Δομές δεδομένων (2)

Αλγόριθμοι

Διδάσκων: Β. Ράπτης

Παράγωγοι τύποι (struct) – σύνοψη προηγουμένων

Πίνακες: πολλές μεταβλητές *ίδιου* τύπου

Παράγωγοι τύποι ή **Δομές** (**struct**): ομαδοποίηση μεταβλητών *διαφορετικού* τύπου

```
struct Student
{
        int AM;
        char *name;
        char *fname;
        int semester;
        float grade;
};
```

```
typedef struct Student  STUDENT;
STUDENT someone;
someone.AM = 25;                τελεστής μέλους “τελεία”
someone.name = "Tade";
κλπ ...
STUDENT  *sptr;        μπορώ να ορίσω δείκτες σε δομές
sptr = &someone;
sptr->AM = 30;                  τότε: τελεστής μέλους “βέλος”.

STUDENT pliroforiki[80];        πίνακες παραγώγων τύπων
Pliroforiki[0] = someone        με απλή ανάθεση εξισώνω τα μέλη
```

Ως ορίσματα σε συναρτήσεις:, π.χ.:

```
void func(struct Student *pliroforiki, int  num);
```

Τύποι με μέλη άλλους παράγωγους τύπους:

```
struct new_type {
        int age;                    x.stdnt.AM = 140;
        STUDENT stdnt;
} x;
```

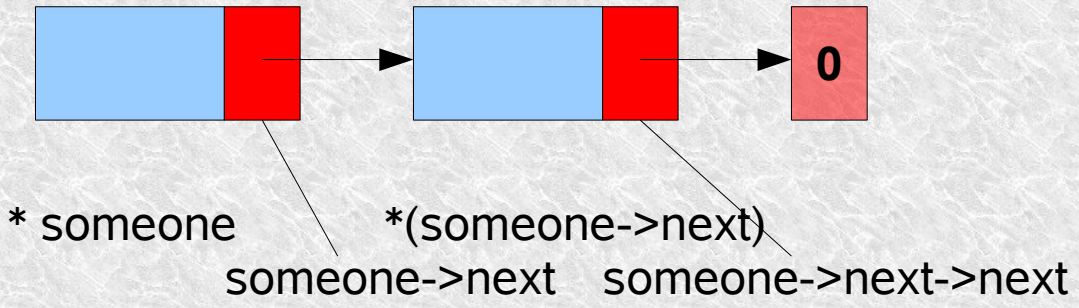
Παράγωγοι τύποι (struct) –αυτοαναφορικοί τύποι

Πιο σημαντική ιδιότητα: αυτοαναφορά = μέλος δείκτης στον τύπο όπου περιέχεται
Επιτρέπει δημιουργία πολύπλοκων δομών δεδομένων

```
struct Student
{     int AM;
      char *name;
      char *fname;
      int semester;
      float grade;
      struct Student *next;
};
```

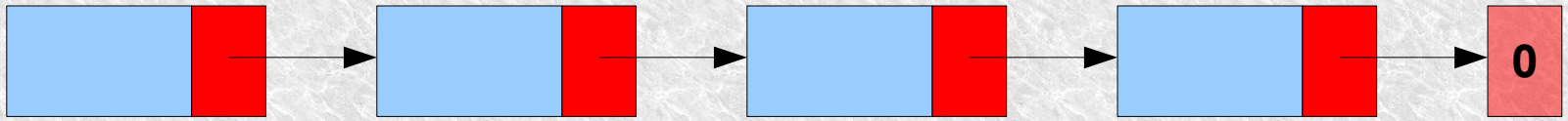
```
typedef struct Student  STUDENT;
STUDENT *someone;
someone = (STUDENT *) malloc (sizeof( STUDENT ) );
someone->next = (STUDENT *) malloc (sizeof( STUDENT
) );
someone->next->next = NULL;
```

