Προγραμματισμός 1

Ταξινόμηση - Αναζήτηση

Ταξινόμηση

- Δεδομένα:
  - Δίνεται ένας πίνακας data από N ακεραίους
- Ζητούμενο:
  - Να ταξινομηθούν τα περιεχόμενα σε αύξουσα αριθμητική σειρά:

    $\forall i : 0 <= i < N-1 \Rightarrow data[i] <= data[i+1]$
- Πώς θα κάνατε ταξινόμηση στην πράξη?
  - Ένα πακέτο από γραπτά ανά βαθμό
  - Ένα πακέτο από γραπτά ανά όνομα (έχει διαφορά?)
  - Βιβλία σε ένα ράφι (ως προς τι?)
  - Συναλλαγές ATM ανά αριθμό λογαριασμού (χωρίς να αλλάξει η χρονολογική σειρά! [stability])

Ιδέες

- Insertion sort:
  - Κράτα χωριστά τους ταξινομημένους από τους μη. Πάρε τον επόμενο μη-ταξινομημένο αριθμό, βρες τη σωστή του θέση στους ταξινομημένους και βάλε τον εκεί
- Selection sort:
  - Κράτα χωριστά τους ταξινομημένους από τους μη. Πάρε τον μικρότερο μη-ταξινομημένο αριθμό και βάλε τον στην επόμενη ελεύθερη θέση στους ταξινομημένους

Ιδέες

- Bucket sort:
  - Φτιάξε μικρότερα πακέτα γραπτών ως προς την ιδιότητα ταξινόμησης, ταξινόμησε κάθε πακέτο χωριστά και ένωσέ τα.
    - πχ. πακέτο από Α, πακέτο από Β, κ.ο.κ.
- Counting sort:
  - Μέτρα όλα τα γραπτά που έχουν βαθμό 0, 1, ... 10. Πάρε τα ένα-ένα και βάλε τα κατευθείαν στη σωστή θέση με βάση το βαθμό τους και πόσοι βαθμοί προηγούνται [δε χρειάζεται καν σύγκριση!]
    - πχ. αν ξέρουμε ότι υπάρχουν 8 γραπτά με 0, τότε το πρώτο γραπτό με 1 θα πάει στην 9η θέση.
- Και πολλές άλλες. Θα καλύψουμε selection και insertion sort.

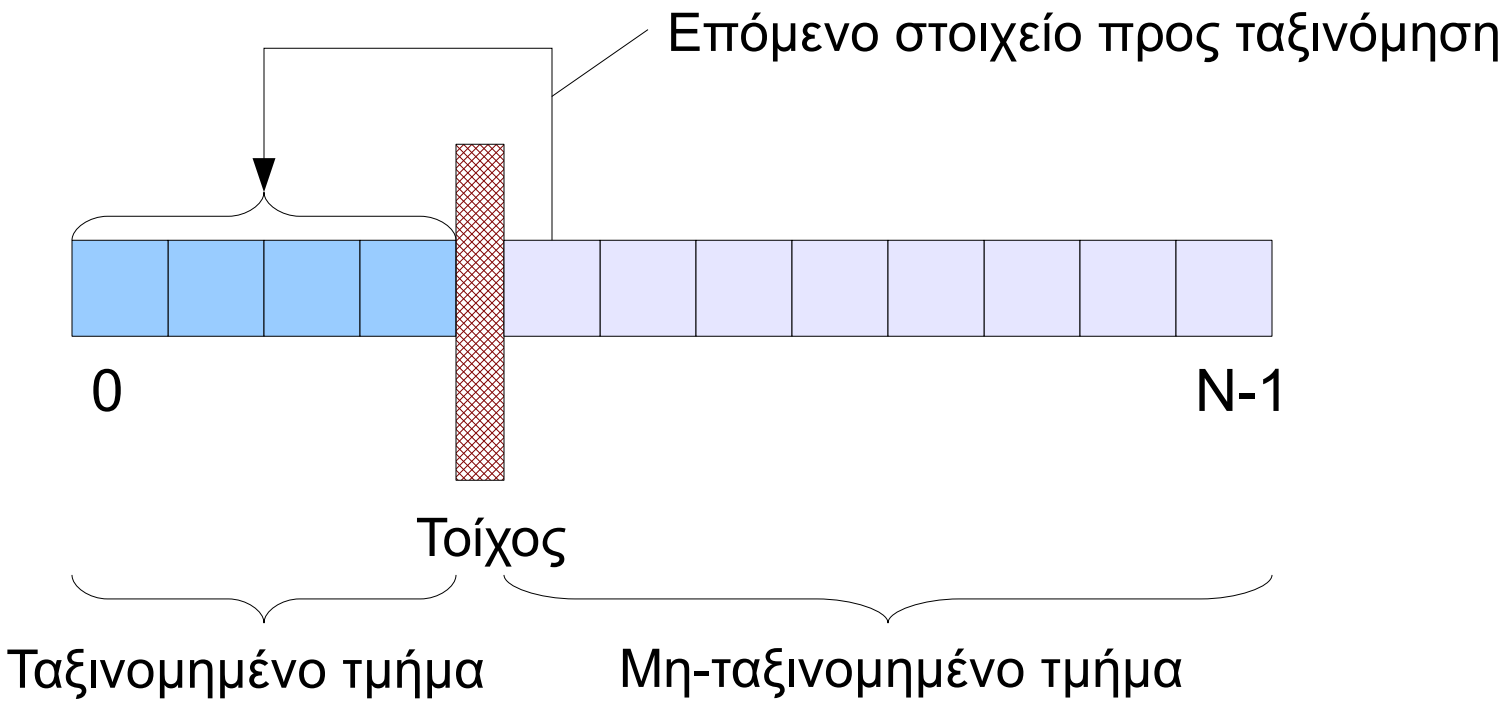
Πώς διαλέγουμε?

