ΠΛΗ111
Δομημένος Προγραμματισμός
'Ανοιξη 2005

Μάθημα 9ο

Ταξινόμηση

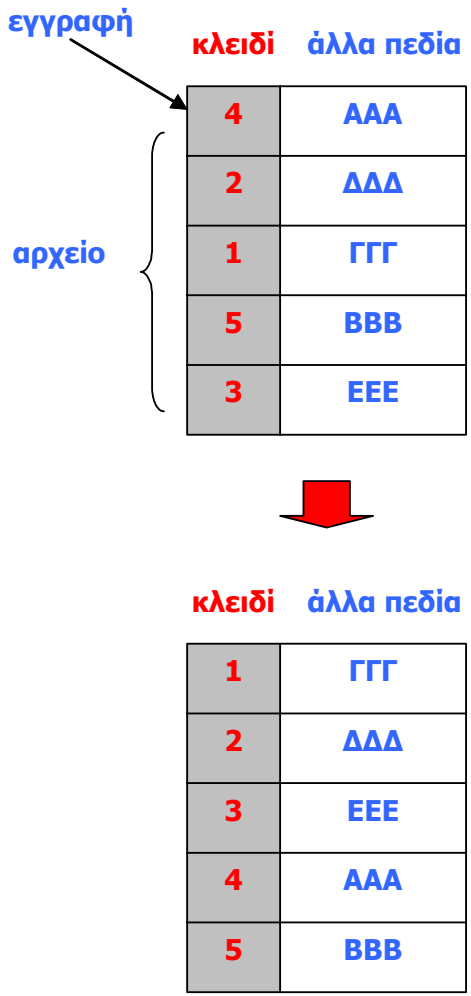
Τμήμα Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών
Πολυτεχνείο Κρήτης

Ταξινόμηση

- Εισαγωγή
- Selection sort
- Insertion sort
- Bubble sort
- Quick sort
- Merge sort

Ορισμός

- **Αρχείο** μεγέθους *n* είναι μια ακολουθία από *n* αντικείμενα *r[0], r[1], …, r[n-1]*
- Κάθε αντικείμενο *r[i]* του αρχείου ονομάζεται **εγγραφή** και έχει ένα κλειδί *k[i]*
- Το αρχείο είναι ταξινομημένο με βάση το κλειδί *k*, εφόσον το *k[i]* προηγείται του *k[j]* για κάθε *i < j*
- Παράδειγμα
  - Ο τηλεφωνικός κατάλογος είναι ένα αρχείο από εγγραφές ταξινομημένες με βάση το όνομα των συνδρομητών
- Η διαδικασία της ταξινόμησης διακρίνεται σε
  - **Εσωτερική** αν όλες οι εγγραφές βρίσκονται στην κύρια μνήμη
  - **Εξωτερική** αν κάποιες από τις εγγραφές βρίσκονται στη δευτερεύουσα μνήμη



Μέτρα Απόδοσης

- Χρόνος εκτέλεσης
  - Πλήθος συγκρίσεων στοιχείων
  - Πλήθος ανταλλαγών στοιχείων
- Μνήμη εκτέλεσης
  - Επιπλέον μνήμη μόνο για τη στοίβα (in place)
  - Επιπλέον μνήμη για αποθήκευση αντιγράφων του πίνακα

Selection Sort

- Στο ταξινομημένο αρχείο
  - Το μικρότερο κλειδί πρέπει να βρίσκεται στη θέση 0
  - Το δεύτερο μικρότερο κλειδί πρέπει να βρίσκεται στη θέση 1
  - ...
- Ταξινόμηση σε πίνακα
  - Αναζητούμε το μικρότερο κλειδί του πίνακα a[0...n-1]
  - Το ανταλλάσουμε με αυτό που βρίσκεται στην πρώτη θέση a[0]
  - Αναζητούμε το δεύτερο μικρότερο κλειδί του πίνακα a[1...n-1]
  - Το ανταλλάσουμε με αυτό που βρίσκεται στη δεύτερη θέση a[1]
  - Επαναλαμβάνουμε μέχρι να ταξινομηθεί ολόκληρο το αρχείο

Παράδειγμα Selection Sort

25 57 48 37 12 92 86 33

Λύση Selection Sort

Υλοποίηση

```
void selectionSort(int a[], int n) {
    int i, j, min, t;

    for (i = 0; i < n-1; i++) {
      /* find smallest in a[i..n-1] */
      min = i;
      for (j = i + 1; j < n; j++)
                  if (a[j] < a[min]) min = j;
      /* swap a[min] with a[i] */
      t = a[min]; a[min] = a[i]; a[i] = t;
    }
}
```

