Αλγόριθμοι και Πολυπλοκότητα

Ανάλυση Αλγορίθμων

Δημήτρης Μιχαήλ

Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Ανάλυση Αλγορίθμων

Η ανάλυση αλγορίθμων περιλαμβάνει τη διερεύνηση του τρόπου με τον οποίο κλιμακώνονται οι απαιτήσεις τους για πόρους, δηλαδή ο χώρος και ο χρόνος που χρησιμοποιούν, με την αύξηση του μεγέθους της εισόδου.

Αποδοτικότητα

Χρειαζόμαστε έναν απτό ορισμό της αποδοτικότητας που να είναι ανεξάρτητος από

- πλατφόρμα υλοποίησης και
- στιγμιότυπο εισόδου,

και ο οποίος θα κάνει πρόβλεψη για τα αυξανόμενα μεγέθη της εισόδου.

Χρόνοι Εκτέλεσης Χειρότερης Περίπτωσης

Χρόνος χειρότερης περίπτωσης. Σκοπός είναι να βρούμε ένα άνω όριο για τον χρόνο ενός αλγορίθμου σε οποιαδήποτε είσοδο μεγέθους n.

- Αποτυπώνει αρκετά καλά την αποδοτικότητα στην πράξη.
- Δρακόντειο μέτρο, αλλά δύσκολο να βρούμε καλύτερη εναλλακτική.

Ανάλυση μέσης περίπτωσης. Στην ανάλυση μέσης τιμής (average-case analysis) μελετάμε την απόδοση ενός αλγορίθμου ως προς τον μέσο όρο κάποιων ”τυχαίων” στιγμιοτύπων.

- Δύσκολο (έως αδύνατο) να μοντελοποιήσουμε πραγματικά προβλήματα με κάποια τυχαία κατανομή.
- Ένας γρήγορος αλγόριθμος για μια κατανομή εισόδου μπορεί να είναι πολύ αργός σε άλλη κατανομή.

Χώροι Αναζήτησης και Κλιμάκωση

Πως ποσοτικοποιούμε την έννοια του ”εύλογου” χρόνου εκτέλεσης;

Χώροι αναζήτησης. Ο χώρος αναζήτησης (search space), δηλαδή ο χώρος των λύσεων ενός προβλήματος, τείνει να αυξάνεται εκθετικά σε συνάρτηση με το μέγεθος της εισόδου n. Εάν το μέγεθος της εισόδου αυξηθεί κατά ένα, ο αριθμός των δυνατοτήτων αυξάνεται πολλαπλάσια.

Ωμή βία. Για πολλά προβλήματα, υπάρχει ένας φυσικός αλγόριθμος **ωμής βίας** (brute force) που ελέγχει όλες τις δυνατές λύσεις.

- Συνήθως χρειάζεται χρόνο 2^n ή περισσότερο για εισόδους μεγέθους n.
- Απαγορευτικό στην πράξη.

Χώροι Αναζήτησης και Κλιμάκωση

Πως ποσοτικοποιούμε την έννοια του ”εύλογου” χρόνου εκτέλεσης;

Χώροι αναζήτησης. Ο χώρος αναζήτησης (search space), δηλαδή ο χώρος των λύσεων ενός προβλήματος, τείνει να αυξάνεται εκθετικά σε συνάρτηση με το μέγεθος της εισόδου n. Εάν το μέγεθος της εισόδου αυξηθεί κατά ένα, ο αριθμός των δυνατοτήτων αυξάνεται πολλαπλάσια.

Καλύτερη ιδιότητα κλιμάκωσης. Θα θέλαμε ένας καλός αλγόριθμος να έχει καλύτερη ιδιότητα κλιμάκωσης: όταν το μέγεθος εισόδου αυξάνεται κατά μια σταθερή ποσότητα – για παράδειγμα, κατά 2 – ο αλγόριθμος θα πρέπει να επιβραδύνεται μόνο κατά ένα σταθερό συντελεστή C.

Πολυωνυμικός Χρόνος

Ιδιότητα "Καλού" Αλγορίθμου. Υπάρχουν απόλυτες σταθερές $c > 0$ και $d > 0$ έτσι ώστε για κάθε στιγμιότυπο εισόδου μεγέθους n, ο χρόνος εκτέλεσης φράσσεται από $c \cdot n^d$ στοιχειώδη υπολογιστικά βήματα[1].