Τώρα, *(someone.next) *είναι μεταβλητή τύπου* STUDENT



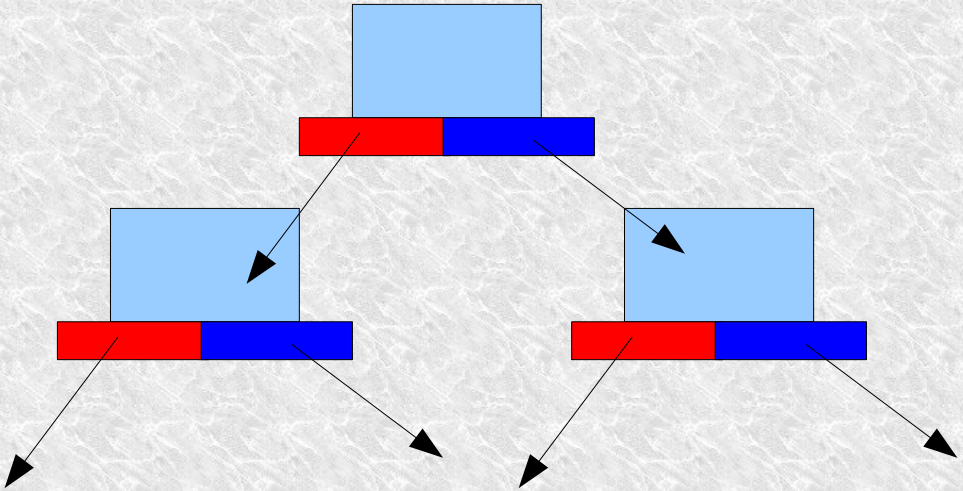
Δομές δεδομένων

Γραμμική Λίστα

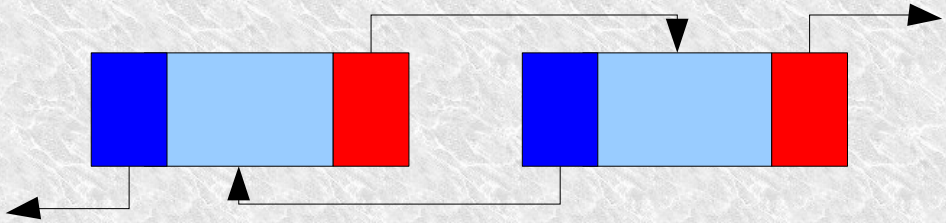


Στοίβα = γραμμική λίστα όπου προσθαφαιρούνται στοιχεία μόνο στην αρχή: LIFO
Ουρά = γραμμική λίστα όπου: εισάγονται στοιχεία μόνο στην αρχή,
εξάγονται στοιχεία μόνο από το τέλος: FIFO

