Προγραμματισμός Ι (ECE115)

#15 αναζήτηση και ταξινόμηση

Σπύρος Λάλης
lalis@uth.gr

Αναζήτηση σε μη ταξινομημένο πίνακα

- Δεν μπορεί να γίνει κάτι «έξυπνο»

- **Γραμμική** και **διεξοδική** αναζήτηση
  - linear exhaustive search
- Ελέγχουμε τα στοιχεία του πίνακα, από την αρχή προς το τέλος, μέχρι να βρούμε αυτό που θέλουμε

- Για να βρούμε αυτό που ψάχνουμε, χρειάζονται κατά μέσο όρο **N/2** βήματα
- Αν αυτό που ψάχνουμε **δεν** υπάρχει, κάνουμε **πάντα** ακριβώς **N** βήματα

```
/* γραμμική/διεξοδική αναζήτηση σε πίνακα ακεραίων */

int linear_search(int a[], int len, int key) {
  int i;

  for(i = 0; (i < len) && (a[i] != key); i++);

  return(i);
}
```

Ταξινομημένος πίνακας

- Η γραμμική αναζήτηση μπορεί να **επιταχυνθεί** αν τα στοιχεία του πίνακα είναι ταξινομημένα
  - π.χ. με αύξουσα σειρά

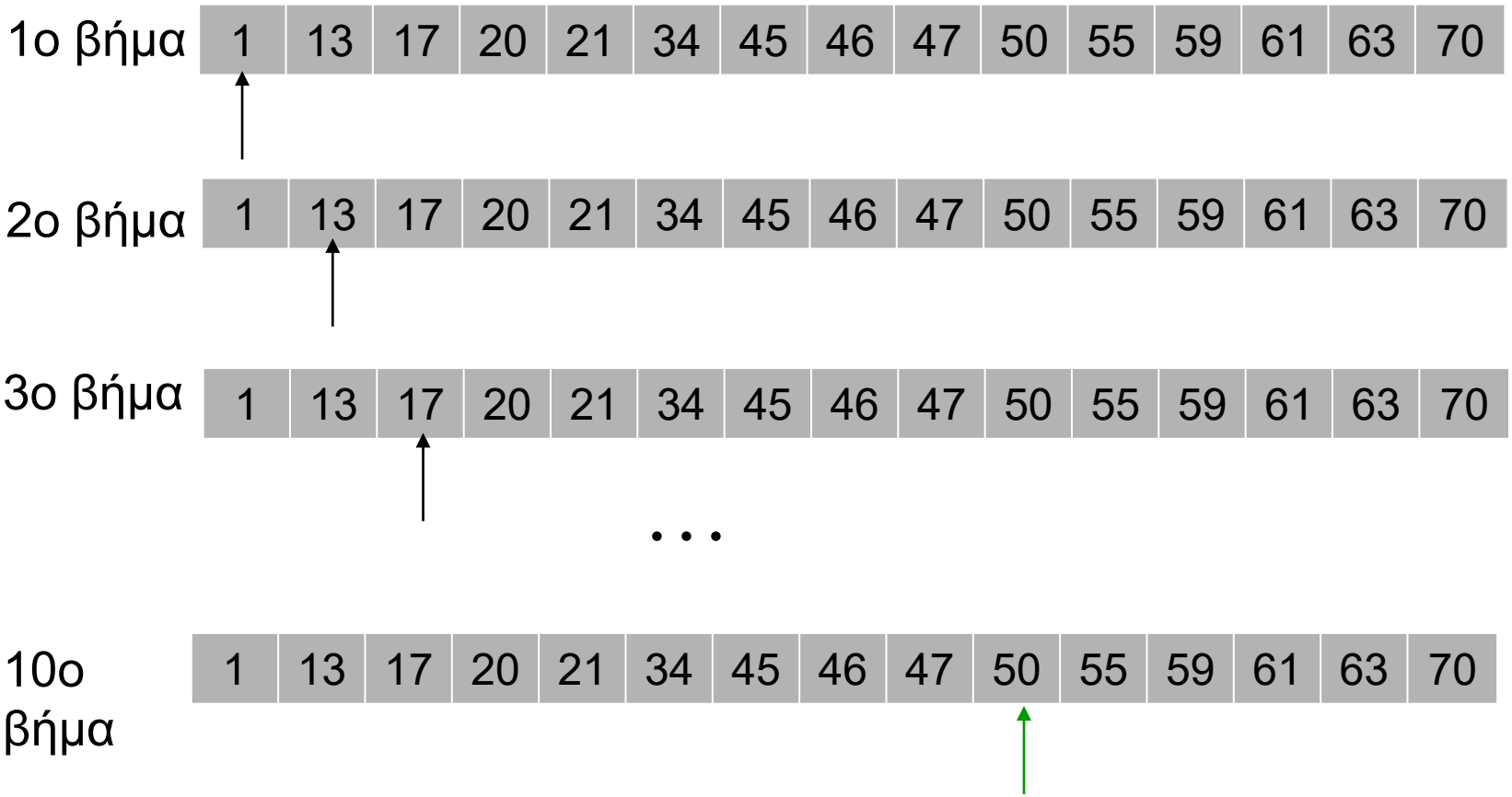
- Δεν χρειάζεται να ψάξουμε μέχρι το τέλος για να διαπιστώσουμε ότι αυτό που θέλουμε δεν υπάρχει
- Αρκεί να φτάσουμε σε ένα στοιχείο που είναι μεγαλύτερο (ή ίσο) από αυτό που αναζητούμε

- Για να βρούμε αυτό που ψάχνουμε, **πάλι** χρειάζονται κατά μέσο όρο **N/2** βήματα
- Αυτό (πλέον) ισχύει **και** στην περίπτωση που **δεν** υπάρχει αυτό που ψάχνουμε

```
/* γραμμική/διεξοδική αναζήτηση σε πίνακα ακεραίων */

int linear_search(int a[], int len, int key) {
  int i;

  for(i = 0; (i < len) && (a[i] < key); i++);

  return(i);
}
```