Ανάλυση Selection Sort

- Για κάθε στοιχείο *i* του *a[0..n-2]* έχουμε
  - *1* ανταλλαγή
  - *n-i-1* συγκρίσεις
- Συνολικά έχουμε
  - *n-1* ανταλλαγές
  - n(n-1)/2 συγκρίσεις

$$\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}1=\sum_{i=0}^{n-2}(n-1-i)=n-1+\ldots+1=\frac{n(n-1)}{2}$$

- Πολυπλοκότητα $O(n^2)$

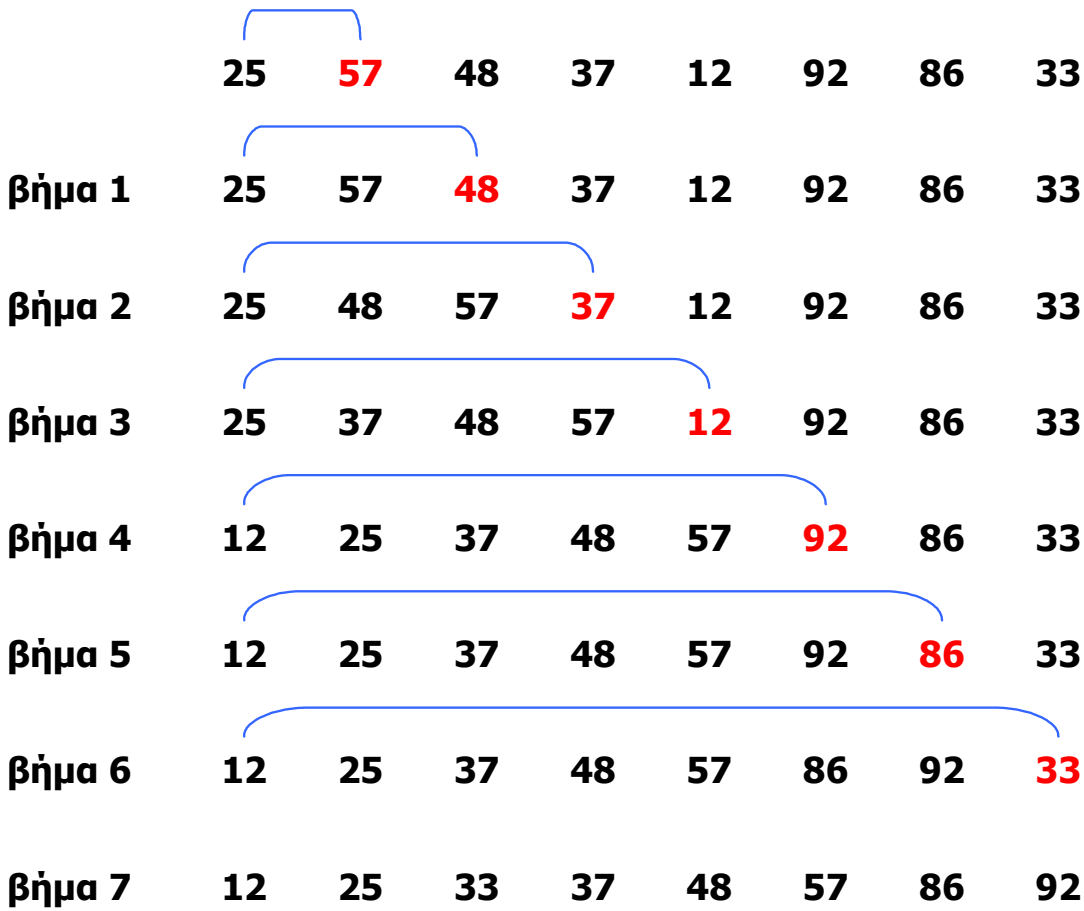
Insertion Sort

- Θεωρούμε στοιχεία αποθηκευμένα σε πίνακα
- Αρχικά όλα τα στοιχεία είναι μη ταξινομημένα
- Ταξινομούμε τα στοιχεία ένα κάθε φορά
  - Παίρνουμε ένα στοιχείο από αυτά που δεν έχουν ταξινομηθεί
  - Βρίσκουμε την κατάλληλη θέση του μεταξύ των ταξινομημένων
  - Μετακινούμε τα μεγαλύτερα στοιχεία του μια θέση δεξιά
  - Εισάγουμε το στοιχείο στη θέση που απελευθερώθηκε
- Επαναλαμβάνουμε μέχρι να ταξινομήσουμε όλα τα στοιχεία