Δέντρο



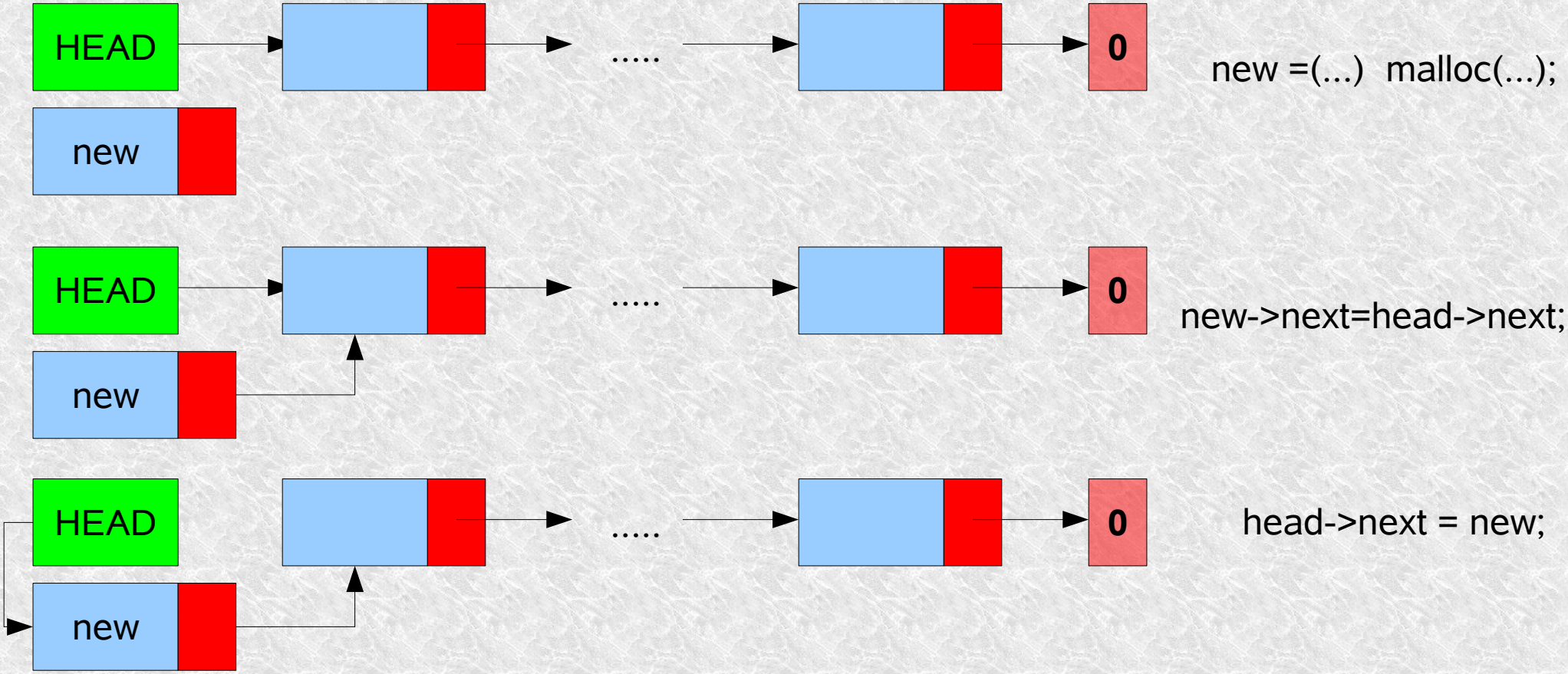
Διπλά συνδεδεμένη λίστα



```
struct tree {
        ....
        struct tree *left, right;
}
```