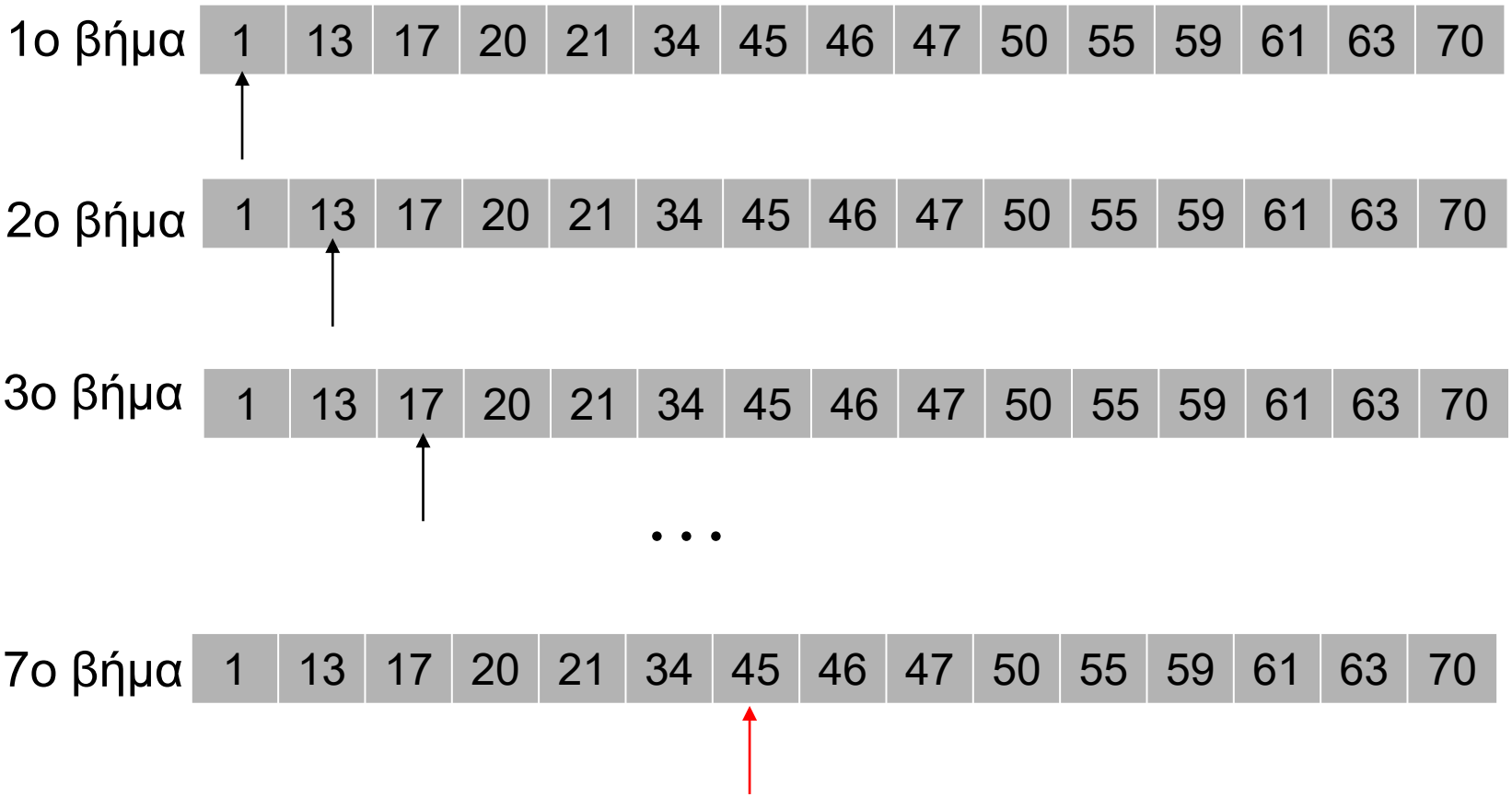
Γραμμική αναζήτηση

βρες τη θέση που υπάρχει (αν υπάρχει) η τιμή 50



Γραμμική αναζήτηση

βρες τη θέση που υπάρχει (αν υπάρχει) η τιμή 37



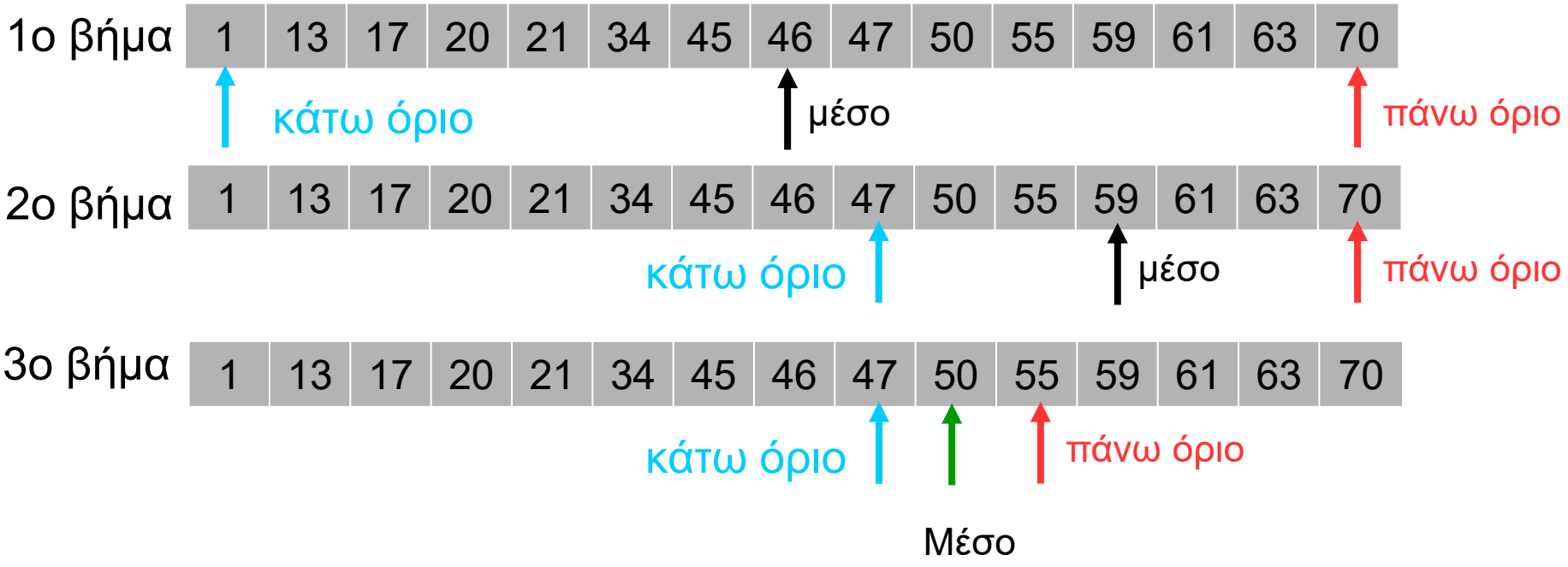
Δυαδική αναζήτηση (ταξινομημένος πίνακας)