- Πόσο γρήγοροι είναι (πχ. σε αριθμό συγκρίσεων)
  - Για πολλά ή για λίγα δεδομένα
  - Για δεδομένα που πιθανώς είναι ήδη σε σχετικά καλή διάταξη
- Πόση επιπλέον μνήμη χρησιμοποιούν
- Κάποιοι αλγόριθμοι εκμεταλλεύονται συγκεκριμένες ιδιότητες των δεδομένων
- Περισσότερα σε μαθήματα Δομών Δεδομένων και Αλγορίθμων

Insertion sort: Ιδέα

- Φανταστείτε ένα νοητό τοίχο που **χωρίζει** τον πίνακα σε δύο μέρη: **ταξινομημένο** και **μη-ταξινομημένο**
- Το ταξινομημένο τμήμα είναι αρχικά άδειο
- Διαλέξτε τον **πρώτο** ακέραιο από το μη-ταξινομημένο τμήμα και **εισάγετέ** τον στη **σωστή θέση** στο ταξινομημένο τμήμα.
  - Ίσως χρειαστεί να μετακινήσετε τους άλλους ακεραίους στο ταξινομημένο τμήμα για να κάνετε χώρο για τον καινούργιο.
  - Ο νοητός τοίχος μετακινείται μια θέση δεξιά
- **Επαναλάβετε** μέχρι να ταξινομηθεί όλος ο πίνακας
- youtube

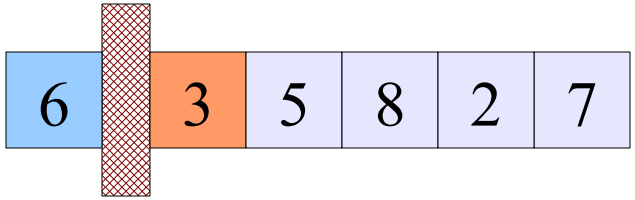
Insertion sort: Ιδέα



Σχήμα προσαρμοσμένο από το βιβλίο: "*Computer Science: A Structured Approach Using C*" by Behrouz A. Forouzan, Richard F. Gilberg

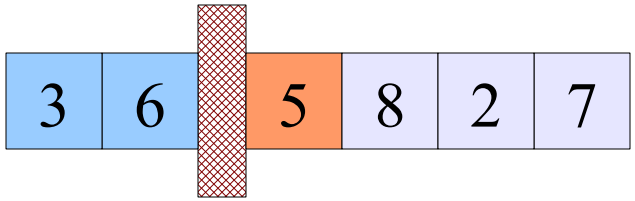
Insertion sort: Παράδειγμα

Ο πίνακας στην αρχή:



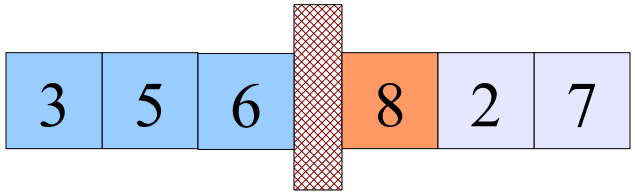
υποψήφιος για ταξινόμηση: 3

Μετά από το πέρασμα 1:



υποψήφιος για ταξινόμηση: 5

Μετά από το πέρασμα 2:

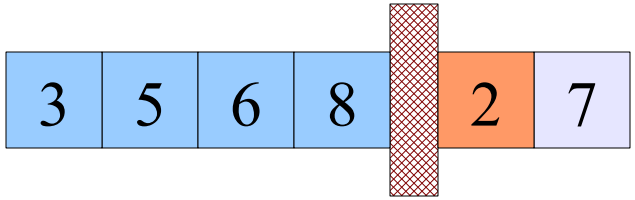


υποψήφιος για ταξινόμηση: 8

Σχήμα προσαρμοσμένο από το βιβλίο: "*Computer Science: A Structured Approach Using C*" by Behrouz A. Forouzan, Richard F. Gilberg

