Δομές Δεδομένων

9η Διάλεξη
Ταξινόμηση - Στοιχειώδεις μέθοδοι

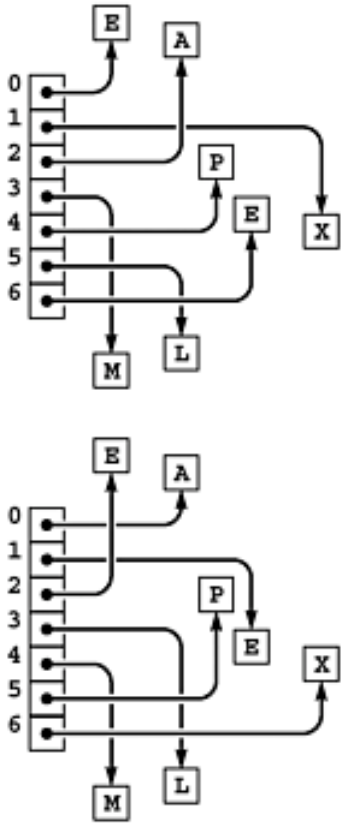
Ε. Μαρκάκης

Περίληψη

- Bubble Sort
- Selection Sort
- Insertion Sort
- Χαρακτηριστικά επιδόσεων
- Shellsort
- Ταξινόμηση συνδεδεμένων λιστών

Γενικές υλοποιήσεις - Υπενθύμιση

- Ιδιότητες γενικής υλοποίησης
  - Λειτουργεί με οποιονδήποτε τύπο αντικειμένων
  - Όχι τόσο αποδοτική για ενσωματωμένους τύπους
    - Απαιτείται ένα πρόσθετο επίπεδο αναφορών
  - Στην πράξη όμως εργαζόμαστε με αντικείμενα (π.χ. εγγραφές) και όχι με στοιχειώδεις τύπους
  - Υλοποιεί ταξινόμηση δεικτών
- Ταξινόμηση δεικτών
  - Οι πίνακες αντικειμένων περιέχουν αναφορές
  - Η αντιμετάθεση γίνεται σε επίπεδο αναφορών
  - Αποφεύγεται η αντιγραφή των αντικειμένων
  - Το κόστος της αντιμετάθεσης είναι μικρό
    - Ανάλογο με την αντιμετάθεση ακεραίων



Γενικές υλοποιήσεις - Υπενθύμιση

- μέθοδος sort(): Καλείται από τους πελάτες με πίνακα τύπου ITEM και δύο όρια για το εύρος του πίνακα

```
class Sort {
 // βοηθητικές μέθοδοι
     static boolean less(ITEM v, ITEM w) {...}
   //κάνει σύγκριση 2 στοιχείων

   static void exch(ITEM[] a, int i, int j) {... }
   // αντιμεταθέτει τα στοιχεία στις θέσεις i και j

   static void compExch(ITEM[] a, int i, int j)
     {...} //συγκρίνει και αντιμεταθέτει αν χρειάζεται

     static void sort(ITEM[] a, int p, int r) {...}
```

Ταξινόμηση φυσαλίδας – Bubble Sort

- Λειτουργία αλγορίθμου
  - Από τις πιο παλιές και απλοϊκές μεθόδους
  - Σαρώνουμε από τα δεξιά προς τα αριστερά
  - Αντιμεταθέτουμε γειτονικά στοιχεία όταν χρειάζεται
  - Το μικρότερο στοιχείο φτάνει στο κάτω άκρο
  - Το αριστερό όριο του πίνακα κινείται μία θέση δεξιά
  - Επαναλαμβάνουμε την ίδια διαδικασία
  - Μετά την επανάληψη i: ταξινομημένο το τμήμα a[0],…,a[i]

```
static void bubble(ITEM[] a, int p, int r) {
  for (int i = p; i < r; i++)
       for (int j = r; j > i; j--)
            compExch(a, j-1, j);
}
```

