include <stdio.h>

#define N 26

#define M 3

main()

{

char T[M] = "EFG";

char S[N] = "ABCDEFGHIJKLMNOPEFEFGTVUWX";

int i=0, j=0, k=0;

/\* String Matching \*/,

while (i<N -M+1) {

while (T[j] == S[i]) {

i++;

j++;

}

if (j == M) {

k++;

printf("\n Βρέθηκε για %d φορά στην θέση %d", k, i -M);

}

if (j>0 && j<M)

i--;

j = 0;

i++;

}

if (k == 0)

printf("\n Δεν βρέθηκε \n");

}

Βέβαια, το ζητούμενο είναι να βρεθεί ένας πιο γρήγορος και πιο έξυπνος τρόπος αντιμετώπισης αυτού του προβλήματος, επειδή η πολυπλοκότητά του είναι της τάξης .



## 2.6 Υπολογιστική γεωμετρία.

Η Υπολογιστική Γεωμετρία είναι ένας κλάδος της Πληροφορικής, ο οποίος ασχολείται με προβλήματα τα οποία μπορούν να περιγραφούν, αναλυθούν, και επιλυθούν, κάνοντας χρήση της Γεωμετρίας. Μερικά από αυτά τα προβλήματα, είναι «το κυρτό περίβλημα», «η τομή δύο ευθειών», και τα λοιπά. Ένα σημαντικό τέτοιο πρόβλημα, είναι το ονομαζόμενο: *πρόβλημα της αναφοράς των γειτονικών σημείων (σε σχέση με σταθερή προκαθορισμένη απόσταση)*. Δηλαδή, δεδομένου ενός συνόλου σημείων, έστω P = {p1,p2,…,pN}, να αναφερθούν εκείνα τα ζευγάρια σημείων τα οποία απέχουν μεταξύ τους, το πολύ ένα προκαθορισμένο μήκος R, ή να βρεθούν εκείνα τα (p, q), για τα οποία η Ευκλείδεια απόστασή τους, είναι μικρότερη ή ίση με R, .



***Πρώτη προσέγγιση:***

Το πρόβλημα μπορεί να λυθεί εύκολα με μία brute force προσέγγιση. Εάν εξετασθούν όλα τα πιθανά ζευγάρια σημείων, τότε απλά αναφέρονται ποια από αυτά απέχουν μεταξύ τους απόσταση ≤ R. To σύνολο των ζευγαριών που πρέπει να εξετασθούν είναι:

.



Οπότε η πολυπλοκότητα του αλγόριθμου είναι της τάξης.



Ο αλγόριθμος ο οποίος προκύπτει από την πρώτη αυτή προσέγγιση, συνοπτικά είναι ο ακόλουθος:

***Αλγόριθμος 5****:**Αναφοράς γειτονικών σημείων σε σχέση με σταθερή απόσταση: Πρώτη προσέγγιση.*

1. Για i🡨1 μέχρι Ν,

2. Για j🡨i+1 μέχρι Ν,

3. Εάν τότε,



4. Τα σημεία είναι γειτονικά,



5. Τέλος\_Εάν,

6. Τέλος\_Για (j),

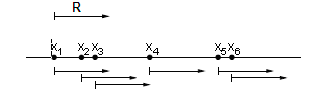
7. Τέλος\_Για (i),

8. Τέλος Αλγόριθμου.

***Δεύτερη προσέγγιση:***

Για να γίνει καλύτερα αντιληπτό το πρόβλημα, και χωρίς να βλάπτεται η γενικότητά του, ας υποτεθεί ότι τα σημεία ανήκουν στον μονοδιάστατο χώρο, δηλαδή όλα τα σημεία ανήκουν σε μία ευθεία, P = {x1, x2,.., xN}. Επίσης, ας υποτεθεί ότι το σύνολο των σημείων P, είναι ταξινομημένο σε αύξουσα διάταξη, δηλαδή, xi < xj για i < j. Τώρα, είναι αρκετά απλή η ακόλουθη σκέψη: Για κάθε i από 1 μέχρι Ν, αρκεί να εξετασθούν εκείνα τα σημεία xj για τα οποία: j 🡨 i+1 μέχρι k, και για τα οποία να ισχύει: , και να τερματίζει στο πρώτο σημείο (κινούμενος πάντα προς τα δεξιά), για το οποίο: (*Σχήμα 2.4*).





