Δομές Δεδομένων

10η Διάλεξη

Ταξινόμηση

Ε. Μαρκάκης

Περίληψη

- Ταξινόμηση με αριθμοδείκτη κλειδιού
- Ταξινόμηση με συγχώνευση – Αλγόριθμος Mergesort
- Διμερής συγχώνευση
- Αφηρημένη επιτόπου συγχώνευση
- Αναλυτική ταξινόμηση Mergesort
- Συνθετική ταξινόμηση Mergesort
- Χαρακτηριστικά επιδόσεων
- Υλοποιήσεις Mergesort με λίστες

Καταμέτρηση με αριθμοδείκτη

- Ταξινόμηση ειδικής μορφής
- Αποφυγή συγκρίσεων αν εκμεταλλευθούμε ειδικές ιδιότητες των κλειδιών
  - Έστω N κλειδιά με διακεκριμένες τιμές από 0 έως N-1

    ```
    for (i = 0; i < N; i++) b[a[i].key] = a[i];
    ```
- Καταμέτρηση με αριθμοδείκτη κλειδιού
  - Έστω N κλειδιά που παίρνουν τιμές 0 ή 1
  - Πρώτη διέλευση: μέτρηση των στοιχείων με κλειδί 0 και αντίστοιχα με κλειδί 1
  - Χρήση πίνακα μετρητών με 2 στοιχεία, cnt[0], cnt[1] και πίνακα b με N στοιχεία
  - cnt[0]: δείχνει αρχικά την θέση που πρέπει να μπει το πρώτο 0 στον b
  - cnt[1]: θέση που πρέπει να μπει το πρώτο στοιχείο με κλειδί 1 στον b
  - Δεύτερη διέλευση στον πίνακα: μεταφέρουμε κάθε κλειδί στη σωστή θέση στον b και αυξάνουμε τον αντίστοιχο μετρητή

    ```
    for (i = 0; i < N; i++) b[cnt[a[i]]++] = a[i];
    ```

Καταμέτρηση με αριθμοδείκτη

- Γενίκευση σε μεγαλύτερο αριθμό κλειδιών
  - Έστω N κλειδιά με τιμές 0 έως M-1
    - Το M πολύ μικρότερο από το N
    - Πιθανόν πολλά κλειδιά με ίδιες τιμές
  - Πρώτη διέλευση: μετράμε το πλήθος κλειδιών για κάθε τιμή
    - Χρήση ενός πίνακα μετρητών cnt[] M στοιχείων
    - Χρήση πίνακα b για προσωρινή αποθήκευση των στοιχείων
  - Μετατροπή των μετρητών σε αθροιστικούς, π.χ. cnt[k] = πόσα στοιχεία έχουν κλειδί μικρότερο από k = θέση που πρέπει να μπει το πρώτο στοιχείο με κλειδί k
  - Δεύτερη διέλευση: μεταφέρουμε κάθε κλειδί στη σωστή θέση
    - Ο αντίστοιχος μετρητής αυξάνεται
    - Η θέση για το επόμενο θα δίνεται πάντα από τον μετρητή
  - Στο τέλος μεταφέρουμε τα στοιχεία πίσω στον a

Καταμέτρηση με αριθμοδείκτη

- Σειρά περασμάτων στους πίνακες
  - 1ο: αρχικοποίηση M μετρητών
  - 2ο: μέτρηση κλειδιών
  - 3ο: μετατροπή M μετρητών σε αθροιστικούς
  - 4ο: μεταφορά N κλειδιών στις σωστές θέσεις
  - 5ο: επαναφορά N κλειδιών στον αρχικό πίνακα

```
static void distCount(int a[], int p, int r) {
  int i, j, cnt[] = new int[M];
  int b[] = new int[a.length];
  for (j = 0; j < M; j++) cnt[j] = 0;
  for (i = p; i <= r; i++) cnt[a[i]+1]++;
  for (j = 1; j < M; j++) cnt[j] += cnt[j-1];
  for (i = p; i <= r; i++) b[cnt[a[i]]++] = a[i];
  for (i = p; i <= r; i++) a[i] = b[i-p]; }
```

Καταμέτρηση με αριθμοδείκτη

- Πόσο εξοικονομούμε με την ταξινόμηση με αριθμοδείκτη?