Ταξινόμηση φυσαλίδας – Bubble Sort

- Σχετικά αργή μέθοδος
  - Πολλές αντιμεταθέσεις σε κάθε επανάληψη!
  - Επιδέχεται βελτιστοποίηση
- Πολυπλοκότητα
  - Worst case: πίνακας σε φθίνουσα σειρά
  - $(N-1)+ (N-2) + \ldots+1 \approx N^2/2 = O(N^2)$ συγκρίσεις και αντιμεταθέσεις
  - Average case: πιο πολύπλοκη η ανάλυση για τον ακριβή αριθμό αντιμεταθέσεων αλλά τελικά $N^2/2 = O(N^2)$
  - Σίγουρα απαιτούνται $O(N^2)$ συγκρίσεις

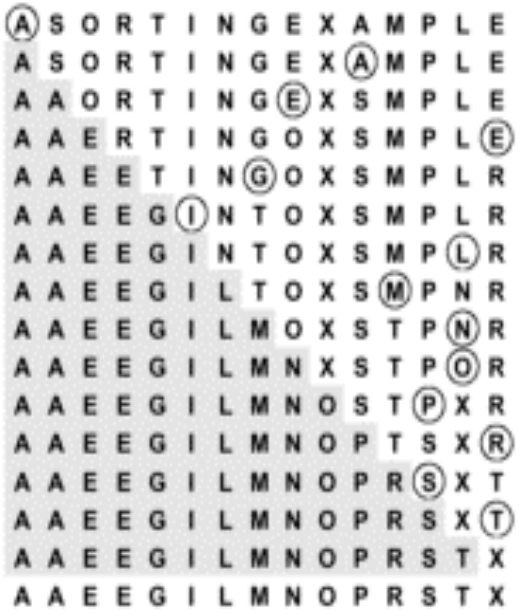
Ταξινόμηση με επιλογή - Selection Sort

- Λειτουργία αλγορίθμου
  - Εντοπίζουμε το μικρότερο στοιχείο του πίνακα
    - Σαρώνουμε όλα τα στοιχεία με τη σειρά
  - Το αντιμεταθέτουμε με αυτό στην πρώτη θέση
  - Επαναλαμβάνουμε από την επόμενη θέση
    - Κάθε πέρασμα μειώνει τη σάρωση κατά μία θέση

```
static void selection(ITEM[] a, int p, int r) {
  for (int i = p; i < r; i++) {
       int min = i;
       for (int j = i+1; j <= r; j++)
           if (less(a[j], a[min])) min = j;//εύρεση min
       exch(a, i, min);
       }
}
```

Ταξινόμηση με επιλογή - Selection Sort

- Αναπαράσταση αλγορίθμου
  - Ο πίνακας χωρίζεται σε δύο τμήματα
    - Αριστερός υποπίνακας: ταξινομημένος
    - Δεξιός υποπίνακας: μη ταξινομημένος
  - Σε κάθε πέρασμα το όριο αλλάζει
    - Εντοπίζουμε το μικρότερο στοιχείο στον δεξιό υποπίνακα
    - Το ενσωματώνουμε στον αριστερό υποπίνακα



- Πολυπλοκότητα της Selection Sort
  - Δεν μπορούμε να δούμε αν βρήκαμε ήδη το ελάχιστο στοιχείο
  - Worst case: O(N) αντιμεταθέσεις, $O(N^2)$ συγκρίσεις,, π.χ. φθίνουσα σειρά
  - Average case: επίσης $O(N^2)$ (πάντα κάνει $O(N^2)$ συγκρίσεις)

Ταξινόμηση με εισαγωγή – Insertion Sort

- Την είδαμε στο προηγούμενο μάθημα:

```
static void sort(ITEM[] a, int p, int r) {
      for (int i = p+1; i <= r; i++)
         for (int j = i; j > p; j--)
                  compExch(a, j-1, j); } }
```