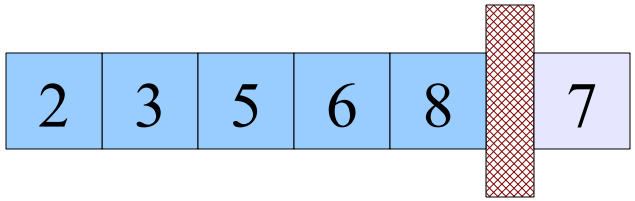
Insertion sort: Παράδειγμα

Μετά από το πέρασμα 3:



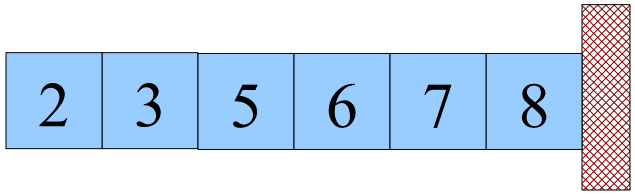
υποψήφιος για ταξινόμηση: 2

Μετά από το πέρασμα 4:



υποψήφιος για ταξινόμηση: 7

Μετά από το πέρασμα 5:

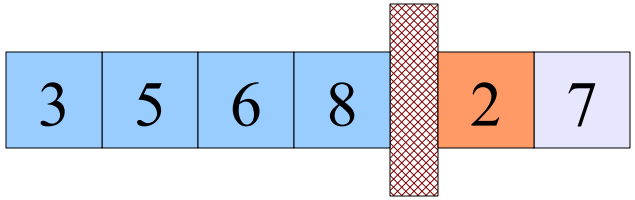


Τέλος!

Σχήμα προσαρμοσμένο από το βιβλίο: "*Computer Science: A Structured Approach Using C*" by Behrouz A. Forouzan, Richard F. Gilberg

Insertion sort: Παράδειγμα

Μετά από το πέρασμα 3:



υποψήφιος για ταξινόμηση: 2

Πώς κάνουμε χώρο?

- Αφού συγκρίνουμε το 2 με 3, 5, 6, 8 ανακαλύπτουμε ότι η σωστή του τελική θέση είναι η 0.
- Πρέπει τα στοιχεία από τη θέση 0 μέχρι και τη θέση 3 να πάνε από μια θέση δεξιά. Προσοχή: Πριν κάνουμε τη μετακίνηση, "σώζουμε" το 2 γιατί θα σβηστεί.
- Μετά, μπορούμε να βάλουμε το 2 στη θέση 0.

Σχήμα προσαρμοσμένο από το βιβλίο: "*Computer Science: A Structured Approach Using C*" by Behrouz A. Forouzan, Richard F. Gilberg

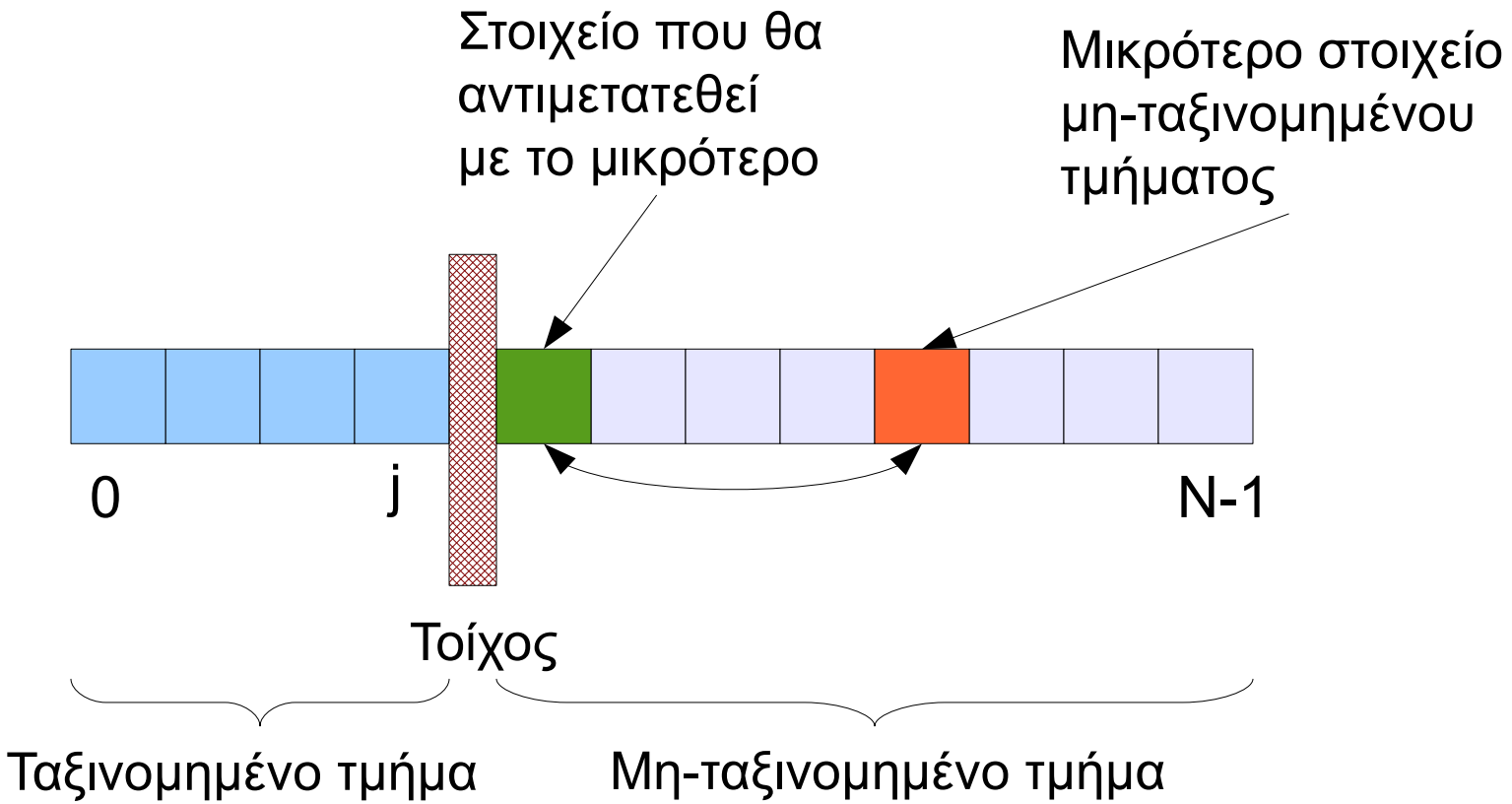
Insertion sort: Κώδικας

```
void insertionSort ( int nums[], int size) {
   int wall, numToSort, pos;

   for (wall = 1; wall < size; wall++) {
      numToSort = nums[wall];   /* πρώτο μη-ταξινομημένο */
      pos = wall-1;

      /* όσο δεν έχουμε φτάσει στην άκρη, και
         όσο βρίσκουμε μεγαλύτερα στοιχεία,  */
      while (pos >= 0 && nums[pos] > numToSort) {
          nums[pos+1] = nums[pos];   /* κάνουμε χώρο */
          pos--;
      }
      /* η θέση pos+1 είναι εκεί  όπου ανοίξαμε χώρο */
      nums[pos+1] = numToSort;
   }
}
```