- Θεώρημα: Η καταμέτρηση με αριθμοδείκτη γίνεται σε χρόνο O(N) υπό την προϋπόθεση ότι το πλήθος των διαφορετικών τιμών των κλειδιών M είναι O(N)
  - Αρκεί να δούμε τις πράξεις στους 5 βρόχους
  - Κάθε βρόχος εκτελείται σε O(M) ή σε O(N)
  - Συνολική πολυπλοκότητα είναι O(M+N), άρα O(N)

Κεφάλαιο 8

Ταξινόμηση με συγχώνευση

Αλγόριθμος Mergesort

Ταξινόμηση με συγχώνευση

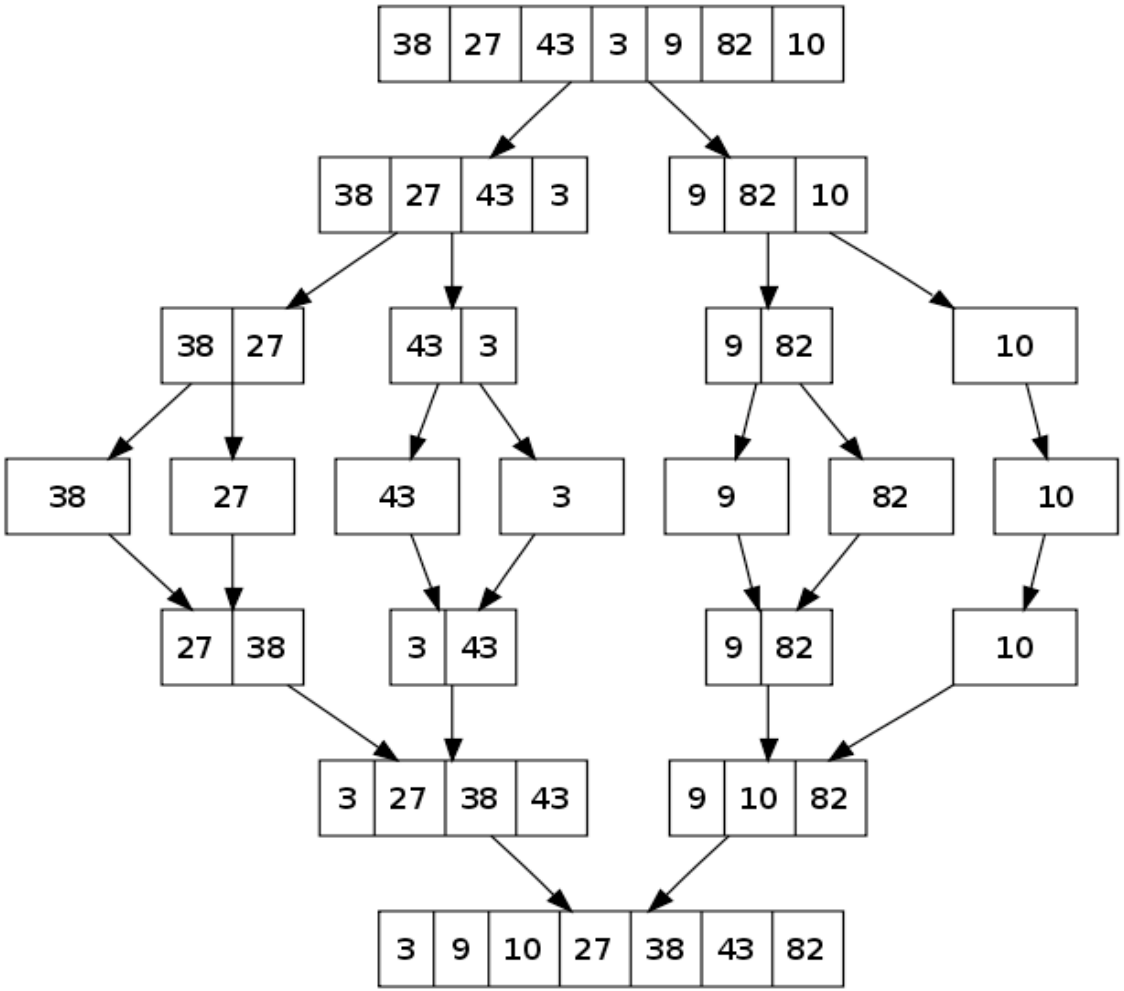
- Βασική ιδέα αλγορίθμων συγχώνευσης:
- Διαίρει και βασίλευε
- Χωρισμός σε 2 υποαρχεία
- Χωριστή ταξινόμηση σε κάθε αρχείο
- Συγχώνευση των 2 υποαρχείων σε ένα ταξινομημένο αρχείο
- Υλοποίηση συνήθως με αναδρομή (μπορούμε και χωρίς αναδρομή)
- Ευσταθής μεθοδος αν είμαστε λίγο προσεκτικοί
- Ακολουθιακή πρόσβαση στα δεδομένα, κατάλληλη για μηχανηματα υψηλής απόδοσης με γρήγορη πρόσβαση σε δεδομένα
- 1η υλοποίηση: von Neumann στον υπολογιστή EDVAC (1945)