Λέμε ότι ο αλγόριθμος έχει **πολυωνυμικό χρόνο εκτέλεσης** (polynomial running time) ή ότι είναι ένας **αλγόριθμος πολυωνυμικού χρόνου** (polynomial-time algorithm).

[1] Επίτηδες αόριστο, μπορείτε όμως να φανταστείτε μια εντολή γλώσσας assembly ή μια εντολή γλώσσας C.

Πολυωνυμικός Χρόνος

Κλιμάκωση. Τα όρια πολυωνυμικού χρόνου έχουν την ιδιότητα κλιμάκωσης που αναζητούμε.

Αν το μέγεθος εισόδου αυξηθεί από n σε $2n$, το όριο του χρόνου εκτέλεσης αυξάνεται από $c \cdot n^d$ σε $c \cdot (2n)^d = c \cdot 2^d \cdot n^d$ που είναι μια καθυστέρηση κατά ένα συντελεστή 2^d.

Το 2^d είναι μια σταθερά αφού το d είναι σταθερά.

Αποδοτικός Αλγόριθμος

Ορισμός. Ένας αλγόριθμος είναι **αποδοτικός** αν έχει πολυωνυμικό χρόνο εκτέλεσης.

Γιατί; Γιατί πραγματικά δουλεύει στην πράξη!

- Υπάρχει σημαντική διαφορά στην πράξη μεταξύ πολυωνυμικού χρόνου και εκθετικού.
- Συνήθως στην πράξη οι πολυωνυμικοί αλγόριθμοι έχουν μικρές σταθερές.
- Παρόλο που $1.33 \cdot 10^{23} \cdot n^{100}$ είναι πολυώνυμο, στην πράξη δεν έχει νόημα.
- Μερικοί εκθετικοί (ή χειρότεροι) αλγόριθμοι χρησιμοποιούνται στην πράξη επειδή τα στιγμιότυπα χειρότερης-περίπτωσης φαίνεται πως είναι πολύ σπάνια.

Χρόνοι

	n	$n \log_2 n$	n^2	n^3
n = 10	< 1 sec	< 1 sec	< 1 sec	< 1 sec
n = 30	< 1 sec	< 1 sec	< 1 sec	< 1 sec
n = 50	< 1 sec	< 1 sec	< 1 sec	< 1 sec
n = 100	< 1 sec	< 1 sec	< 1 sec	1 sec
n = 1,000	< 1 sec	< 1 sec	1 sec	18 λεπτά
n = 10,000	< 1 sec	< 1 sec	2 λεπτά	12 ημέρες
n = 100,000	< 1 sec	2 sec	3 ώρες	32 έτη
n = 1,000,000	1 sec	20 sec	12 ημέρες	31,710 έτη

	1.5^n	2^n	$n!$
n = 10	< 1 sec	< 1 sec	4 sec
n = 30	< 1 sec	< 18 λεπτά	10^{25} έτη
n = 50	< 11 λεπτά	< 36 έτη	∞
n = 100	12,892 έτη	10^{17} έτη	∞
n = 1,000	∞	∞	∞
n = 10,000	∞	∞	∞
n = 100,000	∞	∞	∞
n = 1,000,000	∞	∞	∞

Χρόνοι εκτέλεσης (στρογγυλεμένοι προς τα πάνω) διαφόρων αλγορίθμων, σε επεξεργαστή που εκτελεί ένα εκατομμύριο εντολές υψηλού επιπέδου το δευτερόλεπτο. Όταν ο χρόνος εκτέλεσης υπερβαίνει τα 10^{25} έτη γράφουμε ∞.

Ασυμπτωτικός Ρυθμός Αύξησης

Εκφράζουμε λοιπόν την έννοια ότι ο χρόνος εκτέλεσης ενός αλγορίθμου στη χειρότερη περίπτωση για εισόδους μεγέθους n αυξάνεται με ένα ρυθμό που είναι το πολύ ανάλογος κάποιας συνάρτησης $f(n)$.