***Σχήμα 2.4****: Σημεία πάνω σε ευθεία σε μονοδιάστατο χώρο.*

Η πολυπλοκότητα του αλγόριθμου είναι της τάξης, (λόγω της ταξινόμησης). Γενικά, εάν πρέπει να ταξινομηθούν τα σημεία, δεν μπορεί να αποφευχθεί πολυπλοκότητα μικρότερη της τάξης, . Είναι όμως η ταξινόμηση αναγκαία;



***Απόδειξη:***

Εάν υποτεθεί ότι το σημείο έχει γειτονικά σημεία, τότε και επιπλέον, το κάθε σημείο θα κάνει μία επιπλέον σύγκριση, ώστε να αντιληφθεί ότι έχει ξεπεράσει την απόσταση *R*. Οπότε, ο απαιτούμενος χρόνος εκτέλεσης του αλγόριθμου, είναι (συνυπολογιζόμενης και της ταξινόμησης – N log N):



.



Και κατά συνέπεια, η πολυπλοκότητα του αλγόριθμου είναι .



Η περιγραφή του αλγόριθμου είναι η ακόλουθη (υποτίθεται ότι ο πίνακας των σημείων έχει ήδη ταξινομηθεί):

***Αλγόριθμος 6****:**Αναφοράς γειτονικών σημείων σε σχέση με σταθερή απόσταση: Βελτιωμένος αλγόριθμος.*

1. Για i🡨1 μέχρι Ν,

2. j🡨i+1,

3. Εφόσον ,



4. Τα σημεία είναι γειτονικά,



5. j🡨j+1,

6. Τέλος\_Εφόσον,

7. Τέλος\_Για (i).

8. Τέλος Αλγόριθμου.

***Τρίτη προσέγγιση:***

Το ζητούμενο τώρα είναι, εάν είναι δυνατόν να βρεθεί ένας αλγόριθμος που λειτουργεί σε γραμμικό χρόνο, δηλαδή να είναι της τάξης . Το πρόβλημα είναι ότι πρέπει να αποφευχθεί η ταξινόμηση. Εάν χωρισθούν τα σημεία σε διαστήματα μεγέθους *R*, δηλαδή,



Τότε το κάθε σημείο ανήκει σε ένα από αυτά τα διαστήματα. Επίσης, πρέπει να παρατηρηθεί, ότι το κάθε διάστημα χαρακτηρίζεται από ένα δείκτη . Δηλαδή, κάθε σημείο x ανήκει στο διάστημα με δείκτη,



.



Επομένως, σε χρόνο , το κάθε σημείο τοποθετείται σε ένα τέτοιο διάστημα. Προφανώς, το μέγιστο πλήθος των διαστημάτων είναι *Ν* (εάν υποτεθεί ότι μόνο ένα σημείο ανήκει σε κάθε διάστημα). Είναι προφανές, ότι όλα τα σημεία που ανήκουν στο ίδιο διάστημα είναι γειτονικά, καθότι το μήκος του διαστήματος δεν υπερβαίνει το *R*. Επομένως, τα μόνα σημεία που πρέπει να ελεγχθούν, είναι όλα τα σημεία των διαστημάτων, με εκείνα τα σημεία που βρίσκονται στο αμέσως επόμενο δεξιό διάστημα (εάν υπάρχει τέτοιο).



Τέλος δεύτερης ενότητας.

[](http://creativecommons.org/licenses/by-nc-nd/3.0/deed.el)[](http://www.edulll.gr/)