Ταξινόμηση με συγχώνευση

- Αλγόριθμος Mergesort
- Αναλυτική (top-down) ταξινόμηση με συγχώνευση
  - 2 αναδρομικές κλήσεις σε αρχεία με το μισό αρχικό μέγεθος
  - Αναδρομή τελειώνει όταν το αρχείο έχει μόνο ένα στοιχείο
  - Χρησιμοποιεί τη μέθοδο merge στο τέλος για να συγχωνεύσει τους 2 υποπίνακες

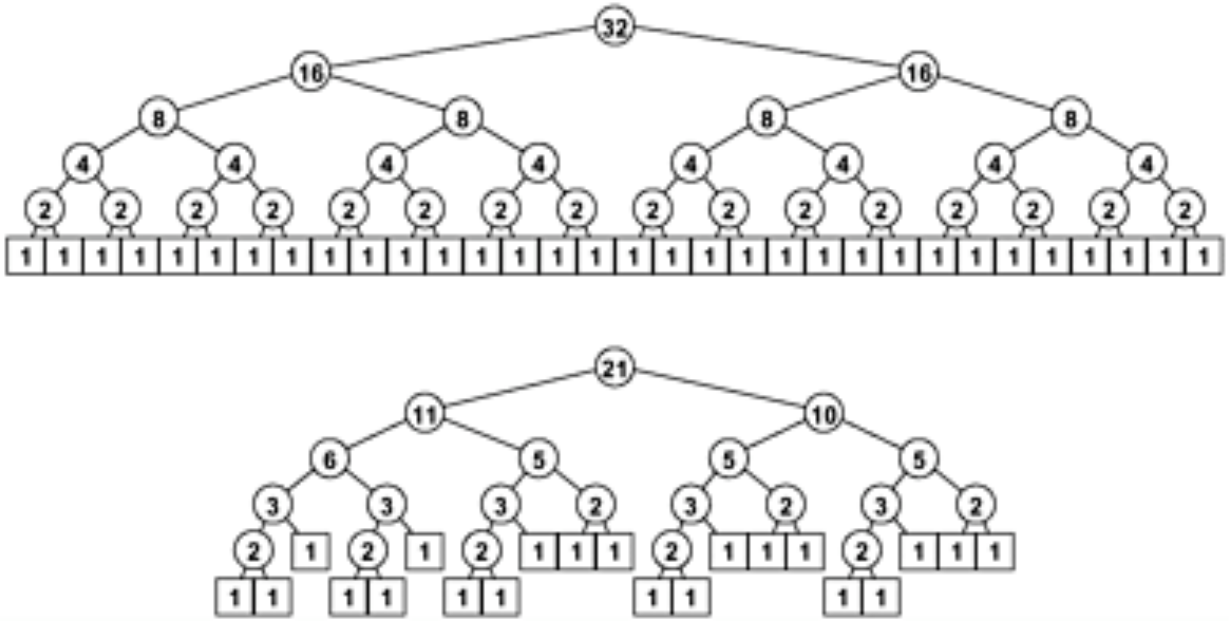
```
static void mergesort(ITEM[] a, int p, int r) {
  if (r <= p) return;
  int m = (r+p)/2;
  mergesort(a, p, m);
  mergesort(a, m+1, r);
  merge(a, p, m, r);/*συγχωνεύει τους ταξινομημένους
  υποπίνακες a[p..m] και a[m+1..r]*/
}
```