Συνήθως περιγράφουμε έναν αλγόριθμο σε κάποια μορφή ψευδοκώδικα, μια ενδιάμεση μορφή μεταξύ γλώσσας προγραμματισμού και αγγλικών.

Όταν δίνουμε ένα όριο για το χρόνο εκτέλεσης ενός αλγορίθμου, μετράμε γενικά τον αριθμό των βημάτων ψευδοκώδικα που εκτελούνται.

Ένα βήμα ενός αλγορίθμου μπορεί να είναι

- εκχώρηση τιμής σε μεταβλητή
- αναζήτηση καταχώρησης σε πίνακα,
- έμμεση αναφορά (dereference) ενός δείκτη
- εκτέλεση μιας αριθμητικής λειτουργίας σε έναν ακέραιο σταθερού μεγέθους
- κ.τ.λ

Δεν μας ενδιαφέρει όμως με τόση λεπτομέρεια, μιας και ο ακριβής χρόνος εκτέλεσης ενός τέτοιου βήματος είναι συνάρτηση της πλατφόρμας εκτέλεσης.

Ασυμπτωτικός Ρυθμός Αύξησης

Θα μπορούσαμε να μετράμε τον χρόνο εκτέλεσης ενός αλγορίθμου με λεπτομέρεια, π.χ

$$f(n) = 1.62n^2 + 3.5n + 8$$

Προβλήματα

- δύσκολο να είμαστε τόσο ακριβής
- θέλουμε να ταξινομήσουμε αλγορίθμους σε ευρείες κατηγορίες, και άρα μικρότερο επίπεδο αναλυτικότητας
- η πραγματική απόδοση εξαρτάται από την πλατφόρμα εκτέλεσης, π.χ
  - ένα μηχάνημα μπορεί να χρειάζεται 25 εντολές μηχανής χαμηλού επιπέδου για να υλοποιήσει μια εντολή υψηλού επιπέδου ενώ ένα άλλο 50.
  - διαφορετικά μηχανήματα έχουν διαφορετικές συχνότητες και άρα πέρνει διαφορετικό χρόνο να εκτελεστεί κάθε βήμα

Ασυμπτωτικός Ρυθμός Αύξησης

Θέλουμε λοιπόν να εκφράσουμε το ρυθμό αύξησης του χρόνου εκτέλεσης και των άλλων λειτουργιών με έναν τρόπο που να μην επηρεάζεται από

1. σταθερούς παράγοντες
2. όρους χαμηλής τάξης

π.χ για την συνάρτηση

$$f(n) = 1.62n^2 + 3.5n + 8$$

θα θέλαμε να μπορούμε να πούμε πως αυξάνεται όπως η συνάρτηση n^2.

Έστω $T(n)$ μια συνάρτηση, π.χ ο χρόνος εκτέλεσης χειρότερης περίπτωσης ενός συγκεκριμένου αλγορίθμου για μια είσοδο μεγέθους n.

Δεδομένου μιας άλλης συνάρτησης $f(n)$ λέμε[2] ότι η $T(n)$ είναι $\mathcal{O}(f(n))$ αν, για αρκετά μεγάλο n, η συνάρτηση $T(n)$ φράσσεται εκ των άνω από ένα σταθερό πολλαπλάσιο της $f(n)$.

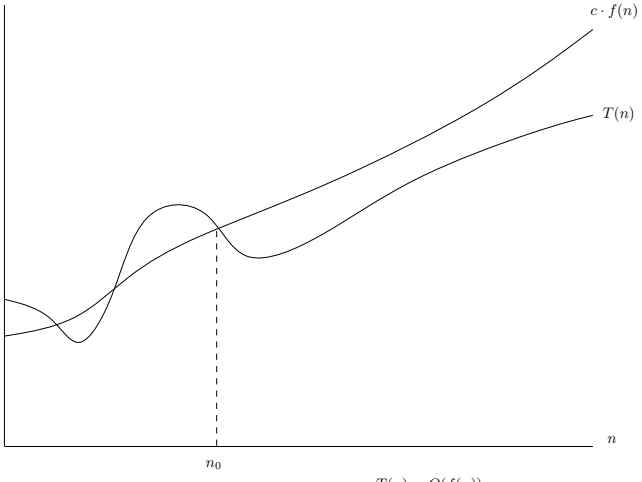
Πολλές φορές γράφουμε $T(n) = \mathcal{O}(f(n))$.

