Απόδοση Αλγορίθμων

Γιάννης Θεοδωρίδης, Νίκος Πελέκης, Άγγελος Πικράκης
Τμήμα Πληροφορικής

Αλγόριθμοι και Προγράμματα

- Αλγόριθμος: μια μέθοδος ή μια διαδικασία που ακολουθείται για την επίλυση ενός προβλήματος
  - Μια συνταγή
- Ένας αλγόριθμος παίρνει την είσοδο ενός προβλήματος (συνάρτηση) και τη μετασχηματίζει στην έξοδο
  - Μια αντιστοίχηση της εισόδου στην έξοδο
- Ένα πρόβλημα μπορεί να επιλύεται με πολλούς εναλλακτικούς αλγορίθμους

Ιδιότητες Αλγορίθμου

- Ένας αλγόριθμος έχει τις ακόλουθες ιδιότητες:
  - Πρέπει να είναι <u>ορθός</u>
  - Πρέπει να αποτελείται από μια σειρά από <u>σαφή βήματα</u>
  - Πρέπει να αποτελείται από ένα <u>πεπερασμένο</u> αριθμό βημάτων
  - Πρέπει να είναι <u>σαφές</u> ποιο βήμα θα εκτελεστεί κάθε φορά
  - Πρέπει να <u>τερματίζει</u>
- Ένα πρόγραμμα υπολογιστή είναι ένα στιγμιότυπο ή μια διακριτή αναπαράσταση για έναν αλγόριθμο σε μια προγραμματιστική γλώσσα.
  - <u>Μαθηματικό υπόβαθρο</u>: Σύνολα, Αναδρομή, Επαγωγικές αποδείξεις, Λογάριθμοι, Αθροίσματα, Επαναληπτικές σχέσεις

Απόδοση Αλγορίθμων

- Συνήθως υπάρχουν πολλοί τρόποι (αλγόριθμοι) για την επίλυση ενός προβλήματος. Πώς επιλέγουμε μεταξύ αυτών;
- Πρέπει να ικανοποιηθούν δύο (αντικρουόμενοι) στόχοι
  1. Ο αλγόριθμος να είναι εύκολα κατανοητός, ενώ παράλληλα να είναι απλός τόσο ο προγραμματισμός του όσο και η αποσφαλμάτωση (debugging)
     - Ο στόχος (1) αφορά τις Αρχές Προγραμματισμού
  2. Ο αλγόριθμος να κάνει αποδοτική χρήση των πόρων του υπολογιστή
     - Ο στόχος (2) αφορά τις Δομές Δεδομένων και τη Θεωρία Αλγορίθμων
  - Όταν εστιάσουμε στο 2ο στόχο πώς μετράμε το κόστος;

Μέτρηση Απόδοσης

1. Εμπειρική σύγκριση (τρέξιμο προγραμμάτων)
2. Ασυμπτωτική ανάλυση αλγορίθμων

Παράγοντες που επηρεάζουν το χρόνο εκτέλεσης ενός προγράμματος:

- Για τους περισσότερους αλγορίθμους, ο χρόνος τρεξίματος εξαρτάται από το 'μέγεθος' της εισόδου
- Ο χρόνος τρεξίματος εκφράζεται ως **f(n)** για κάποια συνάρτηση **f** πάνω στο μέγεθος **n** της εισόδου