Παράδειγμα Insertion Sort

25 57 48 37 12 92 86 33

Λύση Insertion Sort



Υλοποίηση

```
void insertionSort(int a[], int n) {
        int i, j, temp;

        for (i = 1; i < n; i++) {
                temp = a[i];
                /* a[0..i-1] > temp move to the right */
                for (j = i; j > 0 && a[j-1] > temp; j--)
                        a[j] = a[j-1];
                /* temp moves to vacant slot a[j] */
                a[j] = temp;
        }
}
```

Ανάλυση Insertion Sort

- Χειρότερη περίπτωση
  - Αντίστροφα ταξινομημένος πίνακας
  - Κάθε στοιχείο μετακινείται στο αριστερό άκρο του πίνακα
  - Συνολικό πλήθος συγκρίσεων και μετακινήσεων

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

- Καλύτερη περίπτωση
  - Ταξινομημένος πίνακας
  - Κανένα στοιχείο δε μετακινείται
  - Συνολικό πλήθος συγκρίσεων

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

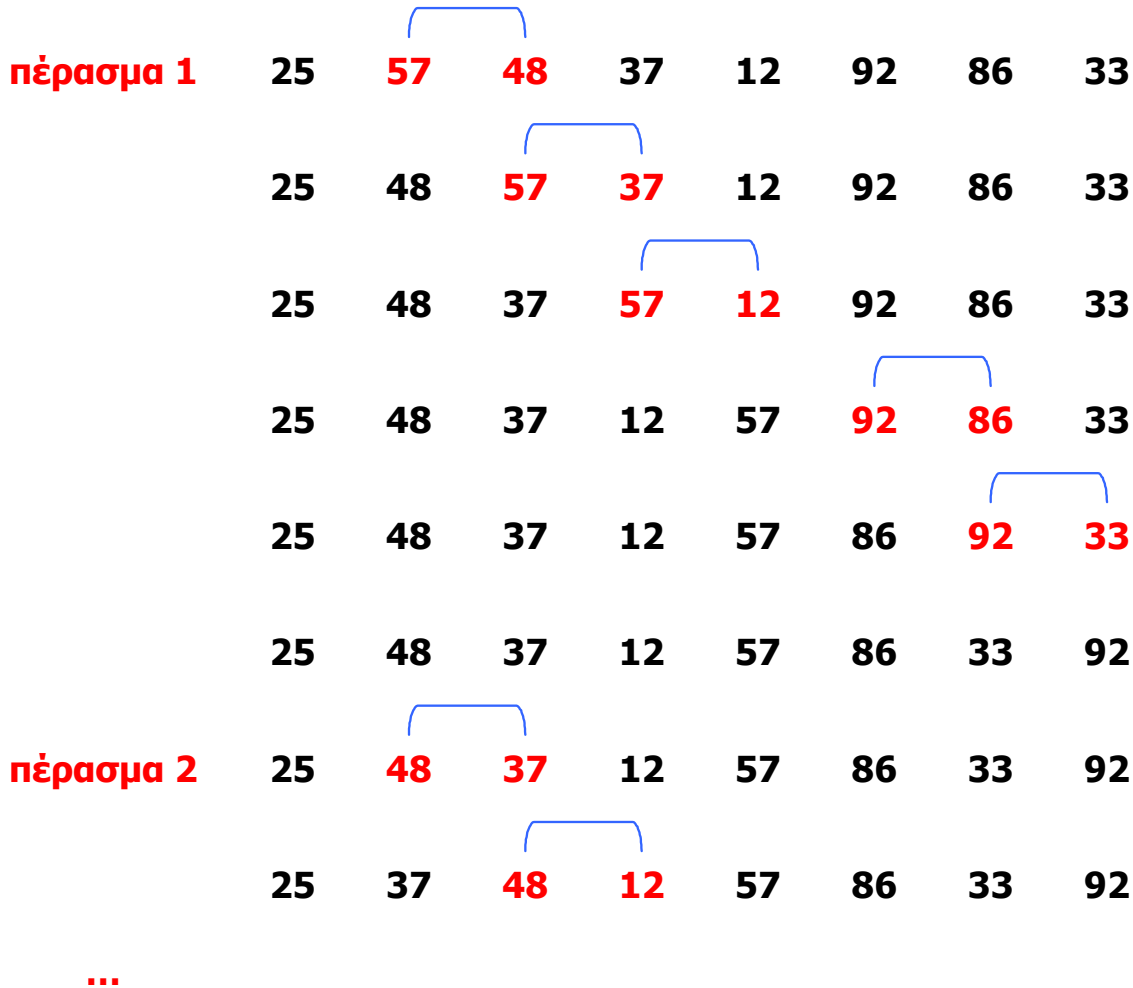
Bubble Sort

- Περνάμε από τα στοιχεία του αρχείου πολλαπλές φορές
- Σε κάθε πέρασμα
  - Συγκρίνουμε το κλειδί κάθε στοιχείου με το επόμενό του
  - Εάν τα στοιχεία δεν είναι στη σωστή διάταξη τα ανταλλάσουμε
- Μετά το i-στό πέρασμα
  - Το στοιχείο a[n-i] βρίσκεται στη σωστή θέση
  - Δε χρειάζεται να εξετάσουμε ξανά τα στοιχεία a[n-i..n-1]
- Όταν δε γίνονται ανταλλαγές στοιχείων σε ένα πέρασμα
  - Τα στοιχεία είναι πλήρως ταξινομημένα
  - Ο αλγόριθμος τερματίζει

Παράδειγμα Bubble Sort