[2] Διαβάζεται η $T(n)$ είναι τάξης $f(n)$.

Ασυμπτωτικά Άνω Όρια

big-Oh notation

Ορισμός. Η $T(n)$ είναι $\mathcal{O}(f(n))$ αν υπάρχουν σταθερές $c > 0$ και $n_0 \geq 0$ έτσι ώστε, για όλα τα $n \geq n_0$, να έχουμε $0 \leq T(n) \leq c \cdot f(n)$.



Προσοχή η σταθερά c πρέπει να λειτουργεί για όλα τα n, δεν μπορεί δηλαδή το c να εξαρτάται από

Έστω ένας αλγόριθμος του οποίου ο χρόνος εκτέλεσης έχει την μορφή

$$T(n) = pn^2 + qn + r$$

για θετικές σταθερές p, q και r.

Παρατηρώντας πως για $n \geq 1$ ισχύει $qn \leq qn^2$ και $r \leq rn^2$, μπορούμε να γράψουμε

$$T(n) = pn^2 + qn + r \leq pn^2 + qn^2 + rn^2 = (p + q + r)n^2$$

για όλα τα $n \geq 1$.

Ακριβώς ότι απαιτεί ο ορισμός του $\mathcal{O}(\cdot)$ όπου $c = p + q + r$ και άρα λέμε πως η συνάρτηση $T(n) = \mathcal{O}(n^2)$.

Ο συμβολισμός $\mathcal{O}(\cdot)$ εκφράζει μόνο ένα άνω όριο, και όχι τον ακριβή ρυθμό αύξησης της συνάρτησης.

Όπως ισχυριστήκαμε πως $T(n) = pn^2 + qn + r = \mathcal{O}(n^2)$ μπορούμε να δείξουμε επίσης πως $T(n) = \mathcal{O}(n^3)$.

Έχει συμβεί πολλές φορές να έχει αποδειχθεί ότι ένας αλγόριθμος έχει χρόνο εκτέλεσης $\mathcal{O}(n^3)$, και μετά από μερικά χρόνια με καλύτερη ανάλυση να αποδειχθεί πως ο ίδιος αλγόριθμος είναι στην πραγματικότητα $\mathcal{O}(n^2)$. Δεν υπήρχε πρόβλημα με το πρώτο αποτέλεσμα, ήταν ένα σωστό άνω όριο. Απλά δεν ήταν ο ”αυστηρότερος” δυνατός χρόνος εκτέλεσης.

Συμβολισμός μεγάλου όμικρον
big-Oh notation

Θεώρημα
$2n^3 + 100n^2 + n \in \mathcal{O}(n^3)$

Συμβολισμός μεγάλου όμικρον
big-Oh notation

Θεώρημα
$2n^3 + 100n^2 + n \in \mathcal{O}(n^3)$

Απόδειξη
Αρκεί να βρούμε θετική σταθερά c και $n_0 \geq 0$ ώστε να ισχύει:

$$2n^3 + 100n^2 + n \leq cn^3$$

□

Θεώρημα
$2n^3 + 100n^2 + n \in \mathcal{O}(n^3)$

Απόδειξη
Αρκεί να βρούμε θετική σταθερά c και $n_0 \geq 0$ ώστε να ισχύει:

$$2n^3 + 100n^2 + n \leq cn^3$$

Διαιρώντας με $n > 1$ έχουμε πως αρκεί

$$2n^2 + 100n + 1 \leq cn^2.$$

□

Θεώρημα
$2n^3 + 100n^2 + n \in \mathcal{O}(n^3)$

Απόδειξη
Αρκεί να βρούμε θετική σταθερά c και $n_0 \geq 0$ ώστε να ισχύει:

$$2n^3 + 100n^2 + n \leq cn^3$$

Διαιρώντας με $n > 1$ έχουμε πως αρκεί

$$2n^2 + 100n + 1 \leq cn^2.$$

Για $n > 1$ έχουμε όμως πως $2n^2 + 100n + 1 \leq 2n^2 + 100n^2 + n^2 \leq 103n^2$. Άρα μπορούμε να θέσουμε $c = 103$ και $n_0 = 1 > 0$. □