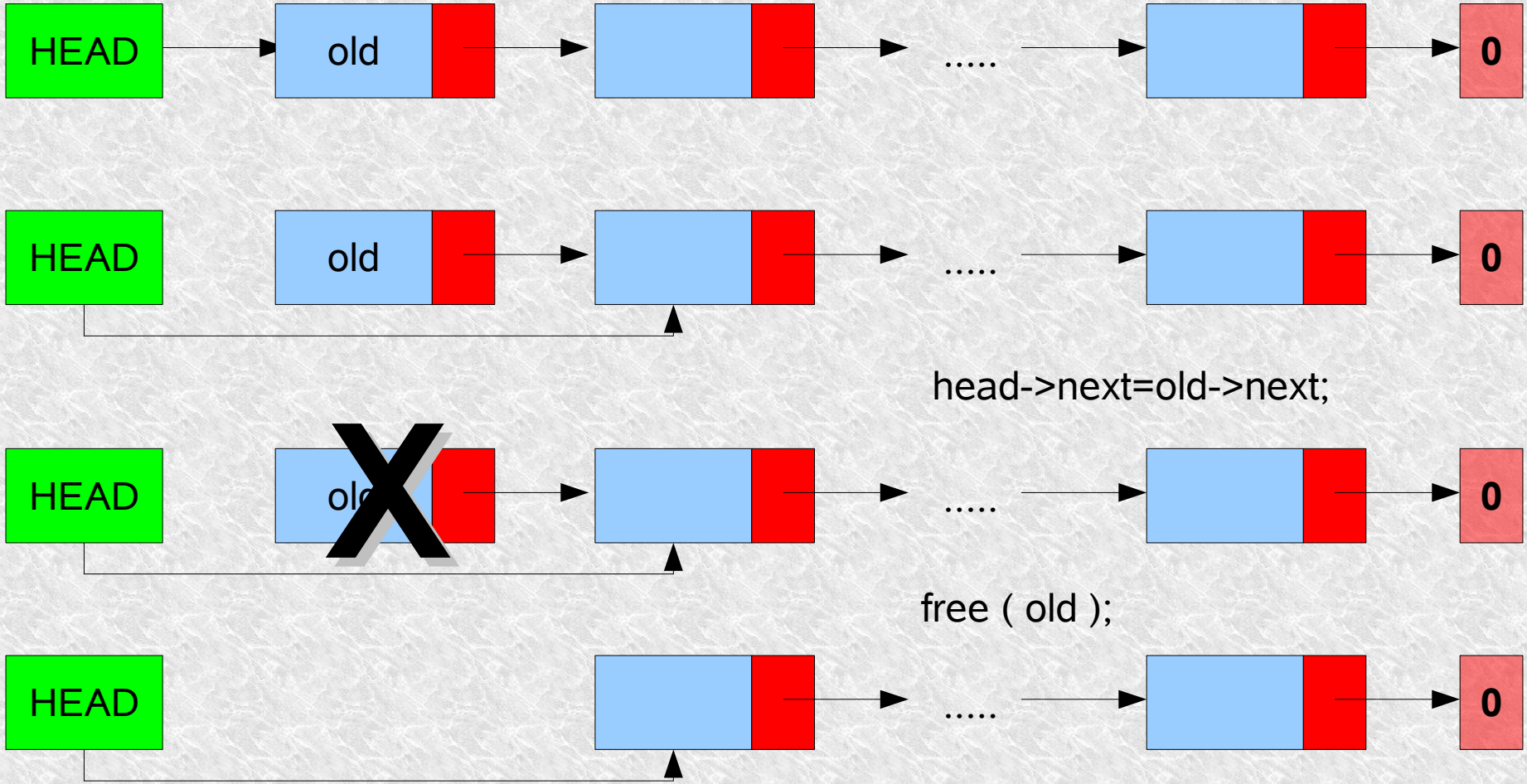
Υλοποίηση στοίβας

Προσθήκη στοιχείου (push)