- Μπορούμε να κάνουμε κάτι πιο **έξυπνο** από μια απλή γραμμική αναζήτηση
- **Δυαδική** αναζήτηση
  - binary search
- Ελέγχουμε το στοιχείο στο μέσο του πίνακα
- Αν δεν είναι αυτό που ψάχνουμε, επαναλαμβάνουμε την διαδικασία στο **μισό** τμήμα του πίνακα
  - το τμήμα όπου θα συνεχιστεί η διαδικασία της αναζήτησης επιλέγεται με βάση με την τιμή του στοιχείου που κοιτάμε

- Κατά μέσο όρο χρειάζονται $\mathbf{\log_2 N}$ βήματα
- Ακόμα και όταν **δεν** υπάρχει αυτό που ψάχνουμε

Δυαδική αναζήτηση

βρες τη θέση που υπάρχει (αν υπάρχει) η τιμή 50



```
/* δυαδική αναζήτηση */

int binary_search(int a[], int from, int to, int key) {
  int m;

  while (from <= to) {
    m = (to+from)/2;
    if (a[m] == key) {
      return(m);
    }
    else if (a[m] > key) {
      to = m-1;
    }
    else {
      from = m+1;
    }
  }
  return (-1);
}
```

Ταξινόμηση

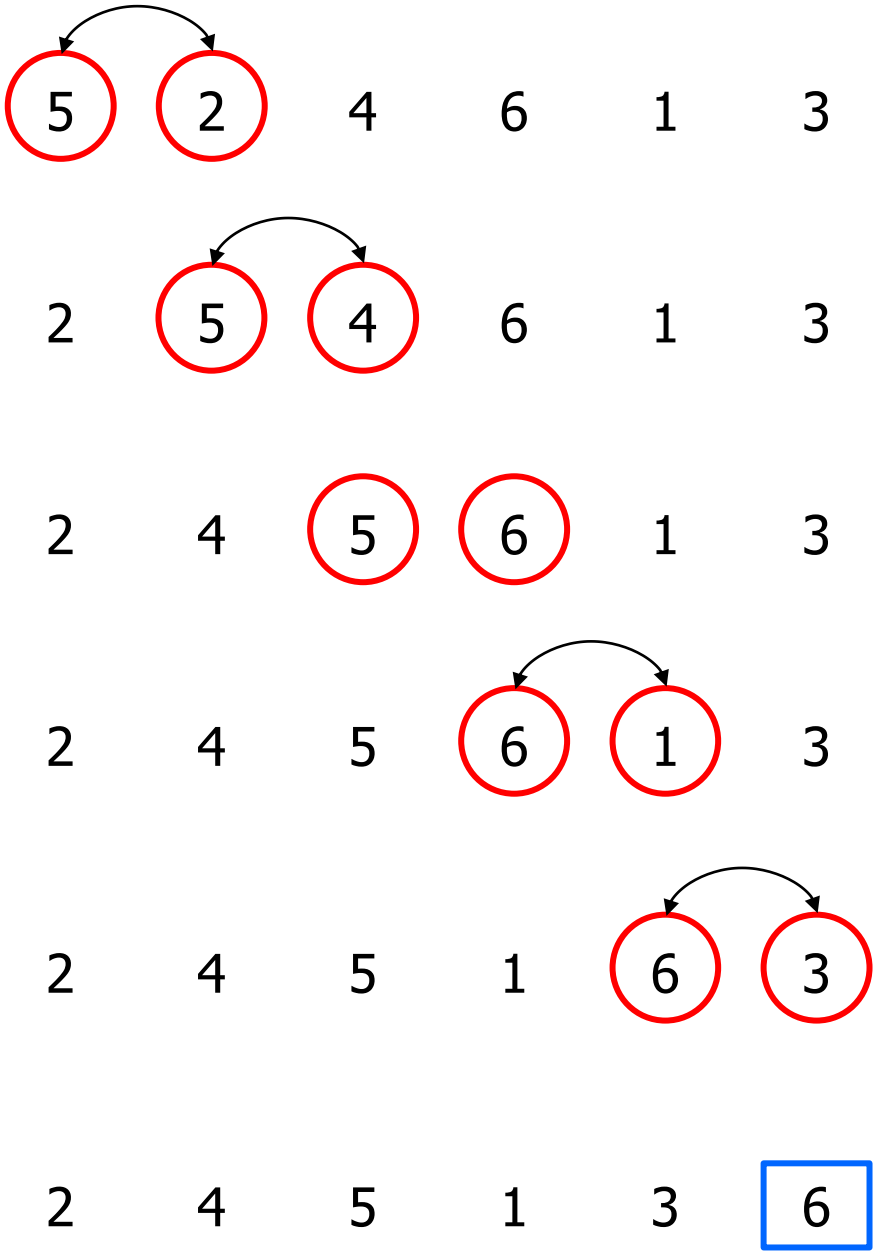
- Ένας πίνακας μπορεί να παραμένει ταξινομημένος, με κατάλληλη υλοποίηση των λειτουργιών διαχείρισης
  - προσθήκη τιμής `v`: μετακίνηση 1 θέση προς τα μπρος όλων των τιμών που είναι μεγαλύτερες του `v`
  - αφαίρεση τιμής `v`: μετακίνηση 1 θέση προς τα πίσω όλων των τιμών που είναι μεγαλύτερες του `v`