Συμβολισμός μεγάλου όμικρον
big-Oh notation

Θεώρημα
$n \in \mathcal{O}(2^n)$

Θεώρημα

$n \in \mathcal{O}(2^n)$

Απόδειξη

Πρέπει να βρούμε $c > 0, n_0 > 0$ ώστε να ισχύει

$$n \leq c \cdot 2^n.$$

□

Θεώρημα

$n \in \mathcal{O}(2^n)$

Απόδειξη

Πρέπει να βρούμε $c > 0, n_0 > 0$ ώστε να ισχύει

$$n \leq c \cdot 2^n.$$

Παρατηρώντας τον τρόπο που μεγαλώνουν οι συναρτήσεις βλέπουμε πως για $c = 1$ και $n_0 = 1$ η ανισότητα ισχύει. Πρέπει όμως να αποδείξουμε πως παραμένει αλήθεια για κάθε $n \geq n_0 = 1$.

□

Θεώρημα

$n \in \mathcal{O}(2^n)$

Απόδειξη

Πρέπει να βρούμε $c > 0, n_0 > 0$ ώστε να ισχύει

$$n \leq c \cdot 2^n.$$

Παρατηρώντας τον τρόπο που μεγαλώνουν οι συναρτήσεις βλέπουμε πως για $c = 1$ και $n_0 = 1$ η ανισότητα ισχύει. Πρέπει όμως να αποδείξουμε πως παραμένει αλήθεια για κάθε $n \geq n_0 = 1$.

Έστω $f(n) = 2^n - n$. Η συνάρτηση είναι συνεχής και παραγωγίσιμη. Η πρώτη παράγωγος είναι $f'(n) = ln(2) \cdot 2^n - 1$ και είναι θετική για κάθε $n \geq 1$. Άρα η συνάρτηση $f(n)$ είναι γνησίως αύξουσα για $n \geq 1$ που σημαίνει πως παραμένει θετική για κάθε $n \geq n_0 = 1$. □

Συμβολισμός μεγάλου όμικρον
big-Oh notation

Θεώρημα
$1000000 \in \mathcal{O}(1)$

Θεώρημα

$1000000 \in \mathcal{O}(1)$

Απόδειξη

Πρέπει να δείξουμε πως υπάρχουν $c > 0, n_0 > 0$ ώστε

$$1000000 \leq c \cdot 1$$

για κάθε $n \geq n_0$.

□

Θεώρημα

$1000000 \in \mathcal{O}(1)$

Απόδειξη

Πρέπει να δείξουμε πως υπάρχουν $c > 0, n_0 > 0$ ώστε

$$1000000 \leq c \cdot 1$$

για κάθε $n \geq n_0$.

Η σχέση ισχύει για $c = 1000001$ και $n_0 = 1$. □

Θεώρημα

Για $a > 0$ ισχύει πως $f(n) = an^2 + bn \in \mathcal{O}(n^2)$.

Θεώρημα

Για $a > 0$ ισχύει πως $f(n) = an^2 + bn \in \mathcal{O}(n^2)$.

Απόδειξη

Πρέπει να βρούμε θετική σταθερά c και σταθερά $n_0 \geq 0$ ώστε

$$an^2 + bn \leq cn^2 \text{ για κάθε } n \geq n_0.$$

□

Θεώρημα

Για $a > 0$ ισχύει πως $f(n) = an^2 + bn \in \mathcal{O}(n^2)$.

Απόδειξη

Πρέπει να βρούμε θετική σταθερά c και σταθερά $n_0 \geq 0$ ώστε

$$an^2 + bn \leq cn^2 \text{ για κάθε } n \geq n_0.$$

Επειδή

$$an^2 + bn \leq |a|n^2 + |b|n \leq |a|n^2 + |b|n^2 = (|a| + |b|)n^2$$

για $n \geq 1$ μπορούμε να διαλέξουμε $c = |a| + |b|$ και n_0 οποιαδήποτε θετική τιμή μεγαλύτερη ή ίση με ένα π.χ $n_0 = 1$.