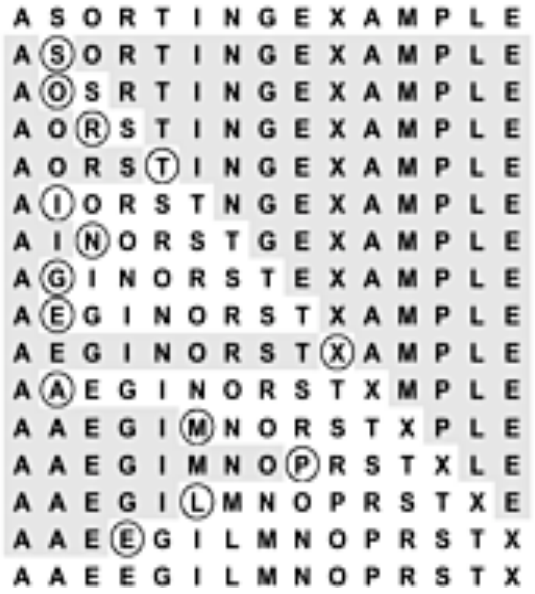
- Πολυπλοκότητα $O(N^2)$
- Βελτιστοποιήσεις:
  - Μπορούμε να σταματήσουμε να εκτελούμε την compExch όταν το κλειδί μπει στη σωστή θέση
  - Διαδοχικές αντιμεταθέσεις που αφορούν το ίδιο στοιχείο δεν είναι αποδοτικές
  - Ο έλεγχος στο 2^o for μπορεί να αποφευχθεί με χρήση στοιχείου φρουρού

Ταξινόμηση με εισαγωγή – Insertion Sort

```
static void insertion(ITEM[] a, int p, int r) {
  int i;
  // τοποθέτηση του ελάχιστου στην πρώτη θέση
  for (i = r; i > p; i--)
       compExch(a, i-1, i);
  for (i = p+2; i <= r; i++) {
       int j = i; ITEM v = a[i];
       // δεν χρειάζεται πρόσθετος έλεγχος
       while (less(v, a[j-1])) {
            // μετακίνηση του j-1 και όχι
  αντιμετάθεση
            a[j] = a[j-1];
            j--; }
       a[j] = v; } //βάζουμε το v στη σωστή θέση
}
```

Ταξινόμηση με εισαγωγή – Insertion Sort

- Λειτουργία βελτιωμένου αλγορίθμου
  - Πάλι ο πίνακας χωρίζεται σε υποπίνακες
  - Αρχικά ελάχιστο στοιχείο πάει μπροστά
  - Παίρνουμε το πρώτο στοιχείο του δεξιού
  - Το εισάγουμε στο σωστό σημείο του αριστερού
  - Το όριο των πινάκων κινείται μία θέση δεξιά



Ταξινόμηση με εισαγωγή – Insertion Sort

- Βελτιώσεις ταξινόμησης με επιλογή
  - Μπορούμε να δούμε αν βρήκαμε τη σωστή θέση
    - Ο αριστερός υποπίνακας είναι ήδη ταξινομημένος
  - Δεν χρειάζονται χωριστοί έλεγχοι τερματισμού
    - Απλά βάζουμε το μικρότερο στοιχείο στην αρχή
    - Παίζει το ρόλο φρουρού
  - Δεν χρειάζονται πλήρεις αντιμεταθέσεις
    - Αρκεί η μετακίνηση των στοιχείων μία θέση δεξιά
    - Στο τέλος αντιγράφεται το αρχικό στοιχείο
- Πολυπλοκότητα
  - Worst case: παραμένει $O(N^2)$ (π.χ. Όταν τα στοιχεία είναι σε φθίνουσα σειρά)
  - Είναι όμως αρκετά πιο γρήγορη από την αρχική υλοποίηση