Ταξινόμηση με συγχώνευση



Ταξινόμηση με συγχώνευση

- Δέντρα «διαίρει και βασίλευε»
- Μας δείχνουν πώς αποσυντίθεται το αρχικό πρόβλημα σε υποπροβλήματα μικρότερου μεγέθους
- Στη Mergesort, είναι ανεξάρτητα από δεδομένα εισόδου



Υλοποίηση συγχώνευσης

- Τελικό βήμα της Mergesort
- Θα δούμε πρώτα μία πιο γενική υλοποίηση
- Είσοδος: 2 ταξινομημένοι πίνακες a και b
- Έξοδος: Συγχώνευση των a, b σε ένα ταξινομημένο πίνακα c
- Σε κάθε βήμα:
  - Εντοπίζουμε το μικρότερο στοιχείο στους a, b
  - Το αντιγράφουμε στο νέο πίνακα
  - Ενημερώνουμε τους δείκτες
- Προσοχή στα όρια των πινάκων!

Υλοποίηση συγχώνευσης

- Πίνακες εισόδου: a με όρια ap, ar, και b με όρια bp br
- Πίνακας εξόδου: c με αρχικό δείκτη cp
- Μέγεθος του c πρέπει να είναι τουλάχιστον cp + (ar- ap +1) + (br- bp +1)

```
static void mergeAB(ITEM[] c, int cp, ITEM[] a, int ap,
   int ar, ITEM[] b, int bp, int br )
{
        int i = ap, j = bp;
        for (int k = cp; k < cp+ar-ap+br-bp+1; k++) {
              if (i > ar) { c[k] = b[j++]; continue; }
             if (j > br) { c[k] = a[i++]; continue; }
             c[k] = less(a[i], b[j]) ? a[i++] : b[j++]; } }
```

Αφηρημένη επιτόπου συγχώνευση

- $2^{η}$ υλοποίηση:
- Συγχώνευση απευθείας στον πίνακα εισόδου
  - Οι δύο υποπίνακες βρίσκονται στον πίνακα εισόδου
    - Αρκεί να ξέρουμε σε ποιον δείκτη m χωρίζονται
    - a[p] έως a[m] και a[m+1] έως a[r]
  - Η έξοδος θα είναι και αυτή στον πίνακα εισόδου
- Αποφυγή των ελέγχων τερματισμού
  - Σε κάθε βήμα ελέγχουμε αν τελείωσε κάθε υποπίνακας
  - Πώς μπορούμε να αποφύγουμε τους ελέγχους;
  - Ιδέα: αντιστροφή δεύτερου υποπίνακα
    - Τα δύο μεγαλύτερα στοιχεία γειτονεύουν
    - Το μεγαλύτερο από τα δύο χρησιμεύει ως φρουρός!