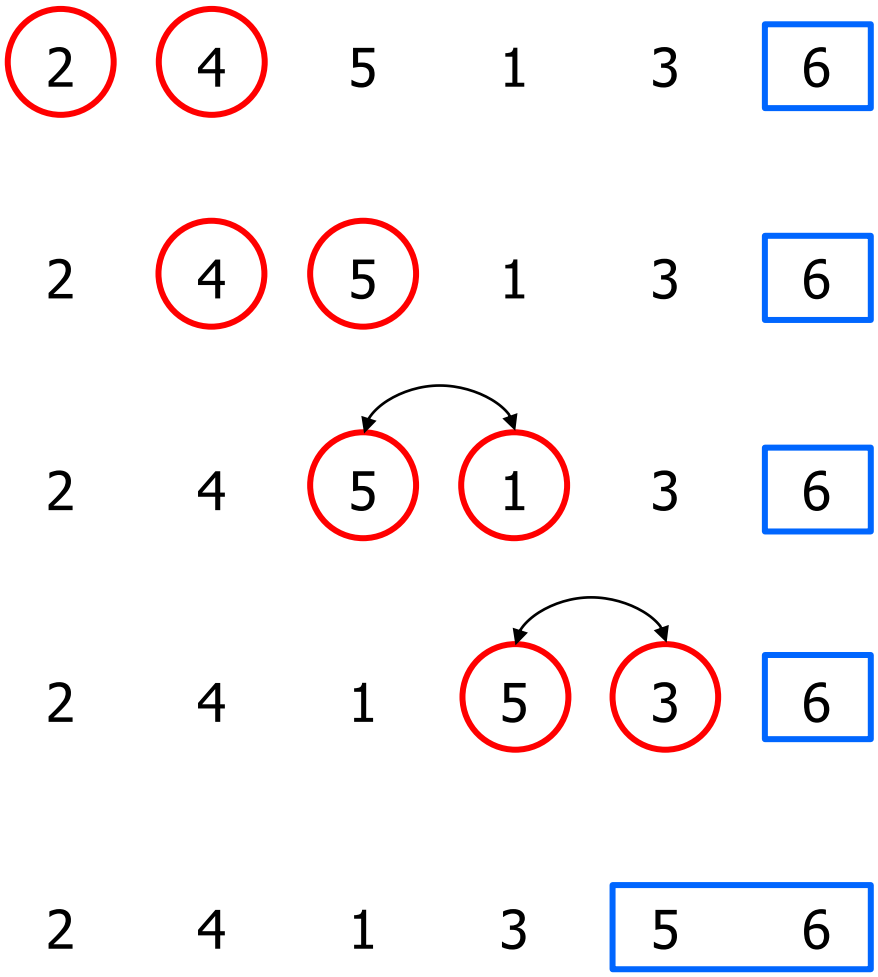
- Εναλλακτικά, μπορεί να γίνει **ταξινόμηση** των στοιχείων ενός πίνακα σε δεύτερο χρόνο
  - υπάρχουν πολλοί διαφορετικοί αλγόριθμοι για αυτό

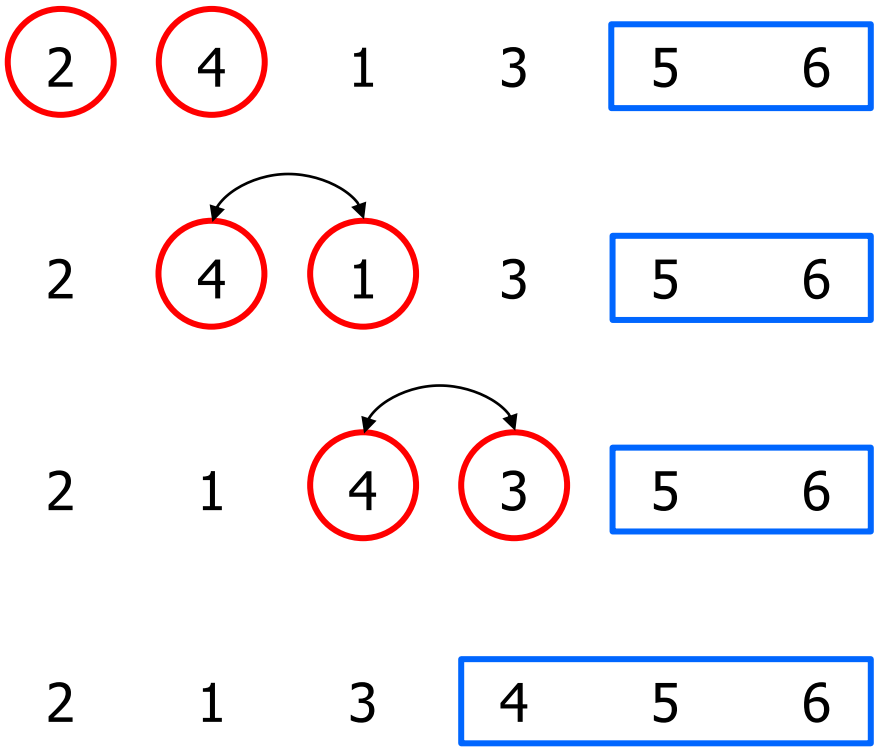
Bubble sort

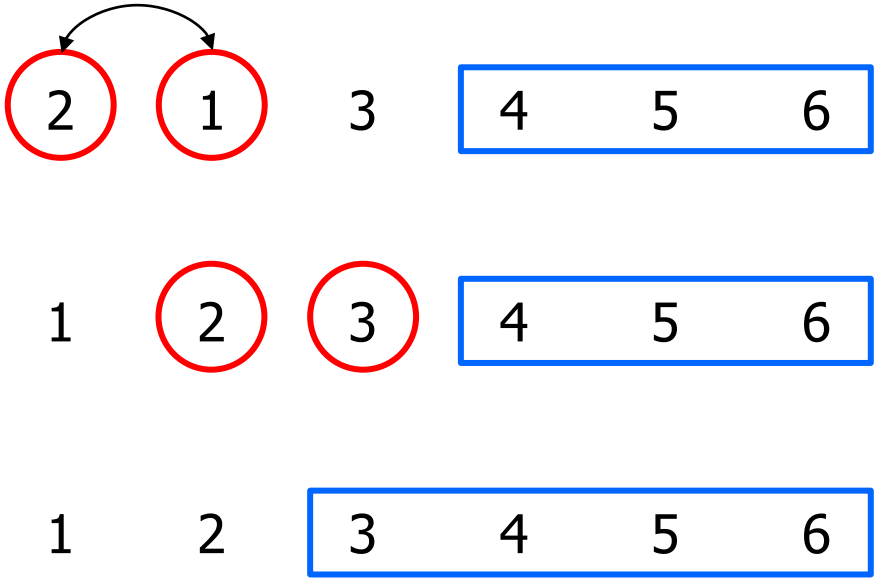
- Αρχίζοντας από την αρχή προς το τέλος του πίνακα, συγκρίνουμε τα γειτονικά στοιχεία ανά ζεύγη
- Αν δύο γείτονες έχουν λάθος σειρά, γίνεται **ανταλλαγή** τιμών (swap)
- Αφού ελέγξουμε και το τελευταίο ζευγάρι, είναι σίγουρο ότι στο τελευταίο στοιχείο του πίνακα βρίσκεται η μεγαλύτερη τιμή
  - ανέβηκε «προς τα πάνω» σαν μια «φουσκάλα»