Στοιχειώδεις Ταξινομήσεις

- Για περισσότερες λεπτομέρειες δείτε εδώ:
- Insertion Sort
- **http://www.youtube.com/watch?v=ejpFmtYM8Cw**

- Selection Sort
- **http://www.youtube.com/watch?v=TW3_7cD9L1A**

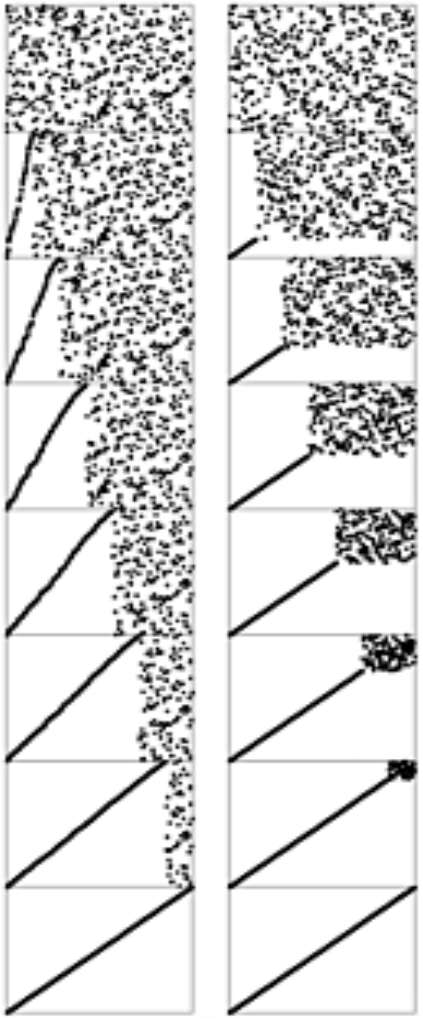
- Bubble Sort
- **http://www.youtube.com/watch?v=P00xJgWzz2c**

Χαρακτηριστικά επιδόσεων

	Συγκρίσεις Worst Case	Συγκρίσεις Average Case	Αντιμεταθέσεις ή μετακινήσεις Worst Case	Αντιμεταθέσεις ή μετακινήσεις Average Case
Bubble Sort	$N^2/2$	$N^2/2$	$N^2/2$	$N^2/2$
Selection Sort	$N^2/2$	$N^2/2$	N	N
Insertion Sort	$N^2/2$	$N^2/4$	$N^2/2$	$N^2/4$

Χαρακτηριστικά επιδόσεων

- Όλες οι στοιχειώδεις ταξινομήσεις είναι $O(N^2)$
  - Αυτό δεν σημαίνει ότι είναι ίδιες όμως!
  - Παράδειγμα: εισαγωγή και επιλογή
    - Εισαγωγή: δεν κοιτάζει πέρα από τη διαγώνιο
    - Επιλογή: δεν κοιτάζει πίσω από τη διαγώνιο
- Ταξινόμηση με επιλογή
  - Κάνει πάντοτε λίγες αντιμεταθέσεις
  - Δεν εξαρτάται καθόλου από την είσοδο
    - Δεν μπορεί να γίνει προσαρμοστική



Χαρακτηριστικά επιδόσεων

- Ταξινόμηση με εισαγωγή
  - Υποθέτουμε προσαρμοστική έκδοση
  - Οι συγκρίσεις σταματάνε όταν το στοιχείο μπει στη θέση του
  - Συγκρίσεις και μετακινήσεις πάνε μαζί
  - Worst case έχει διπλάσια πολυπλοκότητα σε σχέση με average case
- Ταξινόμηση φυσαλίδας
  - Προσαρμοστική έκδοση: έλεγχος αν έγιναν αντιμεταθέσεις στην προηγούμενη επανάληψη του βρόχου
  - Σταματάμε όταν το αρχείο είναι ταξινομημένο
  - Συγκρίσεις και αντιμεταθέσεις πάνε μαζί

