ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

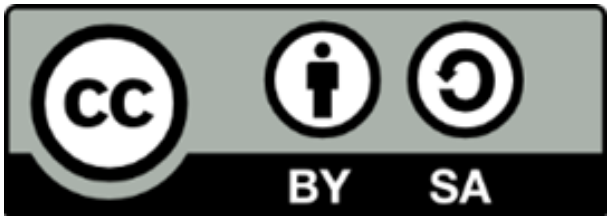
Ενότητα 6β: Ταξινόμηση με εισαγωγή και επιλογή

Μαρία Σατρατζέμη
Τμήμα Εφαρμοσμένης Πληροφορικής

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Μακεδονίας» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- 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.

1. **Ταξινόμηση με Εισαγωγή**
2. **Ταξινόμηση με Επιλογή**

Ταξινόμηση

- Η **ταξινόμηση** (*sorting*) τοποθετεί ένα σύνολο κόμβων ή εγγραφών σε μία συγκεκριμένη διάταξη (αύξουσα ή φθίνουσα) με βάση την τιμή του (πρωτεύοντος) κλειδιού της εγγραφής.
- Σκοπός της ταξινόμησης είναι στη συνέχεια η διευκόλυνση της αναζήτησης των στοιχείων του αντίστοιχου συνόλου.
- Για παράδειγμα, είναι γνωστό ότι η αναζήτηση ενός κλειδιού σε μη ταξινομημένο πίνακα με n εγγραφές γίνεται σειριακά με συγκρίσεις της τάξης $\Theta(n)$, ενώ σε ταξινομημένο πίνακα η δυαδική αναζήτηση απαιτεί κόστος της τάξης $\Theta(\log_2 n)$.

- *Η χρησιμότητα της ταξινόμησης αποδεικνύεται στην πράξη σε περιπτώσεις αναζήτησης αριθμητικών ή αλφαβητικών δεδομένων, όπως σε βιβλιοθηκονομικά συστήματα, λεξικά, τηλεφωνικούς καταλόγους, καταλόγους φόρου εισοδήματος και γενικά παντού όπου γίνεται αναζήτηση αποθηκευμένων αντικείμενων.*

Ορισμός

- Δοθέντων των στοιχείων $a_1, a_2, \ldots, a_n$ η ταξινόμηση συνίσταται στη διάταξης (permutation) της θέσης των στοιχείων, ώστε να τοποθετηθούν σε μία διάταξη

  $a_{k_1}, a_{k_2}, \ldots, a_{kn}$

  έτσι ώστε, δοθείσης μίας **συνάρτησης διάταξης** (ordering function), *f,* να ισχύει:

$$f(a_{k_1}) \leq f(a_{k_2}) \leq \ldots \leq f(a_{kn})$$

- Η προηγούμενη συνάρτηση διάταξης μπορεί να τροποποιηθεί, ώστε να καλύπτει και την περίπτωση όπου η ταξινόμηση γίνεται με φθίνουσα τάξη (descending sequence) μεγέθους του κλειδιού.

- Στη γενικότερη περίπτωση μπορεί να θεωρηθεί ότι η ταξινόμηση στηρίζεται σε δύο ή περισσότερα κλειδιά της εγγραφής.
- Για παράδειγμα, σε πολλά παιχνίδια της τράπουλας οι παίκτες ταξινομούν τα χαρτιά τους με πρώτο κλειδί το χρώμα του φύλλου (πχ. μπαστούνια, σπαθιά, καρά και κούπες) και με δεύτερο κλειδί την αξία του φύλλου (πχ. Άσσος, Ρήγας, Ντάμα, Βαλές, 10, ... , 2).
- Το αντικείμενο της ταξινόμησης είναι πολύ πλούσιο καθώς μπορεί να εφαρμοσθεί σε πολλά περιβάλλοντα (όπως, **εσωτερική** (*internal*) ταξινόμηση στην κύρια μνήμη ή **εξωτερική** (*external*) ταξινόμηση στη δευτερεύουσα μνήμη, και υπό πολλές συνθήκες (όπως, επιτοπίως (in situ, **in** place) ή με επιπλέον χώρο, ως προς τη στατιστική κατανομή των κλειδιών προς ταξινόμηση κλπ.).