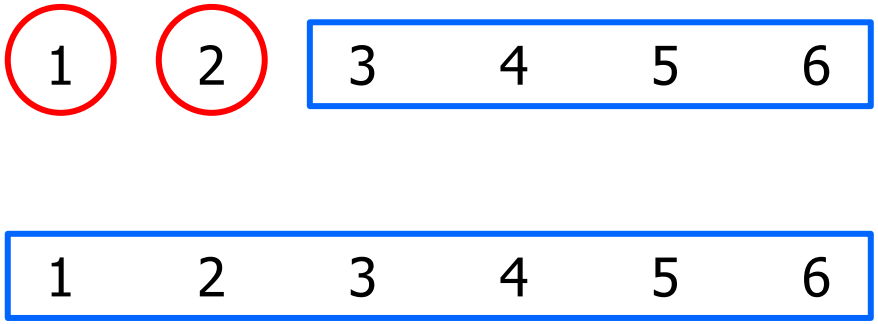
- Επαναλαμβάνουμε την ίδια διαδικασία, για το υπόλοιπο (αταξινόμητο) τμήμα του πίνακα
  - μέχρι να μην απομείνει υπόλοιπο (αταξινόμητο) τμήμα


5 2 4 6 1 3
2 5 4 6 1 3
2 4 5 6 1 3
2 4 5 6 1 3
2 4 5 1 6 3
2 4 5 1 3 6


2 4 5 1 3 6
2 4 5 1 3 6
2 4 5 1 3 6
2 4 1 5 3 6
2 4 1 3 5 6


2 4 1 3 5 6
2 4 1 3 5 6
2 1 4 3 5 6
2 1 3 4 5 6


2 1 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6

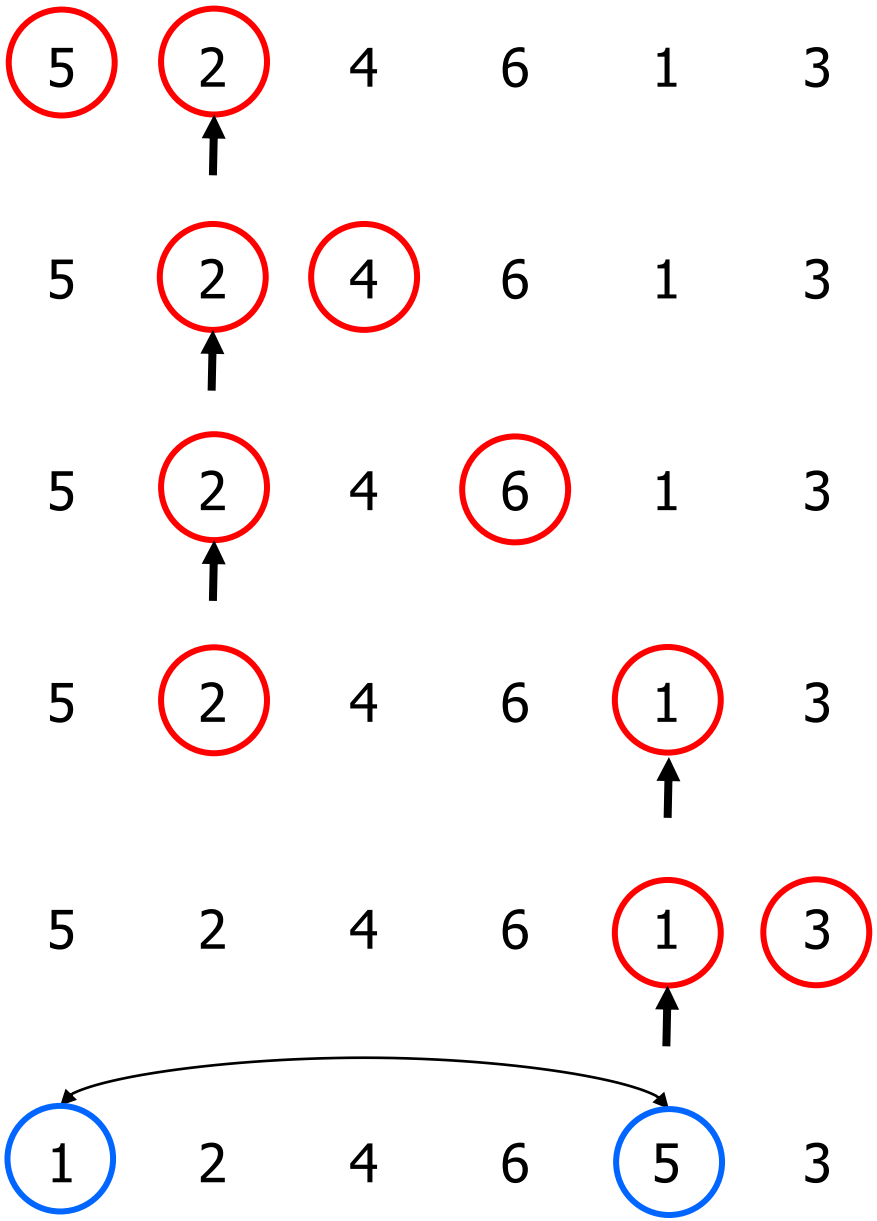

1
2
3
4
5
6
1
2
3
4
5
6

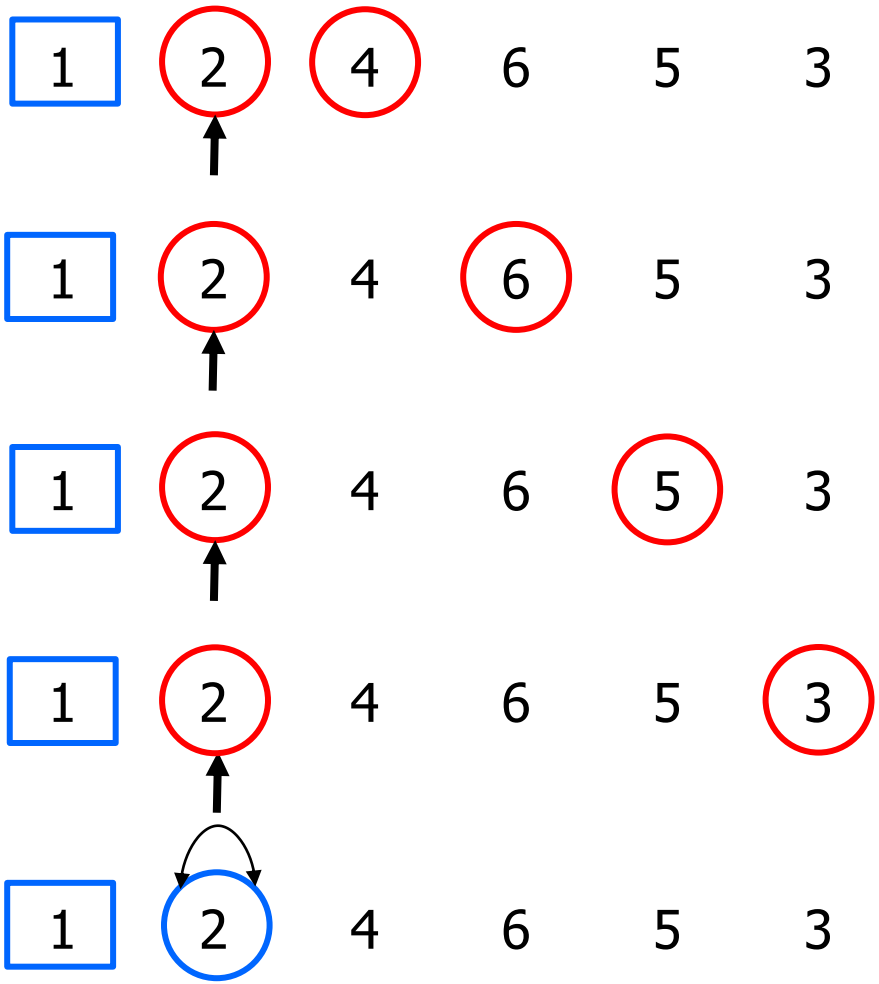
```
/* bubble sort */