Υλοποίηση στοίβας

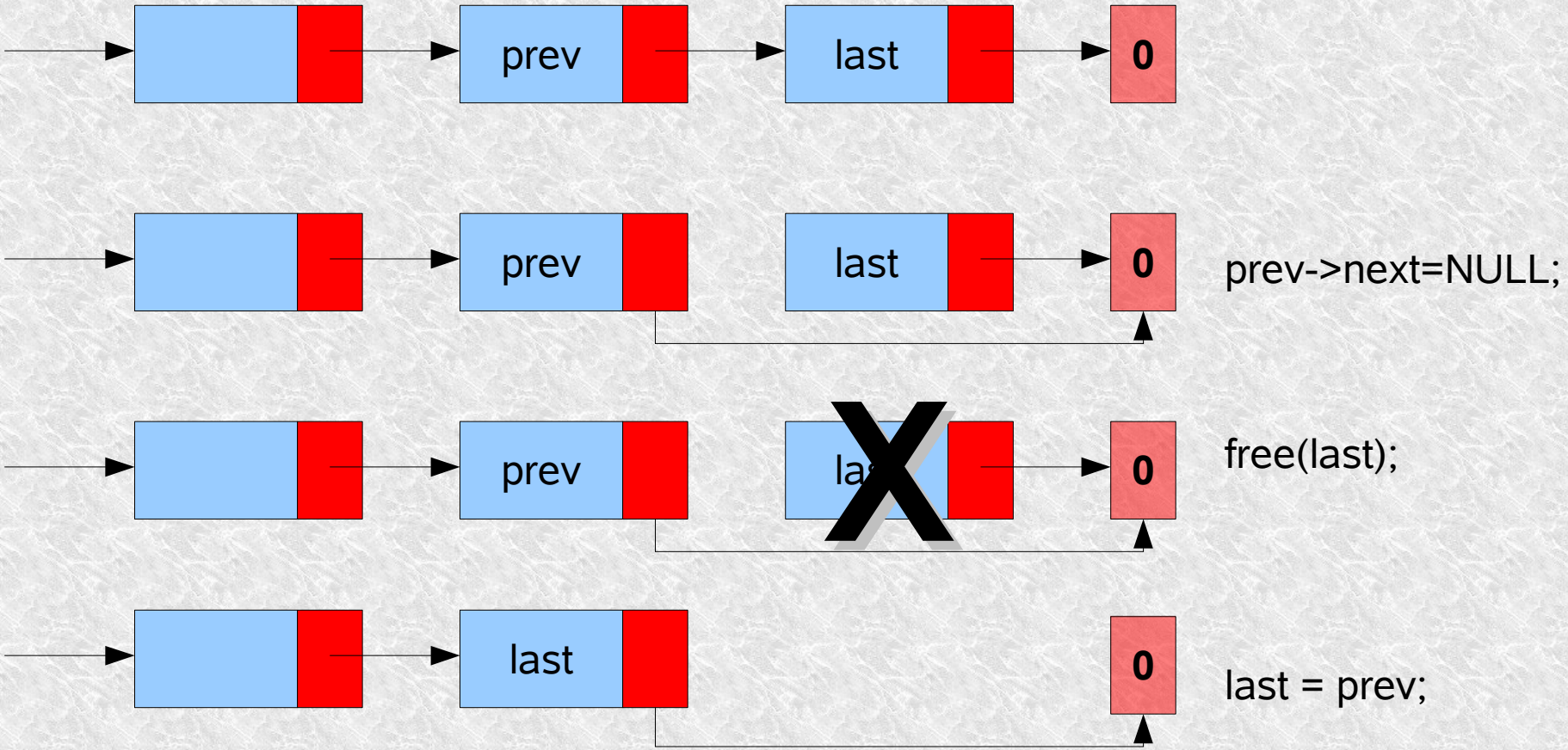
Αφαίρεση στοιχείου (pop)



Υλοποίηση ουράς

Εισαγωγή στοιχείου: όπως και στη στοίβα

Εξαγωγή στοιχείου:



Πρέπει να διατρέξω τη λίστα για να βρω το prev
πιο εύκολα γίνεται με διπλά συνδεδεμένη λίστα

Δομές δεδομένων (2)

Αλγόριθμοι

Διδάσκων: Β. Ράπτης

Αλγόριθμοι: ταξινόμηση και αναζήτηση

Αναζήτηση στοιχείων: βασική και συχνά χρησιμοποιούμενη (π.χ. βάσεις δεδομένων)
Ορισμός: εύρεση συγκεκριμένου στοιχείου από μεγάλο σύνολο δεδομένων

Ταξινόμηση: διευκολύνει σημαντικά την αναζήτηση
Ορισμός: τοποθέτηση δεδομένων σε σειρά (π.χ. αύξουσα, φθίνουσα) σύμφωνα με ένα χαρακτηριστικό (κλειδί)

Δυσκολία αυξάνει με **μέγεθος** του συνόλου δεδομένων και **πολυπλοκότητά** τους.

Δεν υπάρχει γενική μέθοδος (δλδ αποδοτική για κάθε μέγεθος προβλήματος)

Ποικιλία τεχνικών αναζήτησης και ταξινόμησης για δεδομένα τόσο σε πίνακες όσο και σε λίστες ή άλλες δομές δεδομένων.