- Στην ενότητα αυτή θα εξετάσουμε εκ νέου τους γνωστούς αλγορίθμους εσωτερικής ταξινόμησης ώστε να αποδείξουμε και τυπικά τις εκφράσεις των πολυπλοκοτήτων τους.
- Αρχικά θα εξετάσουμε μεθόδους ταξινόμησης που βασίζονται στη σύγκριση, ενώ στη συνέχεια θα μελετήσουμε άλλες μεθόδους.
- Επίσης, θα απαντήσουμε στο ενδιαφέρον ερώτημα: "**Πόσο γρήγορα μπορεί να γίνει η ταξινόμηση**" ,

  με σκοπό να αποδείξουμε το θεωρητικό όριο της πολυπλοκότητας των αλγορίθμων ταξινόμησης.

- Αρχικά θα εξετάσουμε τον αλγόριθμο της **ταξινόμησης με εισαγωγή** και θα καταλάβουμε τη συμπεριφορά του σε δύο περιπτώσεις δίνοντας στην είσοδο τους πίνακες A1=[1, 2, 3, 4, 5, 6] και A2=[6, 5, 4, 3, 2, 1], με στοιχεία που είναι ήδη ταξινομημένα με αύξουσα και φθίνουσα διάταξη αντίστοιχα.
- Παρατηρούμε ότι οι περιπτώσεις αυτές είναι ακραίες και δεν μπορούν να χαρακτηρισθούν ως η μέση περίπτωση, η περίπτωση δηλαδή όπου τα στοιχεία του πίνακα εμφανίζονται με μία τυχαία σειρά. Ακολουθεί ο ψευδοκώδικας για τον αλγόριθμό μας.

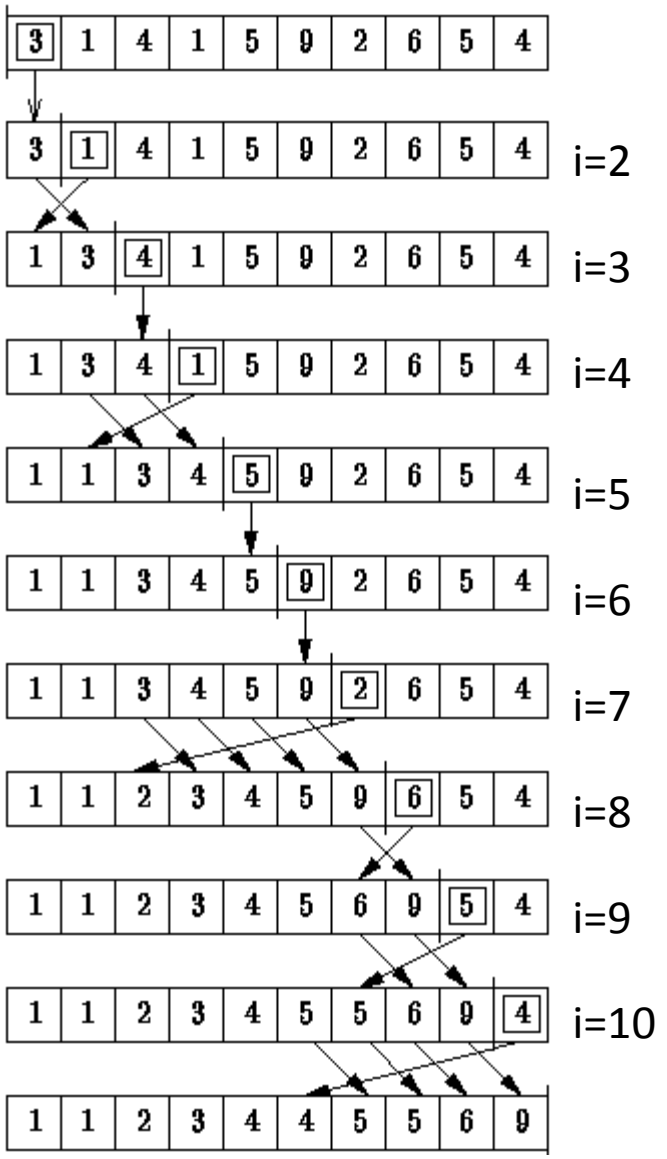
```
1. for i ← 2 to n do
2.     key ← A[i ];
3.     j ← i - 1;
4.     while j > 0 and key < A[j] do
5.         A[j+1] ← A[j]
6.         j ← j - 1
7.     A[j +1] ← key
```

Η μέθοδος αυτή διακρίνει τον πίνακα σε δύο τμήματα: την **ακολουθία πηγή** (*source sequence*) και την **ακολουθία προορισμού** (*destination sequence*).