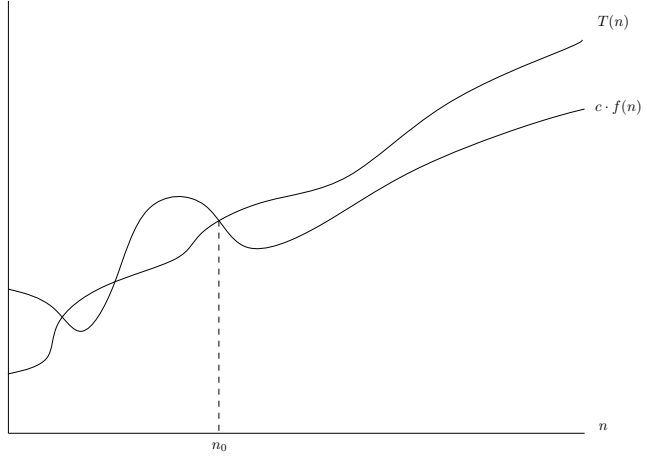
□

Ασυμπτωτικά Κάτω Όρια

Ω notation

Συμπληρωματική σημειογραφία για τα κάτω όρια.

Ορισμός. Η $T(n)$ είναι $\Omega(f(n))$ αν υπάρχουν σταθερές $c > 0$ και $n_0 \geq 0$ έτσι ώστε, για όλα τα $n \geq n_0$, να έχουμε $0 \leq c \cdot f(n) \leq T(n)$.



Έστω πάλι η συνάρτηση $T(n) = pn^2 + qn + r$ όπου p, q και r θετικές σταθερές.

Μπορούμε να πούμε πως για $n \geq 1$ ισχύει $T(n) = pn^2 + qn + r \geq pn^2$ και άρα σύμφωνα με τον ορισμό της $\Omega(\cdot)$ για $c = p$ έχουμε πως $T(n) = \Omega(n^2)$.

Μπορούμε επίσης να πούμε πως $T(n) = \Omega(n)$ αφού για $n \geq 1$ ισχύει πως $T(n) \geq pn^2 \geq pn$.

Αυστηρά Ασυμπτωτικά Όρια

Αν μπορούμε να δείξουμε πως

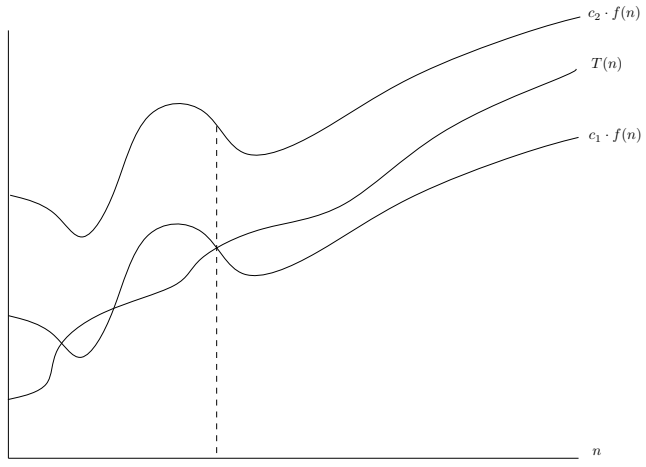
- $T(n) = \mathcal{O}(f(n))$ και
- $T(n) = \Omega(f(n))$

τότε έχουμε βρει το σωστό όριο.

Ορισμός. Εάν μια συνάρτηση $T(n)$ είναι και $\mathcal{O}(f(n))$ και $\Omega(f(n))$, λέμε ότι η $T(n)$ είναι $\Theta(f(n))$.

Αυστηρά Ασυμπτωτικά Όρια

Ορισμός. Εάν μια συνάρτηση $T(n)$ είναι και $\mathcal{O}(f(n))$ και $\Omega(f(n))$, λέμε ότι η $T(n)$ είναι $\Theta(f(n))$.



$T(n) = \Theta(f(n))$

Ιδιότητες

Μεταβατικότητα (transitivity)

- Αν $f = \mathcal{O}(g)$ και $g = \mathcal{O}(h)$ τότε $f = \mathcal{O}(h)$.
- Αν $f = \Omega(g)$ και $g = \Omega(h)$ τότε $f = \Omega(h)$.
- Αν $f = \Theta(g)$ και $g = \Theta(h)$ τότε $f = \Theta(h)$.