Παραδείγματα ρυθμού αύξησης

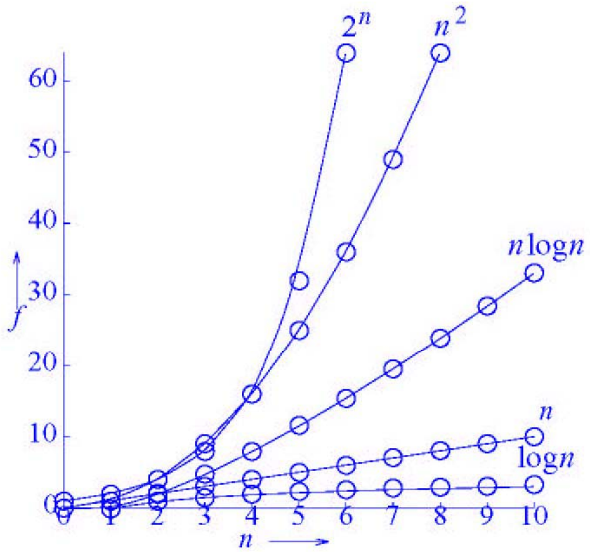
Παράδειγμα 1 (εύρεση του μέγιστου στοιχείου ενός πίνακα)

```
int largest(int array[], int n) {
int currlarge = 0;
     for (int i=1; i<n; i++)       // Για κάθε val
     if (array[currlarge] < array[i])
       currlarge = i;       // αποθήκευση της θέσης
     return currlarge;             // επιστροφή μεγαλύτερου
}
```

Παράδειγμα 2 ('χαζός' υπολογισμός της έκφρασης n(n-1))

```
sum = 0;
for (i=1; i<=n; i++)
  for (j=1; j<n; j++)
    sum++;
}
```

Γραφική αναπαράσταση ρυθμού αύξησης



Σχ. 2.24 (σελ. 97)

Παραδείγματα χρόνων εκτέλεσης

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

- Χρόνοι εκτέλεσης αλγορίθμων σε επεξεργαστή που εκτελεί 1.000.000 εντολές υψηλού επιπέδου /δευτερόλεπτο
- >10^{25} χρόνια = very long time

Καλύτερη, Χειρότερη, Μέση Περίπτωση

- Διαφορετικές είσοδοι του ίδιου μεγέθους συνήθως απαιτούν διαφορετικό χρόνο τρεξίματος
- Παράδειγμα: Σειριακή αναζήτηση ενός στοιχείου (του *K*) σε έναν πίνακα *n* ακεραίων:
  - Ξεκινάμε από το πρώτο στοιχείο του πίνακα και εξετάζουμε στη σειρά κάθε στοιχείο, μέχρι να βρούμε το *K*.
- Ποια είναι η καλύτερη / χειρότερη / μέση περίπτωση;
- Αν και ο μέσος χρόνος δείχνει να είναι το πιο δίκαιο μέτρο, μπορεί να είναι δύσκολο να υπολογιστεί

Ταχύτερος υπολογιστής ή Ταχύτερος αλγόριθμος;

- Τι συμβαίνει όταν αγοράζουμε έναν υπολογιστή 10 φορές ταχύτερο;

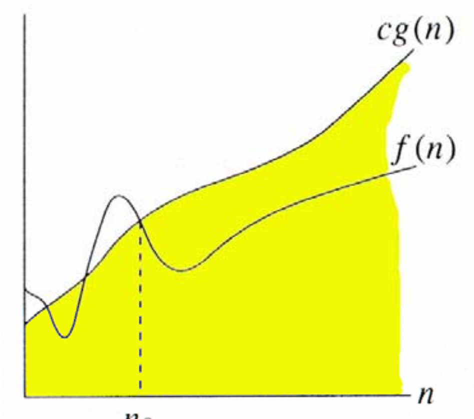
  n (*n'*): το μέγεθος εισόδου που μπορούμε να επεξεργαστούμε σε 1 ώρα με τον αργό (ταχύ) υπολογιστή, χρόνος που αντιστοιχεί σε έστω 10,000 (100,000) πράξεις

f(*n*)	*n*	*n'*	Αλλαγή	*n'*/*n*
$10n$	1,000	10,000	$n' = 10n$	10.00
$20n$	500	5,000	$n' = 10n$	10.00
$5n \log n$	250	1,842	$\sqrt{10}\, n < n' < 10n$	7.37
$2n^2$	70	223	$n' = \sqrt{10}n$	3.16
2^n	13	16	$n' = n + 3$	1.23

Ασυμπτωτική ανάλυση: Συμβολισμός Όμικρον κεφαλαίο (Ο)