void bubble_sort(int a[], int len) {
  int i,j,tmp;

  for (i = 1; i < len; i++) {
    for (j = 0; j < len-i; j++) {
      if (a[j] > a[j+1]) {
        tmp = a[j];
        a[j] = a[j+1];
        a[j+1] = tmp;
      }
    }
  }
}
```

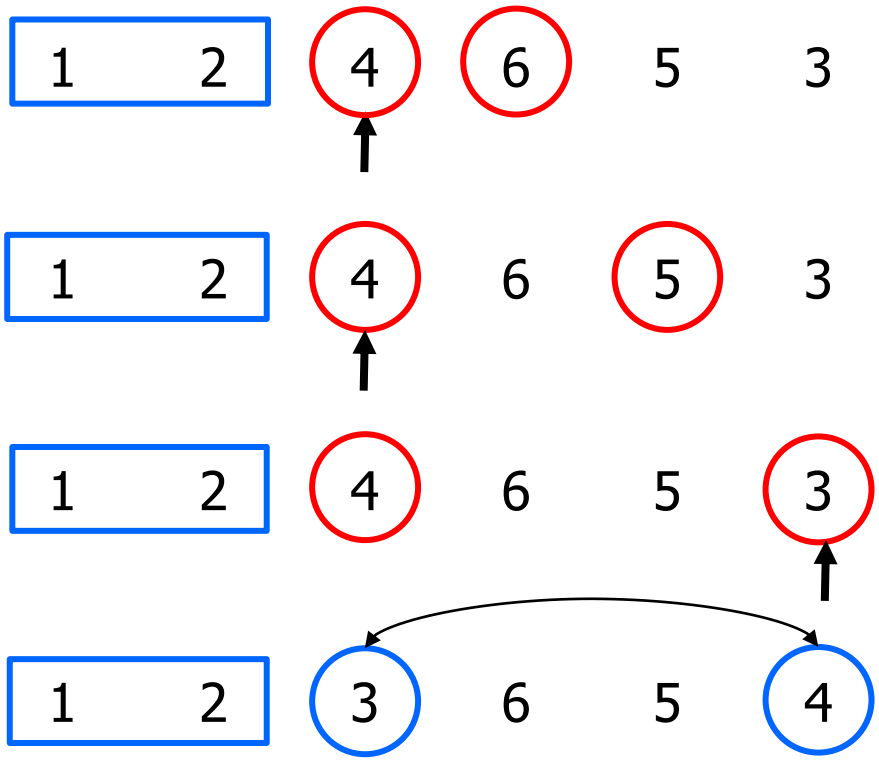
Selection sort

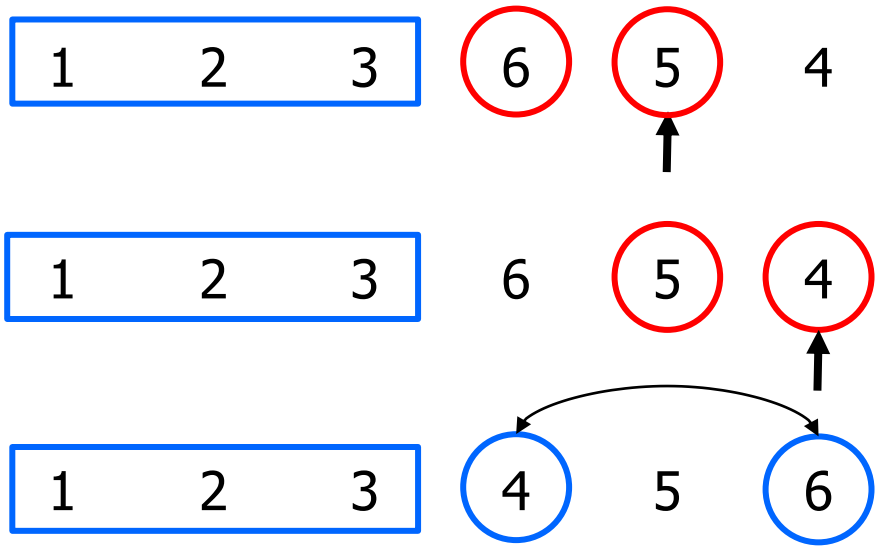
- Ψάχνουμε από την αρχή μέχρι το τέλος του πίνακα για να βρούμε μικρότερο στοιχείο
- Ανταλλάσσουμε το μικρότερο στοιχείο που βρήκαμε με το 1ο στοιχείο του πίνακα
  - γίνεται **μια** μοναδική ανταλλαγή στοιχείων (αντίθετα με το bubble sort)

