Ταξινόμηση πινάκων

- Υπάρχουν διάφορες μέθοδοι, άλλες λιγότερο, άλλες περισσότερο αποδοτικές, για την ταξινόμηση των στοιχείων ενός πίνακα.
- Η ταξινόμηση ενός πίνακα έχει νόημα όταν υπάρχει μία σχέση διάταξης στο σύνολο από το οποίο παίρνουν τιμές τα στοιχεία του πίνακα. Για αριθμούς (ακέραιους ή κινητής υποδιαστολής) έχουμε την αριθμητική διάταξη, για συμβολοσειρές την αλφαβητική. Για άλλους τύπους δεδομένων, πρέπει να έχει ορισθεί σαφώς η σχέση διάταξης. Για παράδειγμα, διάταξη μεταξύ δομών μπορεί να οριστεί με βάση τη διάταξη ως προς ένα μέλος-κλειδί της δομής (αριθμητικό ή συμβολοσειρά).
- Στη συνέχεια, οι αλγόριθμοι, τα παραδείγματα και τα προγράμματα που παρουσιάζονται στοχεύουν στην ταξινόμηση πινάκων, σε αύξουσα σειρά, με στοιχεία που είναι πραγματικοί αριθμοί διπλής ακρίβειας.
- Οι μέθοδοι ταξινόμησης μπορούν να προσαρμοσθούν εύκολα και για περιπτώσεις άλλων μονοδιάστατων δομών δεδομένων, για παράδειγμα συνδεδεμένων λιστών.

- Ένα θέμα που έχει γενικότερο ενδιαφέρον στους αλγορίθμους είναι ο χρόνος που χρειάζεται για την εκτέλεσή τους, ανάλογα με το μέγεθος της εισόδου τους. Αυτό εκφράζεται με την πολυπλοκότητα χρόνου τους. Οι μέθοδοι ταξινόμησης έχουν διάφορες πολυπλοκότητες χρόνου, οι οποίες αντανακλούν την αποδοτικότητά τους.

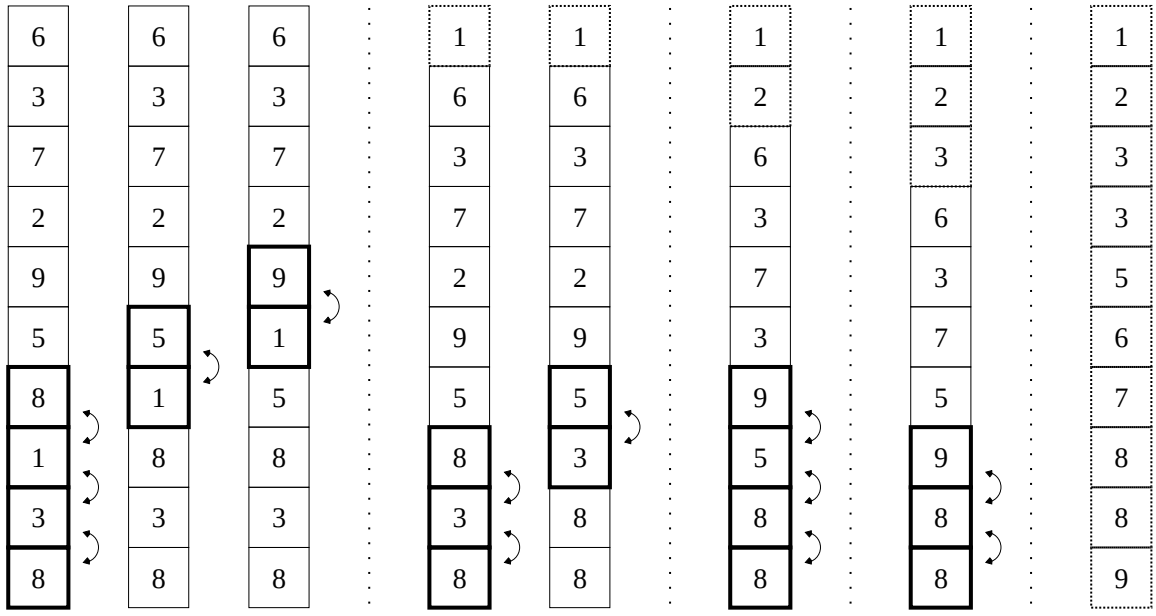
- Για να εκφράσουμε την πολυπλοκότητα ενός αλγορίθμου, συνήθως χρησιμοποιούμε τον συμβολισμό O. Όταν η είσοδος ενός αλγορίθμου έχει μέγεθος n (για παράδειγμα το πλήθος των στοιχείων ενός πίνακα που θέλουμε να ταξινομήσουμε), λέγοντας ότι η πολυπλοκότητα χρόνου του αλγορίθμου είναι $O(f(n))$, εννοούμε ότι ο χρόνος εκτέλεσής του, $T(n)$, δεν έχει μεγαλύτερο ρυθμό αύξησης από αυτόν της συνάρτησης $f(n)$, όσο αυξάνει το n. Δηλαδή υπάρχουν C και n_0 τέτοια ώστε $T(n) < C \cdot f(n)$ για κάθε $n > n_0$.