Αθροίσματα συναρτήσεων

- Αν $f = \mathcal{O}(h)$ και $g = \mathcal{O}(h)$ τότε $f + g = \mathcal{O}(h)$.
- Αν $f = \Omega(h)$ και $g = \Omega(h)$ τότε $f + g = \Omega(h)$.
- Αν $f = \Theta(h)$ και $g = \Theta(h)$ τότε $f + g = \Theta(h)$.

Πολυώνυμα. $a_0 + a_1 n + \ldots + a_d n^d$ είναι $\Theta(n^d)$ εάν $a_d > 0$.

Ορισμός. Ένας αλγόριθμος είναι πολυωνυμικού χρόνου εαν ο χρόνος εκτέλεσης του $T(n)$ είναι $\mathcal{O}(n^d)$ για κάποια σταθερά d, όπου η d είναι ανεξάρτητη από το μέγεθος της εισόδου.

Λογάριθμοι.

- Θυμηθείτε πως $\log_b n$ είναι ένας αριθμός x τέτοιος ώστε $b^x = n$.
- Οι λογάριθμοι είναι συναρτήσεις που αυξάνονται πολύ αργά.

Λήμμα

Για κάθε $b > 1$ και κάθε $x > 0$, έχουμε $\log_b n = \mathcal{O}(n^x)$.

- Πολλές φορές δεν γράφουμε καθόλου την βάση του λογαρίθμου μιας και ισχύει η σχέση

$$\log_a n = \frac{1}{\log_b a} \cdot \log_b n$$

Εκθετικές. Οι εκθετικές συναρτήσεις είναι της μορφής $f(n) = r^n$ για κάποια σταθερή βάση r. Για $r > 1$ η συνάρτηση έχει πολύ γρήγορη αύξηση.

Λήμμα

Για κάθε $r > 1$ και κάθε $d > 0$, έχουμε $n^d \in \mathcal{O}(r^n)$.

Επισκόπηση Συνηθισμένων Χρόνων Εκτέλεσης

Οι πιο συνηθισμένοι χρόνοι εκτέλεσης είναι οι εξής.

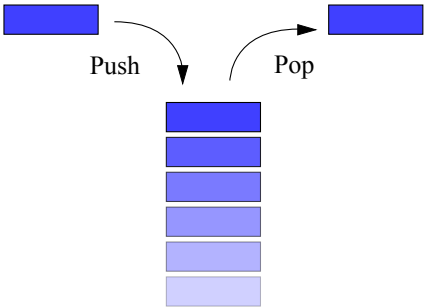
- σταθερός – $\mathcal{O}(1)$
- γραμμικός – $\mathcal{O}(n)$
- $\mathcal{O}(n \log n)$
- τετραγωνικός – $\mathcal{O}(n^2)$
- κυβικός – $\mathcal{O}(n^3)$
- πολυωνυμικός – $\mathcal{O}(n^d)$ για $d > 0$
- εκθετικός – $\mathcal{O}(r^n)$ για $r > 1$

Σταθερός Χρόνος $\mathcal{O}(1)$

Constant Time

Σταθερός χρόνος είναι ο χρόνος που είναι ανεξάρτητος από το μέγεθος της εισόδου *n*.

Λειτουργίες Στοίβας. Προσθήκη ενός στοιχείου σε μια στοίβα (PUSH).



Σε μία σωστά υλοποιημένη στοίβα, το μέγεθος δεν παίζει ρόλο. Ο χρόνος είναι ίδιος εαν η στοίβα έχει 100 στοιχεία ή 10^{100} στοιχεία.

Γραμμικός Χρόνος
Linear Time

Γραμμικός χρόνος. Ο χρόνος εκτέλεσης είναι ανάλογος με το μέγεθος εισόδου.

Υπολογισμός μέγιστου. Υπολογισμός του μέγιστου των n αριθμών $a_1, \ldots, a_n$.