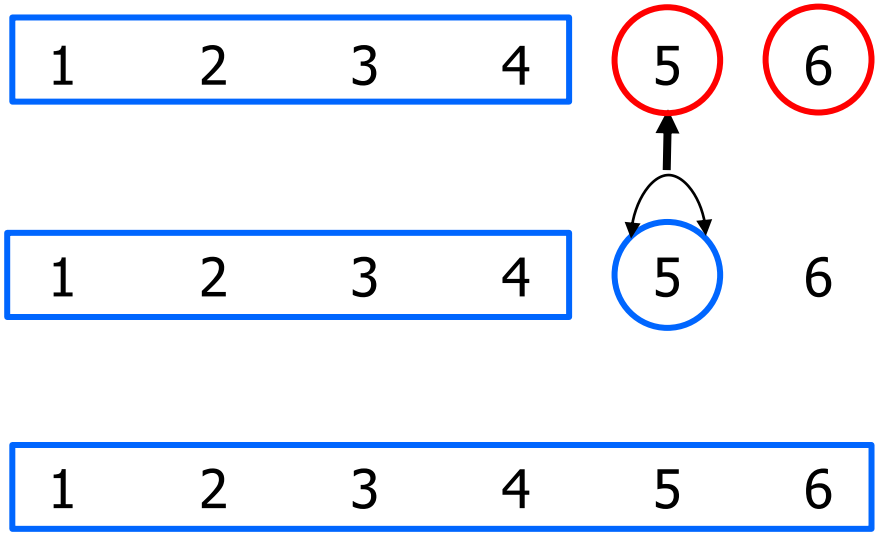
- Επαναλαμβάνουμε την διαδικασία, για το υπόλοιπο (αταξινόμητο) τμήμα του πίνακα
  - μέχρι να μην απομείνει υπόλοιπο (αταξινόμητο) τμήμα


5 2 4 6 1 3
5 2 4 6 1 3
5 2 4 6 1 3
5 2 4 6 1 3
5 2 4 6 1 3
1 2 4 6 5 3


1 2 4 6 5 3
1 2 4 6 5 3
1 2 4 6 5 3
1 2 4 6 5 3
1 2 4 6 5 3


1 2 4 6 5 3
1 2 4 6 5 3
1 2 4 6 5 3
1 2 3 6 5 4


1 2 3
6 5 4
1 2 3
6 5 4
1 2 3
4 5 6


1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6

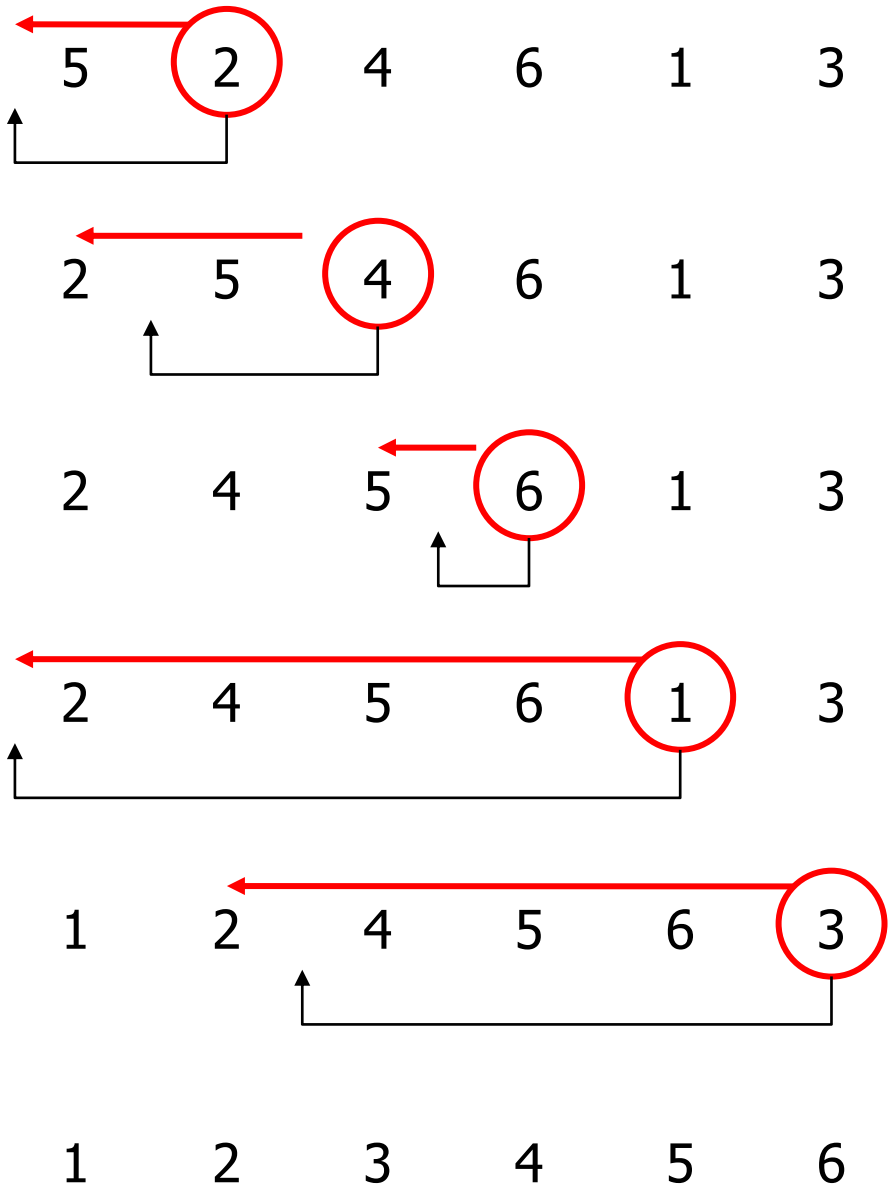
```
/* selection sort */

void selection_sort(int a[], int len) {
  int i,j,min,tmp;

  for (i = 0; i < len; i++) {
    min = i;
    for (j = i+1; j < len; j++) {
      if (a[j] < a[min]) {
        min = j;
      }
    }
    tmp = a[i];
    a[i] = a[min];
    a[min] = tmp;
  }
}
```

Insertion sort

- Εξέτασε τα στοιχεία από τη θέση 1 έως και τη θέση Ν-1 του πίνακα (αν ο πίνακας έχει Ν θέσεις)
  - Για κάθε στοιχείο που εξετάζεις βρες τη «σωστή» θέση στην οποία πρέπει να τοποθετηθεί μεταξύ των στοιχείων που βρίσκονται αριστερά του.