πέρασμα 1	**25**	**57**	**48**	**37**	**12**	**92**	**86**	**33**

Λύση Bubble Sort



Υλοποίηση

```
void bubbleSort(int a[], int n) {
    int i,j,t;
    int sorted = 0;

    for (i = n-1; i>= 0 && !sorted; i--) {
        sorted = 1;
        for (j = 1; j <= i; j++)
            if (a[j-1] > a[j]) {        /* swap */
                t = a[j-1]; a[j-1] = a[j]; a[j] = t;
                sorted = 0;
            }
    }
}
```

Ανάλυση Bubble Sort

- Χειρότερη περίπτωση
  - Αντίστροφα ταξινομημένος πίνακας
  - Στο πέρασμα i χρειάζονται n-i συγκρίσεις και ανταλλαγές
  - Συνολικό πλήθος συγκρίσεων και ανταλλαγών

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

- Καλύτερη περίπτωση
  - Ταξινομημένος πίνακας
  - Κανένα στοιχείο δε μετακινείται
  - Συνολικό πλήθος συγκρίσεων

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

Quick Sort

- "Divide and Conquer" (διαίρει και βασίλευε)
  - Διαίρεσε το αρχείο σε δύο μέρη
  - Ταξινόμησε το κάθε μέρος ανεξάρτητα

```
quicksort(int a[], int l, int r)
        if (r>l) { int i = partition(a, l, r);      /* διαίρεση */
                quicksort(a, l, i-1);
                quicksort(a, i+1, r);
        }
```

- Η βασική ιδέα του αλγόριθμου βρίσκεται στη διαδικασία διαίρεσης
  - Ένα στοιχείο a[i] τοποθετείται στην τελική θέση ταξινόμησης
  - Όλα τα στοιχεία a[0..i-1] είναι μικρότερα ή ίσα του a[i]
  - Όλα τα στοιχεία a[i+1..n-1] είναι μεγαλύτερα ή ίσα του a[i]