Αφηρημένη επιτόπου συγχώνευση

- Υλοποίηση επιτόπου συγχώνευσης
  - Χρησιμοποιεί τον βοηθητικό πίνακα aux
  - Ο δεύτερος υποπίνακας αντιστρέφεται
  - Το μέγιστο στοιχείο είναι στο a[m] ή στο a[m+1]


```
static void merge(ITEM[] a, int p, int m, int r) {
  int i, j;
  for (i = m+1; i > p; i--)
       aux[i-1] = a[i-1];//στο τέλος i=p
  for (j = m; j < r; j++)
       aux[j+1] = a[r+m-j];//στο τέλος j=r
  for (int k = p; k <= r; k++)
       if (less(aux[j], aux[i])) a[k] = aux[j--];
       else a[k] = aux[i++]; }
```

Ταξινόμηση με συγχώνευση

- Ανάλυση της πολυπλοκότητας
- N = r-p+1
- Ανάλυση πρώτα της merge:
  - 1ος βρόχος: O(N)
  - 2ος βρόχος: O(N)
  - 3ος βρόχος: O(N)
  - Συνολικά: O(N)
- Τελικά συνολικός χρόνος που απαιτείται:

$$T_N = 2T_{N/2} + O(N), \quad T_1 = O(1)$$

Ταξινόμηση με συγχώνευση

- Επίλυση της αναδρομικής εξίσωσης

  $T_N = 2T_{N/2} + N,\ T_1 = 1$
- Θα δούμε τη λύση για δυνάμεις του 2. Έστω $N = 2^n$
- Αναπτύσσουμε την εξίσωση

$$T_N = 2T_{N/2} + N = 2(2T_{N/2^2} + N/2) + N$$

$$= 2^2 T_{N/2^2} + N + N = 2^2(2T_{N/2^3} + N/2^2) + 2N$$

$$= 2^3 T_{N/2^3} + 3N = \ldots = 2^k T_{N/2^k} + kN$$

- Όταν $k = \log N$, $T_N = NT_1 + N\log N = O(N\log N)$
- Όταν το N δεν είναι δύναμη του 2, αποδεικνύεται το ίδιο με επαγωγή

Ταξινόμηση με συγχώνευση

- Αρχικά συμπεράσματα:
- O(NlogN): Πολύ καλύτερη πολυπλοκότητα από τις στοιχειώδεις μεθόδους ταξινόμησης
  - Δεν σταματάει άμεσα όμως σε ταξινομημένα αρχεία
- Δομή αναδρομικών κλήσεων
  - Η διαίρεση του αρχείου στα δύο οδηγεί σε ισοζυγισμένο δένδρο
- Πρόσθετος χώρος O(N)
  - Απαιτεί δεύτερο πίνακα
- Απαιτήσεις σε χώρο και χρόνο δεν επηρεάζονται από την αρχική διάταξη των δεδομένων
- Μπορεί να γίνει ευσταθής αν έχουμε διπλά κλειδιά

Συνθετική ταξινόμηση

- Συνθετική (bottom-up) ταξινόμηση με συγχώνευση
  - Μη αναδρομική υλοποίηση!
  - Ιδέα: διαφορετικός τρόπος διάσχισης του δέντρου διαίρει και βασίλευε
  - Η αναδρομική υλοποίηση επεξεργάζεται πρώτα το δεξιό και το αριστερό υποδέντρο και μετά πάει αμέσως στο πιο πάνω επίπεδο.
  - Οποιαδήποτε άλλη διάσχιση που διατρέχει τα υποδέντρα κάποιου κόμβου πριν επισκεφτεί τον κόμβο είναι επίσης σωστή
  - Μπορούμε π.χ. να διασχίσουμε πρώτα το κατώτερο επίπεδο του δέντρου, μετά το επόμενο επίπεδο κ.ο.κ.
  - Αρχικά κάθε στοιχείο είναι υποπίνακας
  - Σε κάθε βήμα συγχωνεύονται γειτονικοί υποπίνακες
    - Το μέγεθος των πινάκων είναι m = 1, 2, 4, 8, …
    - Πρώτα γίνονται όλες οι 1x1 συγχωνεύσεις, μετά οι 2x2, μετά οι 4x4,...