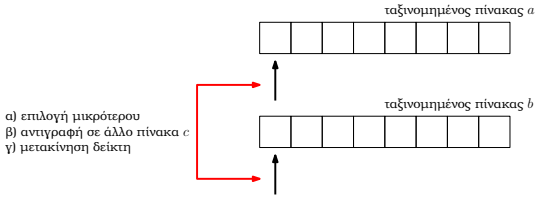
Algorithm 1: Εύρεση Μέγιστου

$max = a_1$
for $i = 2$ *έως* n **do**
  if $a_i > max$ **then**
    θέσε $max = a_i$
  end
end

Γραμμικός Χρόνος

Linear Time

Συγχώνευση. Συνδυασμός δύο ταξινομημένων λιστών $a_1, a_2, \ldots, a_n$ και $b_1, b_2, \ldots, b_n$ σε μια ταξινομημένη λίστα.



Algorithm 2: Merge

i = 1, j = 1
while *καμία άδεια λίστα* **do**
  if $a_i \leq b_j$ **then**
    πρόσθεσε το a_i στην έξοδο και αύξησε το i
  else
    πρόσθεσε το b_j στην έξοδο και αύξησε το j
  end
end
πρόσθεσε όλα τα στοιχεία της μη άδειας λίστας στην έξοδο

$\mathcal{O}(n \log n)$ Χρόνος

Προκύπτει συχνά στους αλγορίθμους ”διαίρει και βασίλευε” που θα εξετάσουμε αργότερα.

Ταξινόμηση. Οι αλγόριθμοι mergesort (ταξινόμηση συγχώνευσης) και heapsort (ταξινόμηση σωρού) κάνουν $\mathcal{O}(n \log n)$ συγκρίσεις.

Τετραγωνικός Χρόνος: $\mathcal{O}(n^2)$

Quadratic Time

Τετραγωνικός Χρόνος. Απαρίθμηση όλων των δυνατών ζευγών ενός συνόλου στοιχείων.

Κοντινότερο ζεύγος σημείων. Με είσοδο n σημείων στο επίπεδο $(x_1, y_1), \ldots, (x_n, y_n)$, θέλουμε να βρούμε το ζεύγος σημείων που έχουν την μικρότερη απόσταση.

Algorithm 3: Κοντινότερο ζεύγος σημείων

$min = (x_1 - x_2)^2 + (y_1 - y_2)^2$
for $i = 1$ *έως* n **do**
  for $j = 1 + 1$ *έως* n **do**
    $d = (x_i - x_j)^2 + (y_i - y_j)^2$
    if $d < min$ **then**
      min=d
    end
  end
end

Γίνεται και πιο γρήγορα!

Κυβικός Χρόνος: $\mathcal{O}(n^3)$

Qubic Time

Πολλαπλασιασμός Πινάκων. Έστω 2 πίνακες A και B μεγέθους $n \times n$.

$$\begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{pmatrix}$$

```
for( i = 0; i < n; i++ )
    for( j = 0; j < n; j++ )
        for( k = 0, c[i][j] = 0.0; k < n; k++ )
            c[i][j] += a[i][k] * b[k][j];
```

Γίνεται και πιο γρήγορα!

Εκθετικός Χρόνος
Exponential Time

Δύο συνηθισμένα όρια είναι τα $\mathcal{O}(2^n)$ και $\mathcal{O}(n!)$.

Ανεξάρτητο Σύνολο. Δεδομένου ενός γραφήματος $G(V, E)$ βρείτε το μεγαλύτερο ανεξάρτητο σύνολο.

Algorithm 4: Μέγιστο Ανεξάρτητο Σύνολο: Ωμή Βία

for *κάθε υποσύνολο κόμβων S* **do**
    Έλεγξε αν το *S* είναι ανεξάρτητο σύνολο
    if *S είναι μεγαλύτερο ανεξάρτητο σύνολο από το μέχρι τώρα* **then**
        Κατάγραψε το μέγεθος του *S* ως τρέχον μέγιστο
    end
end

Ένα σύνολο μεγέθους n έχει 2^n υποσύνολα. Ο έλεγχος κάθε υποσυνόλου πέρνει χρόνο $\mathcal{O}(n^2)$ και άρα ο αλγόριθμος χρειάζεται $\mathcal{O}(n^2 \cdot 2^n)$.