- Για παράδειγμα, η πολυπλοκότητα του αλγορίθμου για την εύρεση της μέσης τιμής n αριθμών είναι $O(n)$, ενώ η πολυπλοκότητα του αλγορίθμου κατασκευής ενός μαγικού τετραγώνου $n \times n$ είναι $O(n^2)$.

- Σε ορισμένες περιπτώσεις, η χρονική απόδοση ενός αλγορίθμου για είσοδο μεγέθους n εξαρτάται και από το ποια είναι η συγκεκριμένη είσοδος. Στις περιπτώσεις αυτές, μπορούμε να αναφερόμαστε στην πολυπλοκότητα χείριστης περίπτωσης (για την πιο "δύσκολη" είσοδο) ή στη μέση πολυπλοκότητα (μέση τιμή απόδοσης για όλες τις πιθανές εισόδους).

- *Η μέθοδος της φυσαλίδας*

  Σύγκρινε ζευγάρια διαδοχικών στοιχείων, από κάτω προς τα επάνω, και όταν βρίσκεις δύο που δεν είναι στη σωστή σειρά, αντιμετάθεσέ τα. Μετά από το πρώτο πέρασμα, πρώτο στοιχείο θα είναι το μικρότερο όλων. Κάνε και δεύτερο πέρασμα, για τα στοιχεία από το δεύτερο και μετά, οπότε έτσι θα έλθει και το δεύτερο κατά σειρά στη θέση του. Μετά από $n-1$ περάσματα συνολικά, ο πίνακας θα έχει ταξινομηθεί.