Ταξινόμηση, αναζήτηση: συνήθεις ενέργειες => σχετικές **πρότυπες συναρτήσεις**

Τεχνικές ταξινόμησης

Ονομασία	Τεχνική	Απόδοση
Μέθοδος **φυσαλλίδας** (bubble sort)	αντιμετάθεση γειτονικών στοιχείων	$O(N^2)$
Ταξινόμηση **παρεμβολής** (straight insertion sort)	Τοποθέτηση στοιχείου σε σχέση με τα ήδη ταξινομημένα	$O(N^2)$
Μέθοδος του **Shell** (Shell sort)	Εφαρμόζει τη μέθοδο παρεμβολής σε ένα υποσύνολο των δεδομένων που μεγαλώνει βαθμιαία μέχρι να τα περιλάβει όλα	$O(N^{3/2})$
Μέθοδος **σωρού** (heap sort)	Τακτοποίηση σε δέντρο όπου κάθε στοιχείο είναι > από τους 2 κλάδους του. Επαναληπτική τακτοποίηση θέτοντας κάθε φορά άλλο στοιχείο στη ρίζα του δέντρου	$O(N \log_2 N)$
Ταχεία ταξινόμηση (quick sort)	Αναδρομική ή επαναληπτική κλήση συνάρτησης που χωρίζει τον πίνακα σε δύο μέρη και αντιμεταθέτει το min του ενός και max του άλλου.	εξαρτάται` συχνά, η ταχύτερη

Συνάρτηση πρότυπης βιβλιοθήκης για ταχεία ταξινόμηση

```
#include <stdlib.h>

void qsort(const void *base, size_t count, size_t width,
           int (*compar) (const void *, const void * ) );
```

base = δείχνει στην αρχή του πίνακα προς ταξινόμηση
count = πλήθος στοιχείων προς ταξινόμηση
width = μέγεθος κάθε στοιχείου, σε bytes
compar = δείχνει σε συνάρτηση που ορίζει ο χρήστης. Παίρνει ως ορίσματα δύο στοιχεία για σύγκριση και επιστρέφει 0 αν είναι ίσα, >0 αν το πρώτο είναι μεγαλύτερο και < 0 αν είναι μικρότερο.

Παράδειγμα:

```
int cmp_double(const void *x, const void *y) { /* συνάρτηση που όρισε ο χρήστης */
    return ( (*x == *y) ? 0 : ( (*x > *y) ? 1 : -1 ) );
}
...
double array[100];
...
qsort(array, 100, sizeof(double), cmp_double);
```

Τεχνικές αναζήτησης

Σειριακή αναζήτηση:
- Παίρνουμε τα στοιχεία με τη σειρά μέχρι να βρούμε το ζητούμενο.
- Η απλούστερη ως προς την υλοποίηση
- Απολύτως γενική (δεν απαιτεί ταξινόμηση)
- Μπορεί να γίνει πολύ αργή
- Αλλά μερικές φορές χρησιμοποιείται αναγκαστικά (π.χ. σειριακά αρχεία)

Δυαδική αναζήτηση:
- Απαιτεί ταξινόμηση
- Χωρίζει το σύνολο στη μέση
- Αν μεσαίο στοιχείο δεν είναι το ζητούμενο, περιορίζεται στο ένα ήμισυ του συνόλου
- Γενικά, πιο γρήγορη από τη σειριακή
- Υπάρχει σε παραλλαγές που μπορεί να είνια ταχύτερες, ανάλογα με τη φύση των δεδομένων

Συνάρτηση πρότυπη βιβλιοθήκης για δυαδική αναζήτηση

```
#include <stdlib.h>

void *bsearch(const void *key, const void *base, size_t count, size_t width,
              int (*compar) (const void *, const void * ) );
```

key = διεύθυνση του δεδομένου προς εντοπισμό μέσα στον πίνακα
base = δείχνει στην αρχή του πίνακα προς διερεύνηση
count = πλήθος στοιχείων προς ταξινόμηση
width = μέγεθος κάθε στοιχείου, σε bytes
compar = δείχνει σε συνάρτηση που ορίζει ο χρήστης. Παίρνει ως ορίσματα δύο στοιχεία για σύγκριση και επιστρέφει 0 αν είναι ίσα, >0 αν το πρώτο είναι μεγαλύτερο και < 0 αν είναι μικρότερο.

Παράδειγμα:

```
int cmp_double(const void *x, const void *y) { /* συνάρτηση που όρισε ο χρήστης */
      return ( (*x == *y) ? 0 : ( (*x > *y) ? 1 : -1 ) );
}
...
double number, array[100];
...
bsearch(&number, array, 100, sizeof(double), cmp_double);
```