Selection sort: Ιδέα

- Φανταστείτε ένα νοητό τοίχο που **χωρίζει** τον πίνακα σε δύο μέρη: **ταξινομημένο** και **μη-ταξινομημένο**
- Το ταξινομημένο τμήμα είναι αρχικά άδειο
- Διαλέξτε τον **μικρότερο** ακέραιο από το μη-ταξινομημένο τμήμα και **αντιμεταθέστε** το με τον πρώτο ακέραιο μετά τον τοίχο.
  - Ο νοητός τοίχος μετακινείται μια θέση δεξιά
- **Επαναλάβετε** μέχρι να ταξινομηθεί όλος ο πίνακας.
- youtube

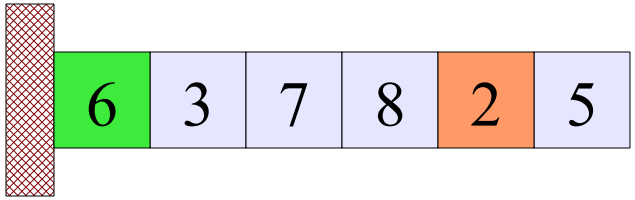
Selection sort: Ιδέα



Σχήμα προσαρμοσμένο από το βιβλίο: "*Computer Science: A Structured Approach Using C*" by Behrouz A. Forouzan, Richard F. Gilberg

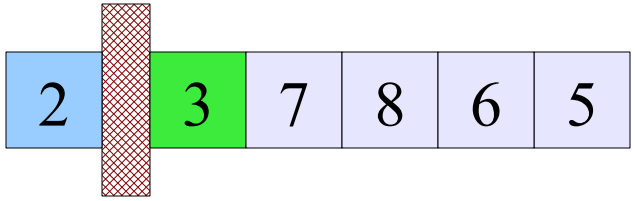
Selection sort: Παράδειγμα

Ο πίνακας στην αρχή:



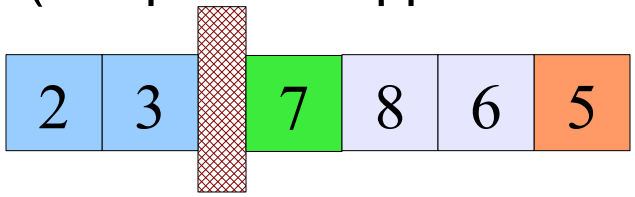
μικρότερο: 2, αντιμετ. με 6

Μετά από το πέρασμα 1:



μικρότερο: 3
(αντιμετάθεση με τον εαυτό του?