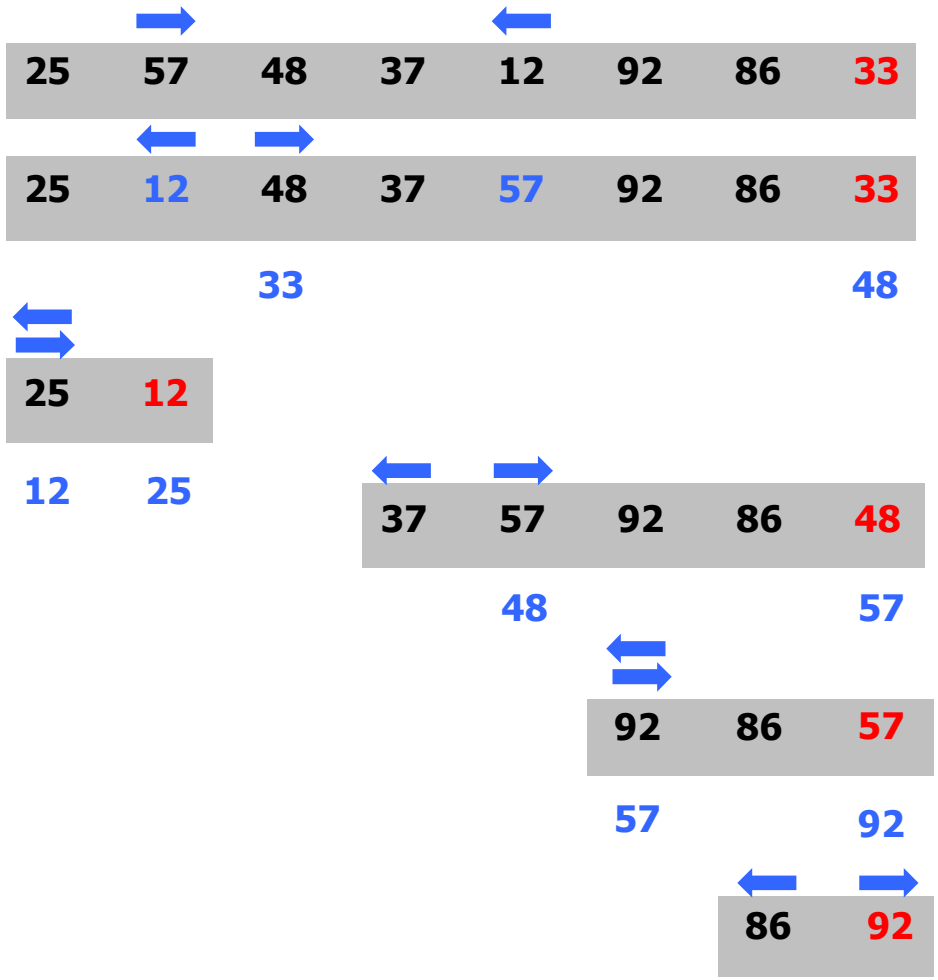
Quick Sort

- Υλοποίηση διαίρεσης
  - Επιλέγουμε οποιοδήποτε στοιχείο a[i] για τελική τοποθέτηση
  - Ξεκινώντας από το a[0] αναζητούμε στοιχείο > a[i]
  - Ξεκινώντας από το a[n-1] αναζητούμε στοιχείο < a[i]
  - Ανταλλάσουμε τα δύο στοιχεία που βρήκαμε παραπάνω
  - Συνεχίζουμε τη διαδικασία αναζήτησης και ανταλλαγής
  - Όταν οι δύο αναζητήσεις συναντηθούν, ανταλλάσουμε το a[i] με το δεξιότερο στοιχείο του αριστερού υποπίνακα

Παράδειγμα Quick Sort

25	57	48	37	12	92	86	33

Λύση Quick Sort



Υλοποίηση

```
void quickSort(int a[], int l, int r) {
    int v, i, j, t;
    if (r > l) {
        /* partitioning */
        v = a[r]; i = l-1; j = r;       /* pivot a[r] */
        for (;;) {
            while (a[++i] < v);
            while (a[--j] > v && j > l);
            if (i >= j) break;
            t = a[i]; a[i] = a[j]; a[j] = t;          /* swap */
        }
        t = a[i]; a[i] = a[r]; a[r] = t;              /* swap */
        quickSort(a, l, i-1);
        quickSort(a, i+1, r);
    }
}
```

Ανάλυση Quick Sort

- Εάν υποδιαιρούμε τον πίνακα σε ισομεγέθη μέρη κάθε φορά, τότε έχουμε χρόνο εκτέλεσης

  $T(1) = 1,$

  $T(n) = n+2T(n/2), n>1$

  οπότε

  $T(n) = n + 2\ n/2 + 4\ n/4 + \ldots + n\ n/n = O(n \log_2 n)$

- Εάν επιλέγουμε το στοιχείο υποδιαίρεσης (pivot) έτσι ώστε ο ένας υποπίνακας να είναι πάντα άδειος, έχουμε

  $T(1) = 1,$

  $T(n) = n+T(n-1), n>1$

  οπότε

  $T(n) = n + n-1 + \ldots + 1 = O(n^2)$

Merge Sort

- Θεωρούμε διαδικασία συγχώνευσης δύο πινάκων a[1..m] και b[1..n]