Συνθετική ταξινόμηση

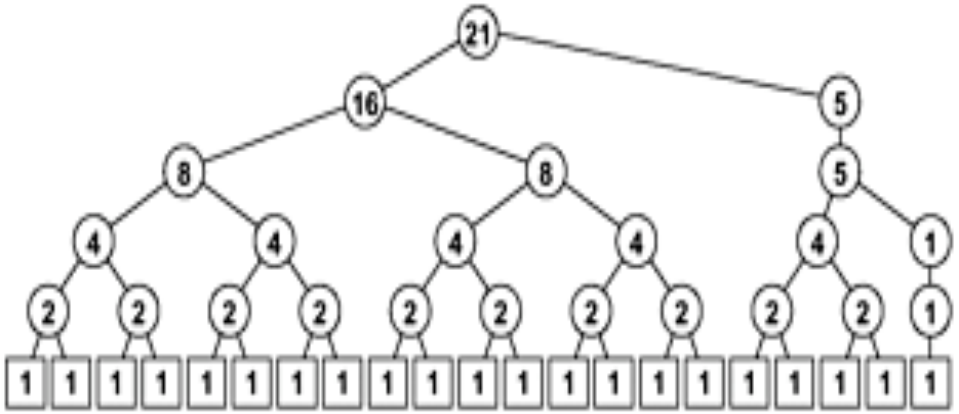
Μη αναδρομική υλοποίηση:

```
static int min(int A, int B) {return (A < B)? A: B; }
static void mergesort(ITEM[] a, int p, int r) {
     if (r <= p) return;
     aux = new ITEM[a.length];
     for (int m = 1; m <= r-p; m = m+m)/* m= 1, 2, 4,…
```

για κάθε `m` κάνουμε συγχωνεύσεις `m` επί `m` μέχρι να φτάσουμε στο δεξιό όριο `r*/`

```
       for (int i = p; i <= r-m; i += m+m)
         merge(a, i, i+m-1, min(i+m+m-1, r)); }
```

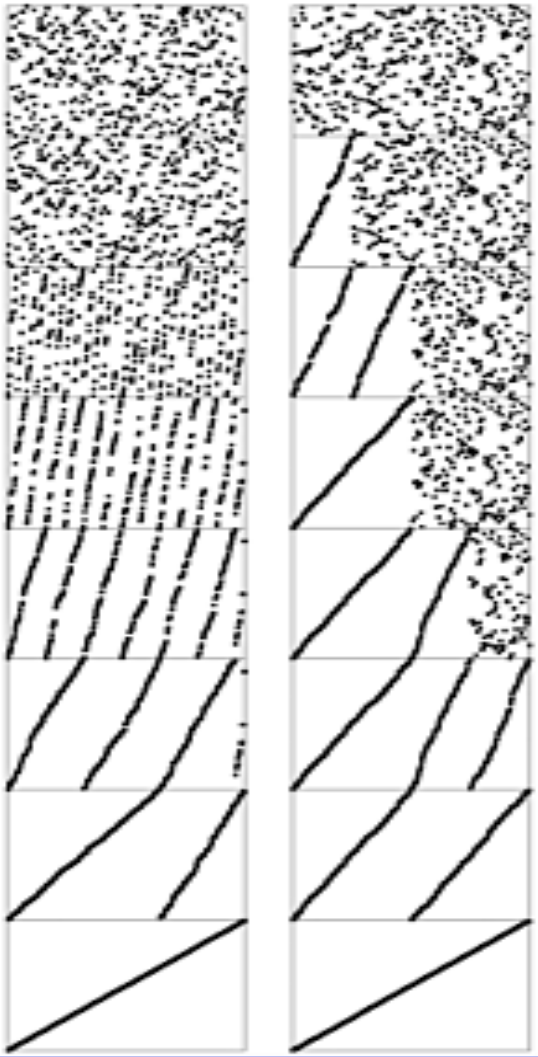
Συνθετική ταξινόμηση



- Δομή κλήσεων (όχι αναδρομικών!)
  - Όταν το N είναι δύναμη του 2 το δέντρο είναι ίδιο με τον αναδρομικό αλγόριθμο
  - Όταν το N δεν είναι δύναμη του 2 έχουμε διαφορετικό δέντρο
  - Σε κάθε επανάληψη το μέγεθος των υποπινάκων που συγχωνεύονται είναι δυνάμεις του 2 εκτός ίσως από τον τελευταίο υποπίνακα.