Μετά από το πέρασμα 2:

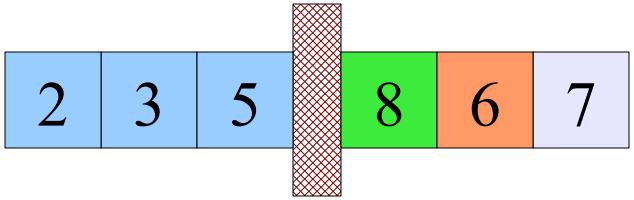


μικρότερο: 5, αντιμετ. με 7

Σχήμα προσαρμοσμένο από το βιβλίο: "*Computer Science: A Structured Approach Using C*" by Behrouz A. Forouzan, Richard F. Gilberg

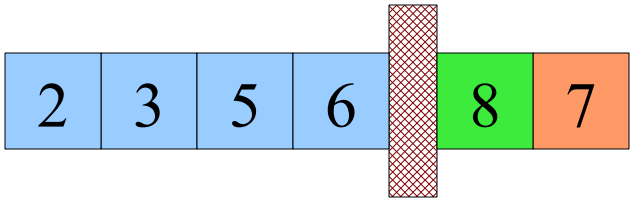
Selection sort: Παράδειγμα

Μετά από το πέρασμα 3:



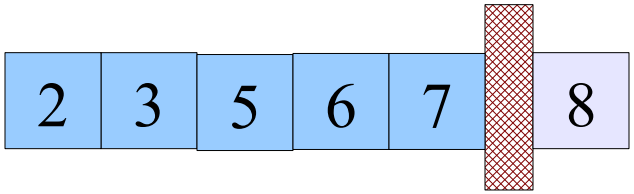
μικρότερο: 6, αντιμετ. με 8

Μετά από το πέρασμα 4:



μικρότερο: 7, αντιμετ. με 8

Μετά από το πέρασμα 5:



Τελειώσαμε!

Σχήμα προσαρμοσμένο από το βιβλίο: "*Computer Science: A Structured Approach Using C*" by Behrouz A. Forouzan, Richard F. Gilberg

Selection sort: κώδικας

```
void selectionSort(int data[], int size)  {
    int i,wall, min, temp;
    for (wall=0; wall < size; wall++) {
      /* εύρεση θέσης μικρότερου στοιχείου */
      min = wall; /* ξεκινώντας από τον τοίχο, */
      for (i=wall+1; i < size; i++) {
          if (data[i] < data[min]) {
              min = i; /* δες αν υπάρχει μικρότερο */
          }
      }
      /* αντιμετάθεση μικρότερου με αυτό που βρίσκεται δεξιά του τοίχου */
      if (min != wall) { /* (αν χρειάζεται) */
          temp = data[wall];
          data[wall] = data[min];
          data[min] = temp;
      }
    }
}
```

Συγκρίσεις

- Ποιος αλγόριθμος σας φαίνεται ότι κάνει περισσότερη δουλειά?
- Αν ένας πίνακας είναι ήδη ταξινομημένος, θα γλιτώσει λίγη δουλειά ο insertion sort? Ο selection sort?

Αναζήτηση

- Δεδομένα:
  - Δίνεται ένας πίνακας data από N ακεραίους και μια επιθυμητή τιμή value
- Ζητούμενο:
  - Να βρεθεί η θέση όπου έχει αποθηκευθεί η τιμή value στον πίνακα (εφόσον υπάρχει)
- Πώς κάνουμε αναζήτηση στην πράξη?
  - Ένα συγκεκριμένο τραπουλόχαρτο
  - Στον τηλεφωνικό κατάλογο

Ιδέες

- Ακολουθιακή/Σειριακή αναζήτηση (linear search):
  - Ξεκινώντας από το πρώτο στοιχείο, τα εξετάζουμε ένα-ένα μέχρις ώτου να βρούμε αυτό που αναζητούμε (οπότε επιστρέφουμε τη θέση του) ή να εξαντλήσουμε τα στοιχεία (οπότε επιστρέφουμε αποτυχία)
- Παρατηρήσεις:
  - Αν είμαστε άτυχοι, το στοιχείο που ψάχνουμε δεν υπάρχει καθόλου ή είναι στην τελευταία θέση, και το βρίσκουμε μετά από N ελέγχους!
  - Κατά μέσο όρο χρειάζονται N/2 βήματα σύγκρισης.

Ιδέες

- Δυαδική αναζήτηση (binary search):
  - Μόνο αν τα δεδομένα είναι ταξινομημένα (ας πούμε σε αύξουσα σειρά)
  - Εξετάζουμε το στοιχείο στη μέση. Αν είναι αυτό που ψάχνουμε, επιστρέφουμε τη θέση του. Αν είναι μικρότερο από αυτό που ψάχνουμε, τότε αρκεί να κάνουμε αναζήτηση με τον ίδιο τρόπο στο δεξί τμήμα, αλλιώς στο αριστερό.
  - Σε κάθε βήμα, το νέο τμήμα έχει μισό μέγεθος από το προηγούμενο. Κάποια στιγμή, το τμήμα που ελέγχουμε δε θα έχει κανένα στοιχείο, οπότε επιστρέφουμε αποτυχία.
- Παρατηρήσεις:
  - Πολύ πιο γρήγορη από σειριακή (χρειάζονται log(N) βήματα)
  - Αν έχουμε να κάνουμε πολλές αναζητήσεις, συμφέρει να ταξινομήσουμε πρώτα!