- <u>Ορισμός</u>: $f(n) = O(g(n))$ αν και μόνο αν υπάρχουν θετικές σταθερές c και n_0 τέτοιες ώστε $f(n) \leq cg(n)$ για όλα τα $n \geq n_0$
- <u>Παράδειγμα</u>: Ο αλγόριθμος ... είναι $O(n^2)$ [καλύτερης, μέσης, χειρότερης] περίπτωσης
- <u>Ερμηνεία</u>: Για όλα τα αρκετά μεγάλα σύνολα δεδομένων (δηλαδή $n \geq n_0$), ο αλγόριθμος πάντα τερματίζει σε λιγότερα από $cg(n)$ βήματα στην [καλύτερη, μέση, χειρότερη] περίπτωση
- Αποτελεί <u>άνω όριο</u> κόστους

Ασυμπτωτική ανάλυση: Συμβολισμός Όμικρον κεφαλαίο (Ο)



$f(n) = O(g(n))$

Συμβολισμός Όμικρον κεφαλαίο (Ο) (συν.)

Ο συμβολισμός Ο δηλώνει ένα άνω όριο.

Παράδειγμα: Εάν $f(n) = 3n^2$ τότε η $f(n)$ είναι $O(n^2)$.

Επιθυμούμε όσο το δυνατό στενότερα όρια:

Παρόλο που ισχύει ότι η συνάρτηση $f(n) = 3n^2$ είναι $O(n^3)$, προτιμούμε $O(n^2)$

Παραδείγματα χρήσης του Ο

- Παράδειγμα 1: Εύρεση της τιμής *X* σε έναν πίνακα (μέσο κόστος) $f(n) = c_s n/2$.
  Για όλα τα n, $c_s n/2 \leq c_s n$.
  Επομένως, $f(n)$ είναι $O(n)$ για $n_0=1$ και $c=c_s$
- Παράδειγμα 2: $f(n) = c_1 n^2 + c_2 n$ στη μέση περίπτωση.
  $c_1 n^2 + c_2 n \leq c_1 n^2 + c_2 n^2 \leq (c_1 + c_2) n^2$ για όλα τα n
  Άρα $f(n) \leq cn^2$ για $c = c_1 + c_2$ και $n_0 = 1$
  Επομένως, $f(n)$ είναι $O(n^2)$ εξ ορισμού
- Παράδειγμα 3: $f(n) = c$. Λέμε ότι είναι $O(1)$

Μια συνηθισμένη παρεξήγηση

- "Η καλύτερη περίπτωση για τον αλγόριθμό μου είναι n=1 επειδή αυτό είναι το γρηγορότερο."

  ΛΑΘΟΣ!

- Το O αναφέρεται σε ένα <u>αυξανόμενο ρυθμό (rate)</u> καθώς το n τείνει στο ∞.

- Η καλύτερη περίπτωση ορίζεται ως: <u>ποια</u> είσοδος μεγέθους n είναι φθηνότερη από όλες τις εισόδους μεγέθους n.

Συμβολισμός Ωμέγα κεφαλαίο (Ω)

- <u>Ορισμός</u>: $f(n) = \Omega(g(n))$ αν και μόνο αν υπάρχουν θετικές σταθερές c και n_0 τέτοιες ώστε $\boldsymbol{f}(n) \geq cg(n)$ για όλα τα $n \geq n_0$
- <u>Παράδειγμα</u>: Ο αλγόριθμος ... είναι $\Omega(n^2)$ [καλύτερης, μέσης, χειρότερης] περίπτωσης
- <u>Ερμηνεία</u>: Για όλα τα αρκετά μεγάλα σύνολα δεδομένων (δηλαδή $n \geq n_0$), ο αλγόριθμος πάντα τερματίζει σε περισσότερα από $cg(n)$ βήματα στην [καλύτερη, μέση, χειρότερη] περίπτωση
- Αποτελεί <u>κάτω όριο</u> κόστους