Σύγκριση των 2 υλοποιήσεων

- Ίδιο κόστος χρόνου χειρότερης περίπτωσης O(NlogN) και πρόσθετος χώρος O(N)
- Δεν χρειάζεται όμως στοίβα αναδρομής στη συνθετική ταξινόμηση
- Κατάλληλες και οι 2 υλοποιήσεις όταν μας ενδιαφέρει περισσότερο ο χρόνος και οχι οι απαιτήσεις μνήμης



Χαρακτηριστικά επιδόσεων

- Βελτιώσεις απόδοσης
  - Ο mergesort δεν είναι ιδιαίτερα αποδοτικός για μικρό N
  - Χρήση Insertion Sort για μικρούς υποπίνακες
    - Λύση που χρησιμοποιείται και σε άλλες ταξινομήσεις
    - Μείωση χρόνου κατά 15% περίπου
  - Εξάλειψη των διπλών αντιγραφών σε κάθε βήμα
  - Χρήση δύο πινάκων και εναλλαγή κλήσεων
    - Απλή σχετικά λύση, αποφεύγει τη μία αντιγραφή
  - Εξάλειψη του πρόσθετου πίνακα
    - Αρκετά πιο περίπλοκη λύση, κέρδος μάλλον οριακό

Χαρακτηριστικά επιδόσεων

- Α: Αναδρομικός αλγόριθμος
- Α*: Αναδρομικός με Insertion Sort για μικρά υποαρχεία
- Σ: συνθετικός αλγόριθμος
- Σ*: συνθετικός με Insertion Sort για μικρά υποαρχεία

- Αναλυτικός ή συνθετικός;
  - Ο αναλυτικός είναι λίγο πιο γρήγορος (περίπου 10%)

N	A	A*	Σ	Σ*
12500	27	16	30	20
25000	43	34	42	36
50000	91	79	92	77
100000	199	164	204	175
200000	421	352	455	396
400000	904	764	992	873
800000	1910	1629	2106	1871

Υλοποίηση με λίστες

- Χρησιμοποιώντας λίστες μπορούμε να αποφύγουμε τη χρήση του βοηθητικού πίνακα.
- Θέλουμε όμως παραπάνω μνήμη για τους δείκτες
- Αναδρομική εκδοχή: θέλουμε και μία αρχική διέλευση όλης της λίστας για να βρούμε τη μέση

```
static Node mergesort(Node c)
{ if (c == null || c.next == null) return c;
  Node a = c, b = c.next;
  while ((b != null) && (b.next != null))
      { c = c.next; b = (b.next).next; }//εύρεση μέσης
  b = c.next; c.next = null;
  return merge(mergesort(a), mergesort(b)); }
```

Υλοποίηση με λίστες

Υλοποίηση της merge:

```
static Node merge(Node a, Node b)
{ Node dummy = new Node();
  Node head = dummy, c = head;
  while ((a != null) && (b != null))
    if (less(a.item, b.item))
       { c.next = a; c = a; a = a.next; }
    else
       { c.next = b; c = b; b = b.next; }
  c.next = (a == null) ? b : a;
  return head.next;
}
```

Μειονεκτήματα της Mergesort

- Δεν σταματά άμεσα σε ταξινομημένο αρχείο
- Ο αριθμός των διελεύσεων εξαρτάται από το μέγεθος του πίνακα, και όχι από τη μορφή των δεδομένων εισόδου
- Ίδιος αριθμός βημάτων σε τυχαία δεδομένα, Gaussian, μερικώς ταξινομημένα, αντίστροφα ταξινομημένα,...