Linear search: Κώδικας

```
/* σε μη ταξινομημένο πίνακα */
int linearSearch(int data[], int value, int size) {
    int pos;
    for (pos=0; pos < size; pos++) {
        if (data[pos] == value) {
            return pos;
        }
    }
    return -1;  /* άκυρη θέση, δηλώνει αποτυχία */
}
```

Linear search: Κώδικας

```
/* σε ταξινομημένο πίνακα */
int linearSearchSorted(int data[], int value, int size){
    int pos;
    for (pos=0; pos < size; pos++) {
        if (data[pos] >= value) {
            break;
        }
    }
    if (pos == size || data[pos] != value) {
        return -1; /* άκυρη θέση, δηλώνει αποτυχία */
    }
    else {
        return pos;
    }
}
```

Linear search: Κώδικας

```
/* σε ταξινομημένο πίνακα */
int linearSearchSorted(int data[], int value, int size)
{
    int pos;
    for (pos=0; pos < size; pos++) {
        if (data[pos] >= value) {
            break;
        }
    }
    if (pos == size || data[pos] != value) {
        printf("not found\n");
        return -1; /* άκυρη θέση, δηλώνει αποτυχία */
    }
    else {
        printf("Found
        return pos;
    }
}
```

η σύμβαση εκτέλεσης του `||` εγγυάται ότι **δεν** θα επιχειρηθεί πρόσβαση `data[pos]` αν το `pos` φτάσει το όριο `size` του πίνακα

Linear search: Κώδικας (παραλλαγή)

```
/* σε ταξινομημένο πίνακα */
int linearSearchSorted(int data[], int value, int size){
    int pos;
    for (pos=0; pos < size && data[pos]<value; pos++){}

    if (pos == size || data[pos] != value) {
        printf("not found\n");
        return -1; /* άκυρη θέση, δηλώνει αποτυχία */
    }
    else {
        printf("Found at position %d\n", pos);
        return pos;
    }
}
```

η σύμβαση εκτέλεσης του `&&` εγγυάται ότι **δεν** θα επιχειρηθεί πρόσβαση `data[pos]` αν το `pos` φτάσει το όριο `size` του πίνακα

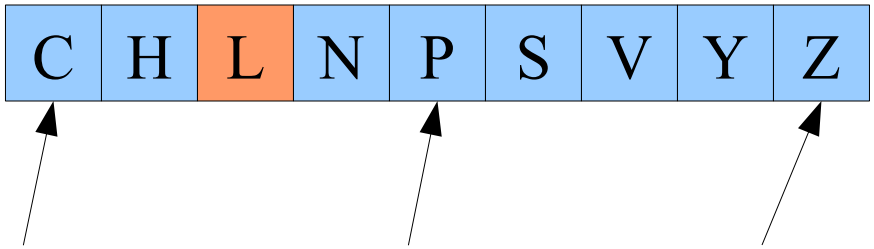
Binary search: Ο αλγόριθμος

1. Θέτουμε τα όρια αναζήτησης (δηλαδή τα όρια του τμήματος του πίνακα που μας ενδιαφέρει) αρχικά σε begin = 0 και end = N-1
2. Βρίσκουμε το μεσαίο στοιχείο του πίνακα:

   middle = (begin + end) / 2
3. Αν begin > end τελειώσαμε (με αποτυχία)
4. Αν data[middle] < value, συνεχίζουμε με τον μισό υποπίνακα που έχει στοιχεία με μεγαλύτερες τιμές του data[middle], δηλαδή θέτουμε begin = middle + 1 και πάμε πάλι στο βημα 2. Το end δεν αλλάζει.
5. Αν data[middle] > value, συνεχίζουμε με τον μισό υποπίνακα που έχει στοιχεία με μεγαλύτερες τιμές του data[middle], δηλαδή θέτουμε end = middle - 1 και πάμε πάλι στο βημα 2. Το begin δεν αλλάζει.