Παράδειγμα χρήσης του Ω

$f(n) = c_1n^2 + c_2n$

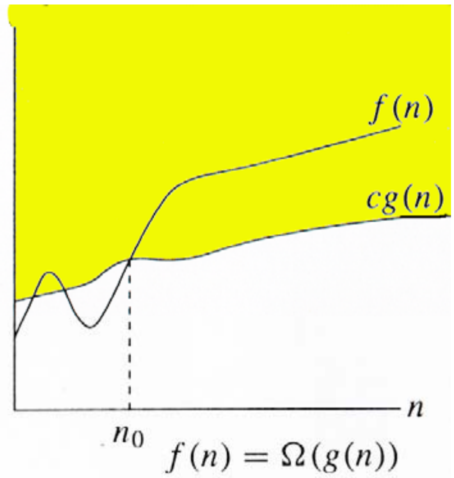
$c_1n^2 + c_2n \geq c_1n^2$ για όλα τα n

Άρα $f(n) \geq cn^2$ για $c = c_1$ και $n_0 = 1$.

Επομένως, η $f(n)$ είναι $\Omega(n^2)$ εξ ορισμού.

Θέλουμε το μεγαλύτερο κάτω όριο.

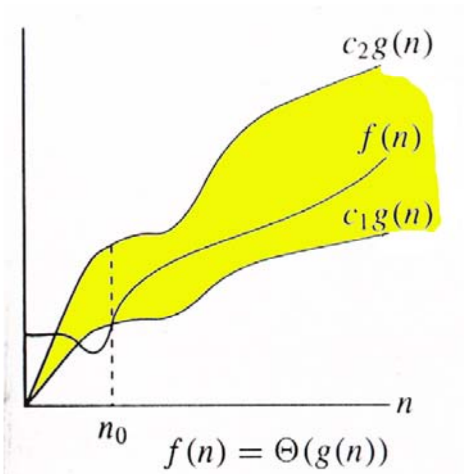
Παράδειγμα χρήσης του Ω



Συμβολισμός Θήτα κεφαλαίο (Θ)

- Όταν τα Ο και Ω ταυτιστούν, το δηλώνουμε αυτό με χρήση του συμβολισμού Θ.
- <u>Ορισμός</u>: $f(n) = \Theta(g(n))$ αν και μόνο αν υπάρχουν θετικές σταθερές c_1, c_2 και n_0 τέτοιες ώστε $c_1 g(n) \leq f(n) \leq c_2 g(n)$ για όλα τα $n \geq n_0$
- <u>Παρατήρηση</u>: Ένας αλγόριθμος λέμε ότι είναι Θ(g(n)) εάν είναι ταυτόχρονα Ο(g(n)) και Ω(g(n))

Συμβολισμός Θήτα κεφαλαίο (Θ)



(ακόμη) μια συνηθισμένη παρεξήγηση

Μπερδεύουμε τη χειρότερη περίπτωση με το άνω όριο

- Το άνω όριο αναφέρεται σε ένα αυξανόμενο ρυθμό
- Η χειρότερη περίπτωση αναφέρεται στη χειρότερη είσοδο ανάμεσα στις πιθανές εισόδους ενός δοθέντος μεγέθους

Κανόνες απλοποίησης

1. Εάν $f(n) = O(g(n))$ και $g(n) = O(h(n))$,
   τότε $f(n) = O(h(n))$
2. Εάν $f(n) = O(kg(n))$ για σταθερά $k > 0$,
   τότε $f(n) = O(g(n))$
3. Εάν $f_1(n) = O(g_1(n))$ και $f_2(n) = O(g_2(n))$,
   τότε $(f_1 + f_2)(n) = O(\max(g_1(n), g_2(n)))$
4. Εάν $f_1(n) = O(g_1(n))$ και $f_2(n) = O(g_2(n))$,
   τότε $f_1(n)f_2(n) = O(g_1(n)g_2(n))$