- Insertion Sort και Bubble Sort αποδίδουν καλά σε πολλά μερικώς ταξινομημένα αρχεία που συναντώνται στην πράξη (όχι όμως και η Selection Sort)

Χαρακτηριστικά επιδόσεων

- Αντιστροφή (Inversion): ένα ζεύγος κλειδιών εκτός σειράς
  - Συνολικός αριθμός αντιστροφών: για κάθε στοιχείο μετράμε τα μεγαλύτερα κλειδιά αριστερά του
  - Λίγες αντιστροφές σημαίνει μερικώς ταξινομημένο αρχείο
- Σταθερό πλήθος αντιστροφών ανά στοιχείο
  - Τα κλειδιά είναι κοντά στις τελικές τους θέσεις
  - Οι ταξινομήσεις εισαγωγής και φυσαλίδας είναι γραμμικές (O(N))
  - Ταξινόμηση με εισαγωγή είναι πάντα ανάλογη με το πλήθος των αντιστροφών
  - Για ταξινόμηση φυσαλίδας απόδειξη πιο πολύπλοκη
- Σταθερό πλήθος στοιχείων με μη σταθερές αντιστροφές
  - Λίγα κλειδιά βρίσκονται σε λάθος θέσεις
  - Η ταξινόμηση εισαγωγής είναι γραμμική
- Αρχεία με πολύ μεγάλα στοιχεία και μικρά κλειδιά
  - Το κόστος των μεταθέσεων κυριαρχεί επί των συγκρίσεων
  - Η ταξινόμηση με επιλογή είναι γραμμική

Πειραματική Μελέτη

	Στοιχεία τύπου int				Στοιχεία με κλειδιά τύπου int			Στοιχεία με κλειδιά τύπου string		
N	S	I*	I	B	S	I	B	S	I	B
1000	14	54	8	54	100	55	130	129	65	170
2000	54	218	33	221	436	229	569	563	295	725
4000	212	848	129	871	1757	986	2314	2389	1328	3210

- S: Selection Sort
- I*: Insertion Sort
- I: βελτιωμένη Insertion Sort
- B: Bubble Sort

Ταξινόμηση Shellsort

- Προτάθηκε από τον Shell (1959)
- Επέκταση της Insertion Sort
- Μειονεκτήματα προηγούμενων μεθόδων:
  - Συγκρίσεις μόνο μεταξύ γειτονικών στοιχείων
  - Αν το ελάχιστο είναι στο τέλος χρειάζονται πολλές μετακινήσεις ή αντιμεταθέσεις για να πάει στην αρχή
- Ιδέα: ταξινόμηση ανά h στοιχεία
- Ένας πίνακας είναι *h-sorted* αν οι υποπίνακες

  a[0], a[h], a[2h],…

  a[1], a[h+1], a[2h+1],…

  a[2], a[h+2], a[2h+2],…

  είναι ταξινομημένοι

Ταξινόμηση Shellsort

- Λειτουργία του αλγορίθμου:
- Ξεκινάμε με κάποια μεγάλη τιμή του h
- Κάνουμε τον πίνακα h-sorted
  - Ίδιος κώδικας με Insertion Sort με τους δείκτες στους βρόχους να αυξομειώνονται κατά h
- Μειώνουμε την τιμή του h και επαναλαμβάνουμε
- Όταν το h γίνει 1, ουσιαστικά καταλήγουμε σε Insertion Sort αλλά με πολύ μικρότερο αριθμό συγκρίσεων
- Ερώτηση: τι τιμές πρέπει να διαλέξουμε για το h?
- Ουσιαστικά χρειαζόμαστε μία ακολουθία τιμών

Ταξινόμηση Shellsort

Version που χρησιμοποιεί την ακολουθία 1, 4, 13, 40, 121, 364,...