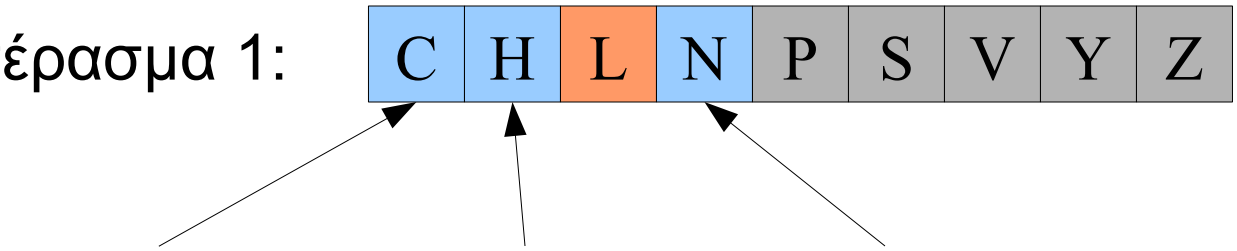
Binary search: Παράδειγμα 1

Αρχικά:
Ψάχνουμε το L



data[middle] > value

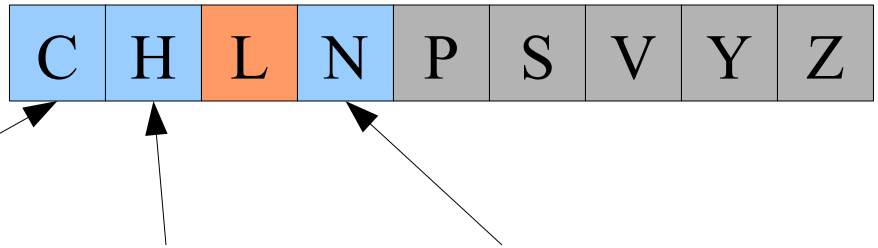
Μετά από το πέρασμα 1:



data[middle] < value

Binary search: Παράδειγμα 1

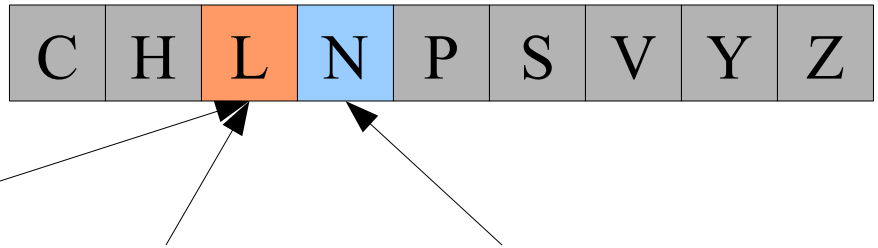
Μετά από το πέρασμα 1:
(από προηγ. διαφάνεια)



begin = 0 middle = 1 end = 3

data[middle] < value

Μετά από το πέρασμα 2:

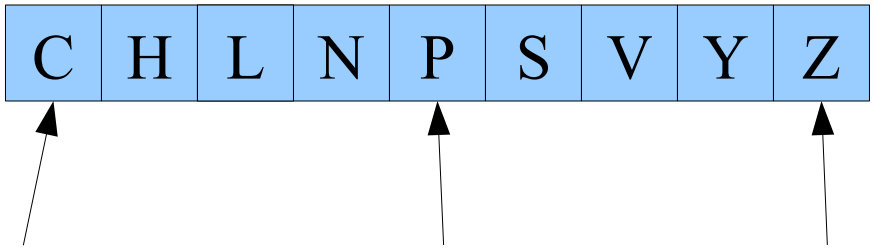


begin = 2 middle = 2 end = 3

data[middle] == value

Binary search: Παράδειγμα 2

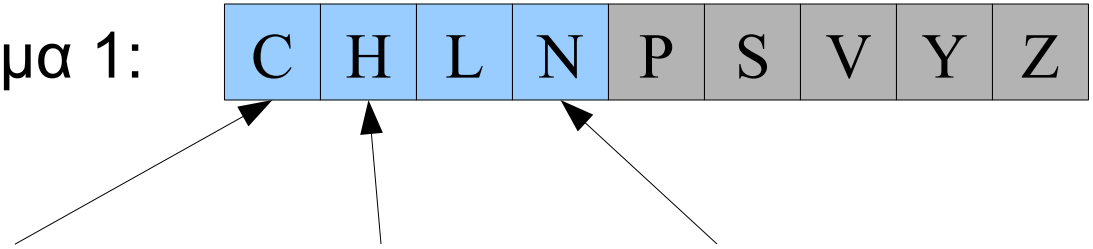
Αρχικά:
Ψάχνουμε το M



data[middle] > value

Μετά από το πέρασμα 1:

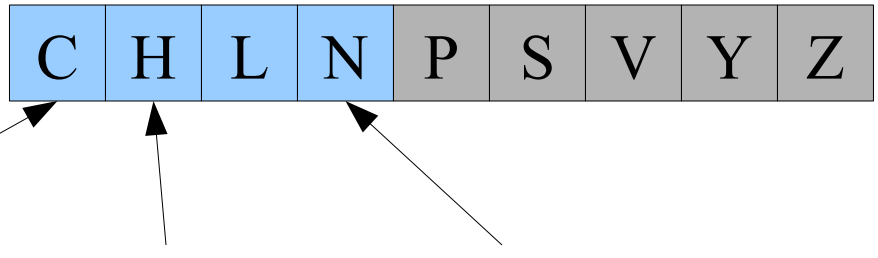
στόχος: M



data[middle] < value

Binary search: Παράδειγμα 2

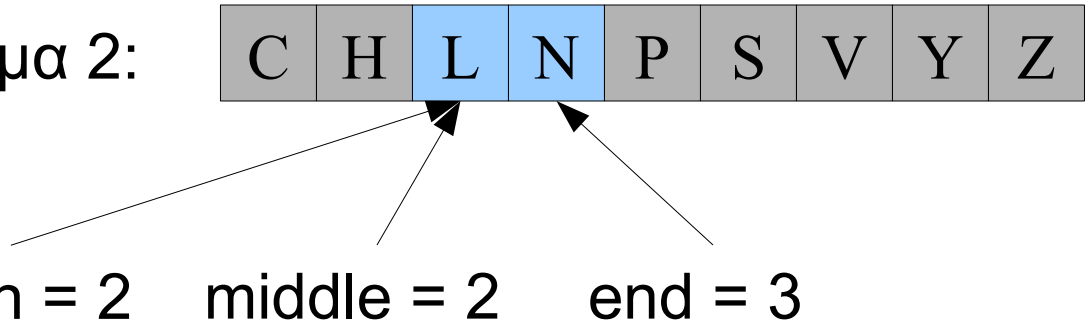
Μετά από το πέρασμα 1:
(από προηγ. διαφάνεια)



data[middle] < value

Μετά από το πέρασμα 2:

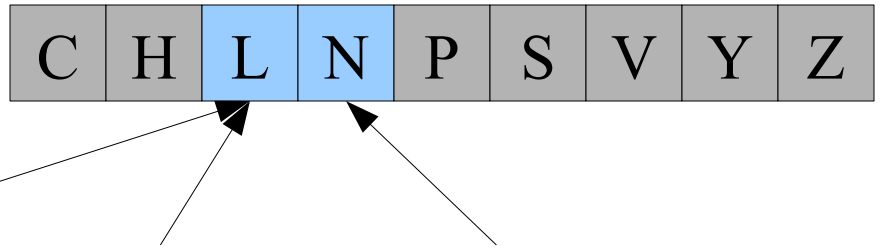
στόχος: M



data[middle] < value

Binary search: Παράδειγμα 2

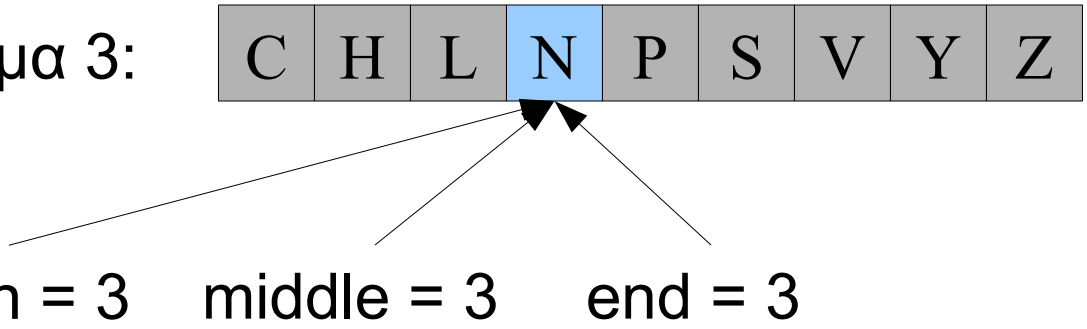
Μετά από το πέρασμα 2:
(από προηγ. διαφάνεια)



data[middle] < value

Μετά από το πέρασμα 3:

στόχος: M

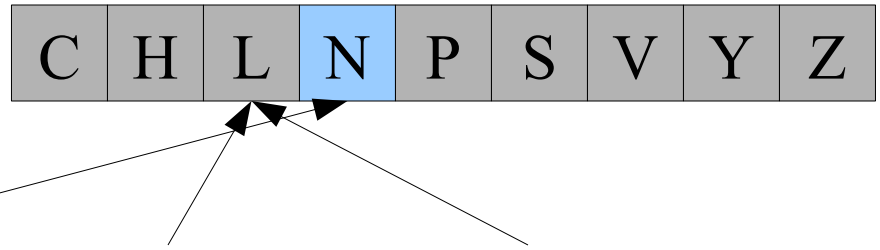


data[middle] > value

Binary search: Παράδειγμα 2

Μετά από το πέρασμα 3:
(από προηγ. διαφάνεια)

στόχος: M



begin = 3 middle = 2 end = 2

begin > end

Αποτυχία!

Binary search: κώδικας

```
int binarySearch(int data[], int value, int size) {
    int begin, end, middle;
    begin = 0;
    end = size-1;
    while (begin <= end) {
        middle = (begin + end) / 2;
        if (data[middle] == value) {
            return middle;
        }
        else if (data[middle] < value) {
            begin = middle + 1;
        }
        else {   /* data[middle] > value */
            end = middle - 1;
        }
     }
     return -1;
}
```