Ασυμπτωτικές ταυτότητες

	$f(n)$	Asymptotic
E1	c	$\oplus(1)$
E2	$\sum_{i=0}^{k} c_i n^i$	$\oplus(n^k)$
E3	$\sum_{i=1}^{n} i$	$\oplus(n^2)$
E4	$\sum_{i=1}^{n} i^2$	$\oplus(n^3)$
E5	$\sum_{i=1}^{n} i^k, k>0$	$\oplus(n^{k+1})$
E6	$\sum_{i=0}^{n} r^i, r>1$	$\oplus(r^n)$
E7	$n!$	$\oplus(n\ (n/e)^n)$
E8	$\sum_{i=1}^{n} 1/i$	$\oplus(\log n)$

$\oplus$ can be any one of O, Ω, and Θ

Σχ. 2.15 (σελ. 90)

Παραδείγματα

- Παράδειγμα 1: **a = b;**
  Αυτή η ανάθεση παίρνει σταθερό χρόνο *c*, οπότε είναι Θ(1)

Παράδειγμα 2: υπολογισμός n(n+1)/2

```
sum = 0;
for (i=1; i<=n; i++)
  sum += i;
```

Παράδειγμα 3: (εναλλακτικός) υπολογισμός n(n+1)/2

```
sum = 0;
for (i=1; i<=n; i++)
  for (j=1; j<=i; j++)
    sum++;
```

Σύγκριση 2 και 3: Υπολογισμός ίδιας έκφρασης με διαφορετικό κόστος! Θ(n) έναντι $\Theta(n^2)$

Παραδείγματα (συν.)

Παράδειγμα 4: υπολογισμός n^2

```
sum = 0;
for (i=1; i<=n; i++)
   for (j=1; j<=n; j++)
      sum++;
```

Σύγκριση 3 και 4:
Υπολογισμός διαφορετικών εκφράσεων με ίδιο κόστος!
$\Theta(n^2)$

Παράδειγμα 5:

```
sum = 0;
for (k=1; k<=n; k*=2)
   for (j=1; j<=n; j++)
      sum++;
```

Κόστος $\Theta(n \log n)$

Παράδειγμα 6:

```
sum = 0;
for (k=1; k<=n; k*=2)
   for (j=1; j<=k; j++)
      sum++;
```

Κόστος $\Theta(n)$

Παράδειγμα: Δυαδική αναζήτηση

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Key	11	13	21	26	29	36	40	41	45	51	54	56	65	72	77	83

- Πόσα στοιχεία εξετάζουμε στη χειρότερη περίπτωση?
  - Απάντηση: 4 (= $\log_2 16$)

Δυαδική αναζήτηση (συν.)

```
// Επιστροφή της θέσης του στοιχείου με τιμή
// x στον ταξινομημένο πίνακα a μεγέθους n.

int BinarySearch(int a[], int n, int x) {
  int left = 0; int right = n-1;      // όρια πίνακα
  while (left <= right) {           // σταμάτα όταν τα
                                    // όρια ταυτιστούν
    int middle = (left + right) / 2;
                    // έλεγχος του μεσαίου στοιχείου
    if (x == a[middle]) return middle;    // βρέθηκε
    if (x > a[middle]) left = middle + 1;
    else right = middle - 1;
   // αναζήτηση στο δεξί ή αριστερό μισό, αντίστοιχα
  }
  return -1;              // ένδειξη ότι δεν βρέθηκε
}
```


Χωρικά όρια κόστους – Ισορροπία μεταξύ χρονικού / χωρικού κόστους

- Τα χωρικά όρια μπορούν επίσης να αναλυθούν με χρήση ασυμπτωτικής ανάλυσης
- <u>Χρόνος</u>: Αλγόριθμος που επιλύει το πρόβλημα
- <u>Χώρος</u> της δομής δεδομένων που απαιτείται για την υλοποίηση του αλγορίθμου
- Μπορούμε να μειώσουμε το χρόνο εάν θελήσουμε να θυσιάσουμε το χώρο (και αντίστροφα)

  Σκεφτείτε (στην πορεία του μαθήματος) παραδείγματα ...