- ✓Λαμβάνει στοιχεία από την ακολουθία πηγή και τα κατευθύνει στη σωστή θέση στην ακολουθία προορισμού.
- ✓Κάθε φορά το μέγεθος της ακολουθίας πηγής μειώνεται κατά ένα, ενώ το μέγεθος της ακολουθίας προορισμού αυξάνεται κατά ένα.
- ✓Σε κάθε επανάληψη το λαμβανόμενο στοιχείο είναι το πρώτο της ακολουθίας πηγής (εντολή 3).
- ✓Το στοιχείο αυτό συγκρίνεται με τα στοιχεία της ακολουθίας προορισμού αρχίζοντας από το τέλος και συνεχίζοντας προς την αρχή (από δεξιά προς αριστέρα) μέχρι να εντοπίσει την κατάλληλη θέση

Ένα Παράδειγμα

```
1. for i ← 2 to n do
2.     key ← A[i ];
3.     j ← i - 1;
4.     while j > 0 and key < A[j] do
5.         A[j+1] ← A[j]
6.         j ← j - 1
7.     A[j +1] ← key
```



Πρόταση.

Η πολυπλοκότητα της ταξινόμησης με εισαγωγή για την καλύτερη, τη μέση και τη χειρότερη περίπτωση είναι $\Theta(n)$, $\Theta(n^2)$ και $\Theta(n^2)$, αντιστοίχως

Απόδειξη

- ✓ **Αν στην είσοδο θεωρηθεί ο πίνακας *A1=[1, 2, 3, 4, 5, 6]* , τότε η ταξινόμηση συμπεριφέρεται πολύ αποτελεσματικά και δεν εισέρχεται μέσα στο σώμα της εντολής 4. Έτσι ουσιαστικά το κάθε φορά λαμβανόμενο στοιχείο από την ακολουθία πηγής συγκρίνεται με ένα μόνο στοιχείο (όπως προκύπτει από τη συνθήκη ελέγχου 4).**
- ✓ **Θεωρώντας αυτή τη σύγκριση ως την πράξη βαρόμετρο, καταλήγουμε ότι για ένα πίνακα μεγέθους *n* στοιχείων, ο αντίστοιχος αριθμός συγκρίσεων είναι *n* - 1. Επομένως προκύπτει ότι στην καλύτερη περίπτωση η πολυπλοκότητα είναι γραμμική *Θ(n).***

Τώρα ας θεωρήσουμε την περίπτωση όπου στην είσοδο δίνεται ο πίνακας A2 =[6, 5, 4, 3, 2, 1].

Το κάθε φορά επιλεγόμενο στοιχείο είναι μικρότερο από τα ήδη τοποθετημένα στην ακολουθία προορισμού.

Έτσι εκτελείται ένας συγκεκριμένος αριθμός συγκρίσεων μέχρι να τοποθετηθεί το νέο στοιχείο στη σωστή θέση, δηλαδή στην πρώτη θέση της ακολουθίας προορισμού.

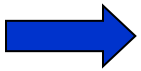
Έτσι η **εντολή 5** εκτελείται *i* φορές, όπου το *i* μεταβάλλεται από ***1*** **μέχρι *n* - 1**.

```
1.  for i ← 2 to n do
2.      key ← A[i ];
3.      j ← i - 1;
4.      while j > 0 and key < A[j] do
5.          A[j+1] ← A[j]
6.          j ← j - 1
7.  A[j +1] ← key
```

Θεωρώντας την πράξη αυτή ως βαρόμετρο, καταλήγουμε ότι για ένα πίνακα μεγέθους *n* στοιχείων , ο αντίστοιχος αριθμός συγκρίσεων είναι

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$$

η πολυπλοκότητα στη χειρότερη περίπτωση είναι τετραγωνική $\Theta(n^2)$.