- Για κάθε στοιχείο, έλεγξε αν είναι μικρότερο από το προηγούμενό του
  - Αν είναι, μετακίνησε το προηγούμενο στοιχείο μία θέση δεξιά για να κάνεις «χώρο»
  - Επανέλαβε τον ίδιο έλεγχο και με τα προηγούμενα. Εφόσον το στοιχείο είναι μικρότερο από το προηγούμενο, μετακίνησε παρόμοια το προηγούμενο μία θέση δεξιά.
- Μόλις βρεθεί η «σωστή» θέση, τοποθέτησε το στοιχείο (έχει δημιουργηθεί χώρος από τις μετακινήσεις)

- Μετά το τέλος κάθε βήματος, το κομμάτι του πίνακα αριστερά από το στοιχείο που εξετάστηκε είναι ταξινομημένο.
- Επαναλαμβάνουμε την διαδικασία, για το υπόλοιπο (αταξινόμητο) τμήμα του πίνακα
  - μέχρι να μην απομείνει υπόλοιπο (αταξινόμητο) τμήμα


5 2 4 6 1 3
2 5 4 6 1 3
2 4 5 6 1 3
2 4 5 6 1 3
1 2 4 5 6 3
1 2 3 4 5 6

```
/* insertion sort */

void insertion_sort(int a[], int len) {
  int i,j,tmp;

  for (i=1; i<len; i++) {
    tmp = a[i];
    j = i-1;
    while ((j >= 0) && (a[j] > tmp)) {
      a[j+1] = a[j];
      j = j-1;
    }
    a[j+1] = tmp;
  }
}
```

Αποφυγή αντιγραφής εγγραφών

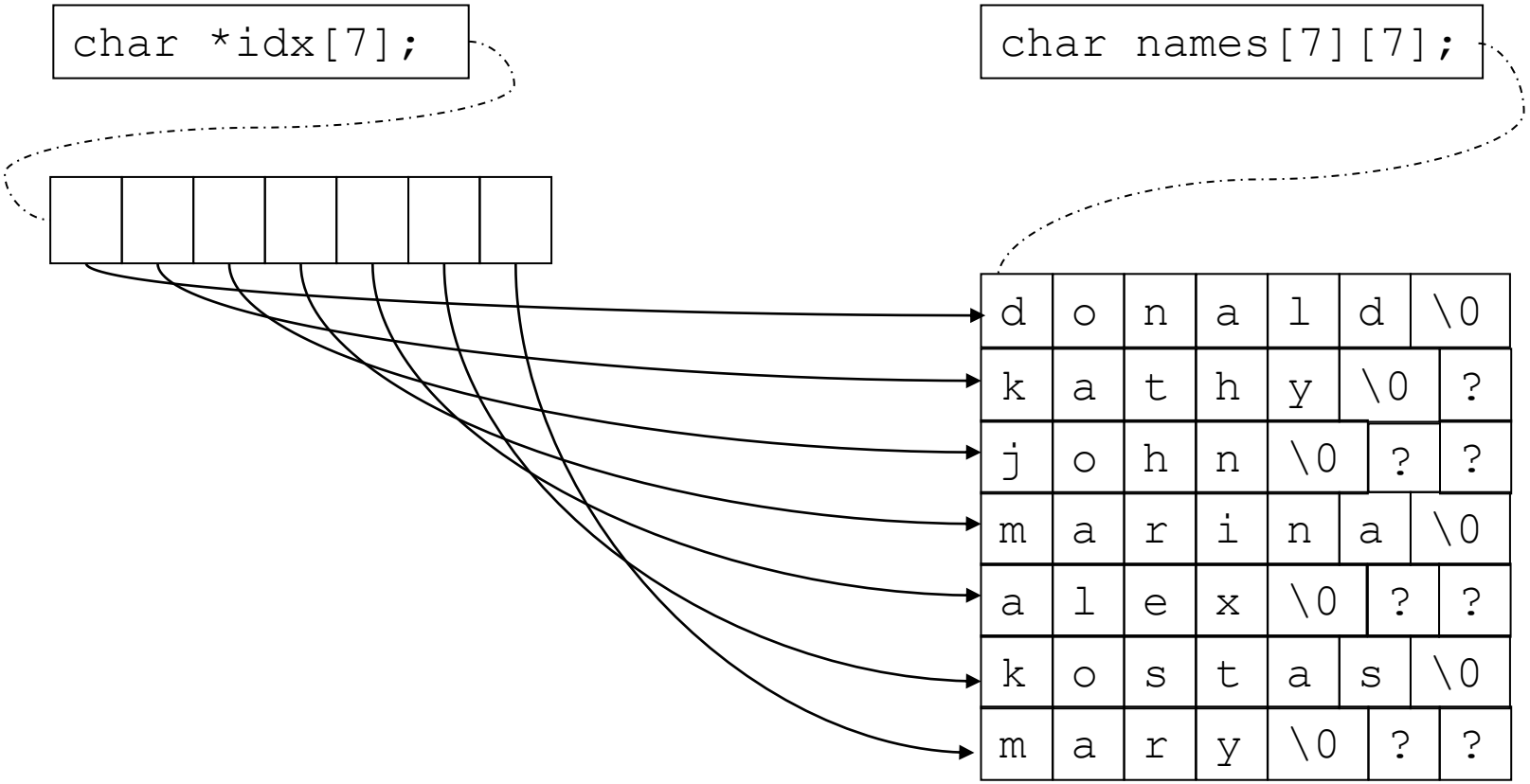
- Κάθε μετακίνηση ή ανταλλαγή στοιχείων που γίνεται κατά την ταξινόμηση, απαιτεί **τριπλή αντιγραφή** των δεδομένων
- Αν οι εγγραφές του πίνακα είναι μεγάλες σε μέγεθος, αυτό μπορεί να επιφέρει σημαντική καθυστέρηση

- Λύση: **αποφεύγουμε** τις αντιγραφές!
- Χρησιμοποιούμε έναν επιπλέον **πίνακα με δείκτες** στις εγγραφές που βρίσκονται στον κανονικό πίνακα
  - παίζει τον ρόλο ευρετηρίου
- Η ταξινόμηση του ευρετηρίου είναι γρήγορη
  - τα στοιχεία του πίνακα είναι απλοί δείκτες, **ανεξάρτητα** από το μέγεθος των εγγραφών στον κυρίως πίνακα

Ευρετήριο: **πριν** την ταξινόμηση

πίνακας με **δείκτες** σε στοιχεία του κυρίως πίνακα

κυρίως πίνακας με τις εγγραφές (π.χ. strings)



Ευρετήριο: **μετά** την ταξινόμηση

πίνακας με **δείκτες** σε στοιχεία του κυρίως πίνακα

κυρίως πίνακας με τις εγγραφές (π.χ. strings)