δηλαδή $h_{i+1} = 3h_i + 1$

```
static void shell(ITEM[] a, int p, int r)
{ int h;
  for (h = 1; h <= (r-l)/9; h = 3*h+1);
  for ( ; h > 0; h /= 3)
    for (int i = p+h; i <= r; i++) {
        int j = i;
        ITEM v = a[i];
        while (j >= p+h && less(v, a[j-h]))
        { a[j] = a[j-h]; j -= h; }
        a[j] = v;
     } }
```

Ταξινόμηση Shellsort

- h = 1, 4, 13, 40,…
- Προτάθηκε από τον Knuth (1969)
- Πρώτα γίνεται 13-ταξινόμηση
- Μετά 4-ταξινόμηση
- Και στο τέλος 1-ταξινόμηση, δηλαδή Insertion Sort
- Πολύ μικρότερος αριθμός συγκρίσεων κατά την Insertion Sort
- Μακρινές μετακινήσεις γίνονται στις μεγάλες τιμές του h

Ταξινόμηση Shellsort - Πολυπλοκότητα

- Η πολυπλοκότητα εξαρτάται από την ακολουθία $\{h_i\}$
- Δεν έχει βρεθεί ακόμα αποδεδειγμένα η καλύτερη ακολουθία (ανοιχτό ερευνητικό πρόβλημα)
- Στην πράξη χρησιμοποιούνται ακολουθίες που μειώνονται γεωμετρικά
- Με κατάλληλες ακολουθίες, η shellsort είναι αρκετά πιο γρήγορη από τις στοιχειώδεις ταξινομήσεις

Ταξινόμηση Shellsort - Πολυπλοκότητα

- Τι γνωρίζουμε μέχρι σήμερα:

- **Ιδιότητα 1:** Η shellsort με την ακολουθία του Knuth 1, 4, 13, 40,… εκτελεί λιγότερες από $O(N^{3/2})$ συγκρίσεις
- **Ιδιότητα 2:** Η shellsort με την ακολουθία 1, 8, 23, 77, 281… ($h_i = 4^{i+1} + 3\ 2^i + 1$) εκτελεί λιγότερες από $O(N^{4/3})$ συγκρίσεις
- **Ιδιότητα 3:** Η shellsort με την ακολουθία 1, 2, 3, 4, 6, 9, 8, 12, 18, 27, 16, 24, 36, 54, 81,… εκτελεί λιγότερες από $O(N (\log N)^2)$ συγκρίσεις (Pratt 1971)

Ταξινόμηση συνδεδεμένων λιστών

- Προσαρμογή ταξινομήσεων πινάκων
  - Βασικός κανόνας: τροποποιούμε δείκτες και όχι κόμβους
    - Οι κόμβοι μπορεί να έχουν και εξωτερικούς δείκτες
  - Αλλάζουμε τη δομή και όχι το περιεχόμενο της λίστας
- Selection Sort: προαρμογή σε λίστες
  - Λίστα εισόδου: μη ταξινομημένη
  - Λίστα εξόδου: ταξινομημένη
  - Εντοπισμός μέγιστου στοιχείου
    - Το μέγιστο από όσα έχουν μείνει
  - Αποσύνδεση από λίστα εισόδου
  - Σύνδεση στην αρχή της εξόδου
    - Μικρότερο από τα επόμενα
  - Λίστα εξόδου σε αύξουσα σειρά

Ταξινόμηση συνδεδεμένων λιστών

- Χρήση κόμβου φρουρού στην αρχή της λίστας εισόδου

```
private static Node findMax(Node h) {
  for (Node t = h; t.next != null; t = t.next)
       if (h.next.item < t.next.item) h = t;
  return h; }
static Node sort(Node h) {
  Node head = new Node(-1, h), out = null;
  while (head.next != null) {
    Node max = findMax(head);
    Node t = max.next;
    max.next = t.next;
    t.next = out;
    out = t; }
  return out; }
```