Το σημαντικότερο ερώτημα προκύπτει στην περίπτωση ταξινόμησης τυχαίων δεδομένων εισόδου,
δηλαδή στην περίπτωση όπου οποιαδήποτε από τις *n!* διατάξεις των δεδομένων εισόδου μπορεί να εμφανισθεί με ίση πιθανότητα *1/n!.*

- ✓ Η επίδοση αυτή μπορεί να προκύψει θεωρώντας ότι το στοιχείο που κάθε φορά λαμβάνεται από την ακολουθία πηγής θα διανύσει το μισό δρόμο μέχρι την αρχή της ακολουθίας προορισμού.
- ✓ Όπως προηγουμένως, και πάλι θα προκύψει ότι στη μέση αυτή περίπτωση η επίδοση περιγράφεται από μία τετραγωνική συνάρτηση, επομένως και στην περίπτωση αυτή η πολυπλοκότητα είναι $\Theta(n^2)$.

Αναλυτική απόδειξη του της πολυπλοκότητας του αλγόριθμου

	κόστος	επαναλήψεις
1. **for** $i \leftarrow 2$ **to** n **do**	c_1	n
2. $key \leftarrow A[i]$;	c_2	$n-1$
3. $j \leftarrow i - 1$;	c_3	$n-1$
4. **while** $j > 0$ **and** $key < A[j]$ **do**	c_4	t_i
5. $A[j+1] \leftarrow A[j]$	c_5	t_i-1
6. $j \leftarrow j - 1$	c_6	t_i-1
7. $A[j+1] \leftarrow key$	c_7	$n-1$

Οπου t_i είναι το πλήθος των επαναλήψεων εκτέλεσης του while-loop (γραμμή 4) για κάθε τιμή του i. Η τιμή του i μεταβάλλεται από 2 μέχρι n.

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{i=2}^{n} t_i + \\ + c_5 \sum_{i=2}^{n} (t_i - 1) + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7 (n-1)$$

Ανάλυση καλύτερης περίπτωσης

Η καλύτερη περίπτωση συμβαίνει όταν ο πίνακας είναι ήδη ταξινομημένος. Για κάθε i = 2, 3, …, n, το στοιχείο *A*[*j*] είναι μικρότερο ή ίσο από το key όταν το *j* έχει την αρχική τιμή *i*-1. Δηλαδή, όταν *j* = *i* - 1, πάντα βρίσκει το κλειδί *A*[*j*] στην πρώτη φορά εκτέλεσης του while-loop.

Συνεπώς για t_i = 1 για *i* = 2, 3, …, *n*

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{i=2}^{n} t_i + c_5 \sum_{i=2}^{n} (t_i - 1) + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7(n-1)$$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{i=2}^{n} 1 + c_5 \sum_{i=2}^{n} (1-1) + c_6 \sum_{i=2}^{n} (1-1) + c_7(n-1)$$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$

$$T(n) = (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7) = \Theta(n)$$

Ανάλυση χειρότερης περίπτωσης

Η χειρότερη περίπτωση συμβαίνει αν ο πίνακας είναι ταξινομημένος σε φθίνουσα διάταξη. Στη περίπτωση αυτή το στοιχείο *A*[*j*] είναι μεγαλύτερο από το *key*. Ετσι θα πρέπει να συγκρίνουμε κάθε στοιχείο *A*[*i*] με κάθε στοιχείο του ταξινομημένου υποπίνακα *A*[1 .. *i* − 1] και συνεπώς $t_i = i$ for $i = 2, 3, \ldots, n$

Δηλαδή, το while-loop θα εκτελεστεί μέχρι το j να γίνει 0, άρα ένας επιπλέον έλεγχος μετά τους i – 1 ελέγχους.

Αρα $t_i = i$ for $i = 2, 3, \ldots, n$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{i=2}^{n} t_i + c_5 \sum_{i=2}^{n} (t_i - 1) + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7(n-1)$$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{i=2}^{n} i + c_5 \sum_{i=2}^{n} (i-1) + c_6 \sum_{i=2}^{n} (i-1) + c_7(n-1)$$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \frac{n(n-1)}{2} + c_5 \frac{(n-1)(n-2)}{2} + c_6 \frac{(n-1)(n-2)}{2} + c_7(n-1)$$

$$= \Theta(n^2)$$

Ανάλυση μέσης περίπτωσης

Συνήθως μας ενδιαφέρει η μελέτη της πολυπλοκότητας της χειρότερης περίπτωσης καθώς εκφράζει τον μεγαλύτερο χρόνο εκτέλεσης για οποιαδήποτε τιμή του n.