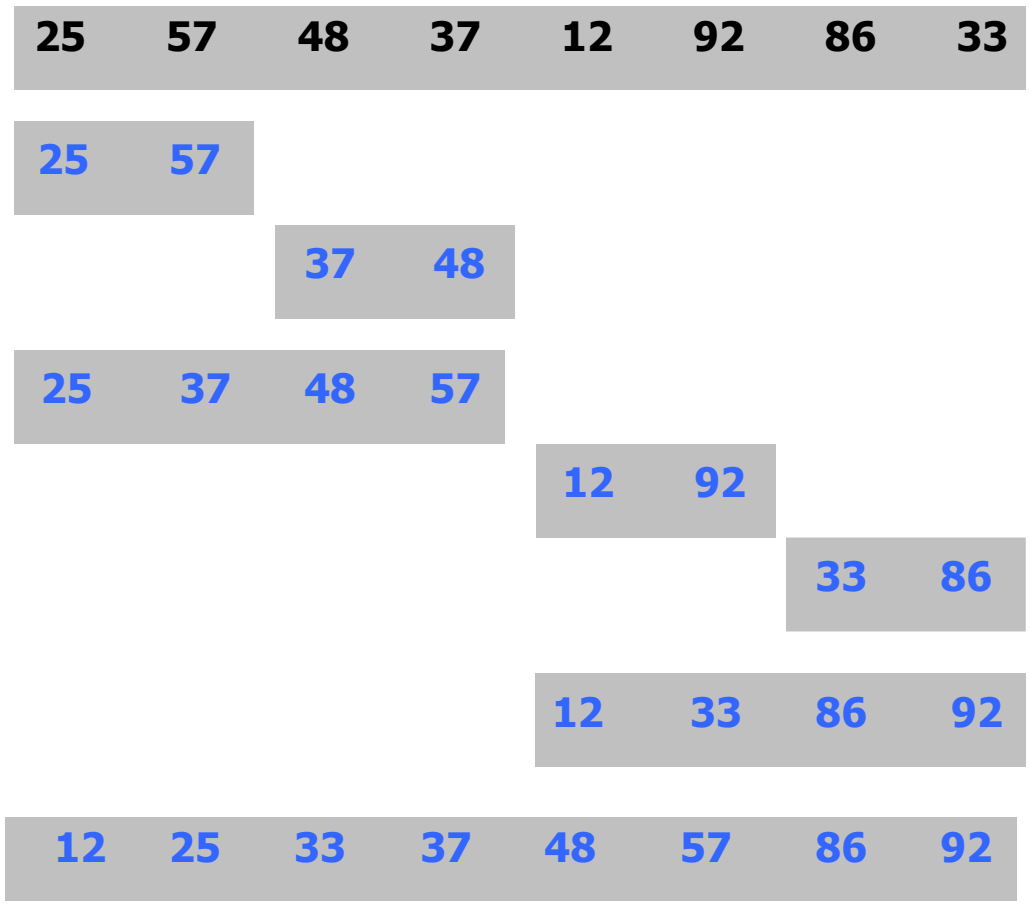
```
i = 1; j = 1;
a[m+1] = INT_MAX; b[n+1] = INT_MAX;
for (k = 1; k <= m+n; k++)
        c[k] = (a[i] < b[j]) ? a[i++] : b[j++];
```

- Για να ταξινομήσουμε έναν πίνακα
  - Διαιρούμε τον πίνακα σε δύο ίσα μέρη ("divide and conquer")
  - Ταξινομούμε αναδρομικά κάθε μέρος του πίνακα
  - Συγχωνεύουμε τα δύο μέρη

Παράδειγμα Merge Sort

25	57	48	37	12	92	86	33

Λύση Merge Sort



Υλοποίηση

```
void mergeSort(int a[], int l, int r) {
      int i, j, k, m;
      if (r > l) {
            m = (r+l)/2;
            mergeSort(a, l, m);
            mergeSort(a, m+1, r);
            /* αντιγραφή του a[l..m] στο b[l..m] */
            for (i = m; i >= l; i--) b[i] = a[i];
            /* αντιστροφή του a[m+1..r] στο b[m+1..r] */
            for (j = m; j < r; j++)   b[r+m-j] = a[j+1];
             /* συγχώνευση b[l..m] και b[m+1..r] στο a[l..r]*/
            for (k = l; k <= r; k++)
                        a[k] = (b[i] < b[j]) ? b[i++] : b[j--];
      }
}
```

Ανάλυση Merge Sort

- Από την αναδρομή έχουμε πλήθος συγκρίσεων

  $T(1) = 0$

  $T(n) = n + 2\,T(n/2),\ n>1$

  οπότε

  $T(n) = n + 2\,n/2 + 4\,n/4 + \ldots + n\,n/n = O(n \log_2 n)$

- Επίσης λόγω του βοηθητικού πίνακα χρησιμοποιούμε επιπλέον μνήμη O(n)