Ο πολυπλοκότητα της χειρότερης περίπτωσης μας δίνει ένα άνω όριο του χρόνου εκτέλεσης, δηλαδή αυτό το άνω όριο μας δίνει την εγγύηση ότι ο αλγόριθμος δεν θα χρειαστεί σε καμιά περίπτωση μεγαλύτερο χρόνο εκτέλεσης.

Για τον αλγόριθμο της ταξινόμησης με εισαγωγή υποθέτουμε ότι επιλέγονται με τυχαίο τρόπο οι n αριθμοί.

Κατά μέσον όρο το $A[i]$ είναι μικρότερο από τα μισά στοιχεία του υποπίνακα $A[1 .. i - 1]$ και μεγαλύτερο από τα υπόλοιπα μισά. Αυτό σημαίνει ότι το while-loop θα πρέπει να κάνει συγκρίσεις με τα μισά στοιχεία του υποπινάκα $A[1 .. i - 1]$ για να αποφασίσει που θα τοποθετήσει το *key*.

Που σημαίνει ότι $t_i = i/2$

Συνεπώς παρά το γεγονός ότι ο χρόνος εκτέλεσης της μέσης περίπτωσης είναι κατά προσέγγιση ο μισός της χειρότερης περίπτωσης παρ' όλα αυτά είναι τετραγωνική συνάρτηση του n.

Ταξινόμηση με Επιλογή

Αλγόριθμος select

1. **for** $i \leftarrow 1$ to n-1 do
2. $min \leftarrow i$;
3. **for** $j \leftarrow i$+1 to n **do**
4. **if** $A[j] < A[min]$ then
5. $min \leftarrow j$
6. SWAP($A[i]$, $A[min]$)

Πρόταση.

Η πολυπλοκότητα της ταξινόμησης με επιλογή είναι $\Theta(n^2)$ σε κάθε περίπτωση.

Απόδειξη

Ο αλγόριθμος σαρώνει τον πίνακα ώστε να εντοπίσει το μικρότερο στοιχείο και να το τοποθετήσει στην πρώτη θέση. Στη συνέχεια περιορίζει την αναζήτηση στα υπόλοιπα *n*-1 στοιχεία ώστε να εντοπίσει το μικρότερο μεταξύ αυτών και να το τοποθετήσει στη δεύτερη θέση κοκ. Επομένως, εύκολα προκύπτει ότι η εντολή 4

if A **[*j*] <** A **[*min*]**

θα εκτελεσθεί τον ίδιο αριθμό επαναλήψεων ανεξαρτήτως του περιεχομένου του πίνακα εισόδου. Είτε, λοιπόν, στην είσοδο δοθεί ο πίνακας Α1 είτε ο Α2 (όπως αναφέρονται στον προηγούμενο αλγόρθμο), το αντίστοιχο πλήθος συγκρίσεων θα είναι το ίδιο.

Πραγματοποιείται μια σύγκριση σε κάθε επανάληψη του εσωτερικού βρόχου, για κάθε δηλαδή τιμή της μεταβλητής *j* μεταξύ των ορίων *i* + 1 και *n.* Αυτό επαναλαμβάνεται για κάθε επανάληψη του εξωτερικού βρόχου, για κάθε τιμή της μεταβλητής *i* μεταξύ των ορίων 1 και *n* -1.

Οπότε έχουμε το πλήθος των συγκρίσεων θα είναι: $\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}1$

Γνωρίζουμε για το άθροισμα :

$$\sum_{j=i+1}^{n}1 = n-(i+1)+1$$

Οπότε:

$$\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}1 = \sum_{i=1}^{n-1}\left[n-(i+1)+1\right] = \sum_{i=1}^{n-1}\left[n-i-1+1\right] = \sum_{i=1}^{n-1}(n-i) =$$

$$= (n-1)+(n-2)+\ldots+(n-n+1) = 1+2+\ldots+(n-2)+(n-1) =$$

$$\frac{(n-1)n}{2} = \frac{n^2-n}{2} \in \Theta(n^2)$$

- http://courses.cs.vt.edu/csonline/Algorithms/Lessons/index.html
- http://www.personal.kent.edu/~rmuhamma/Algorithms/algorithm.html
- https://www.coursera.org/course/algo

Τέλος Ενότητας