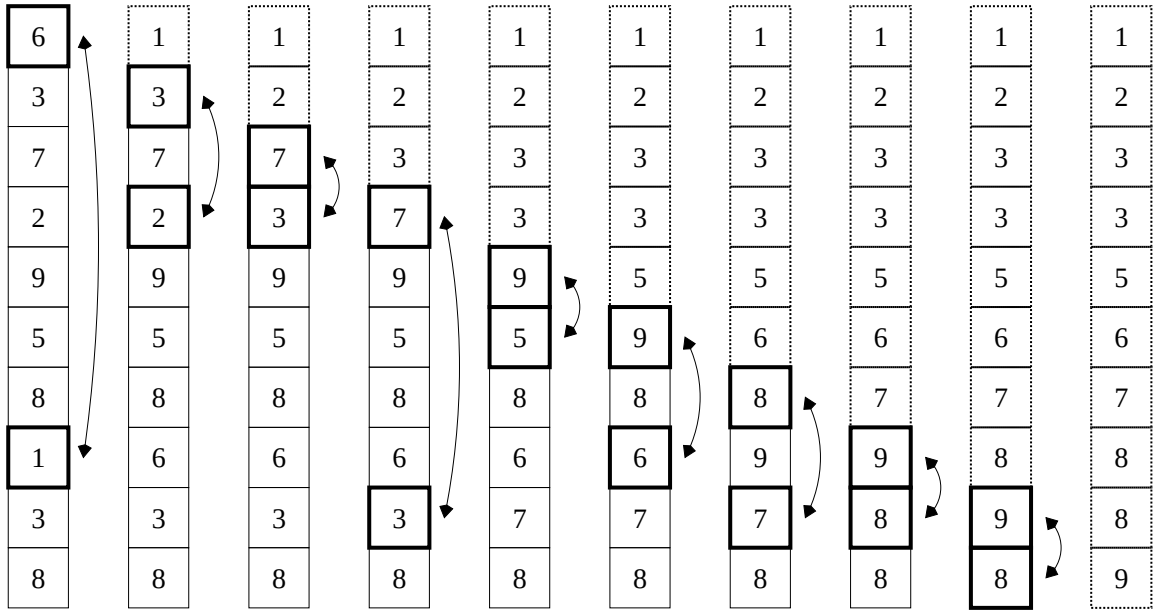


πολυπλοκότητα $O(n^2)$

Για $i = 1, 2, \ldots, n-1$
  Για $j = n-1, n-2, \ldots, i$
    Αν $x_{j-1} > x_j$ **τότε**
      Αντιμετάθεσε x_{j-1} και x_j

- *Η μέθοδος της επιλογής*

  Βρες το μικρότερο από όλα τα στοιχεία και αντιμετάθεσέ το με το πρώτο. Βρες το μικρότερο από τα υπόλοιπα και αντιμετάθεσέ το με το δεύτερο. Μετά από $n-1$ επιλογές και αντιμεταθέσεις συνολικά, ο πίνακας θα έχει ταξινομηθεί.

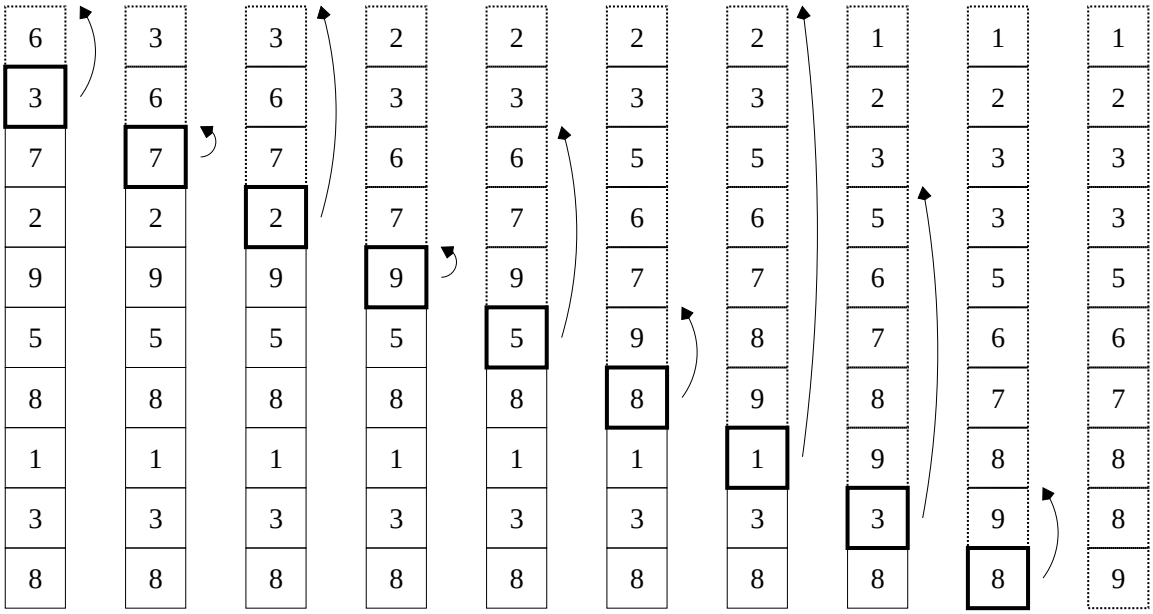


πολυπλοκότητα $O(n^2)$

Για $i = 1, 2, \ldots, n-1$
  $min = i-1$
  Για $j = i, i+1, \ldots, n-1$
    Αν $x_j < x_{min}$ **τότε**
      $min = j$
  Αντιμετάθεσε x_{i-1} και x_{min}

- *Η μέθοδος της εισαγωγής*

  Τοποθέτησε το δεύτερο στοιχείο πριν ή μετά το πρώτο ώστε να είναι στη σωστή σειρά. Τοποθέτησε το τρίτο στη σωστή θέση στα ήδη ταξινομημένα πρώτο και δεύτερο. Μετά τοποθέτησε το τέταρτο στα τρία πρώτα. Μετά από $n-1$ εισαγωγές συνολικά, ο πίνακας θα έχει ταξινομηθεί.

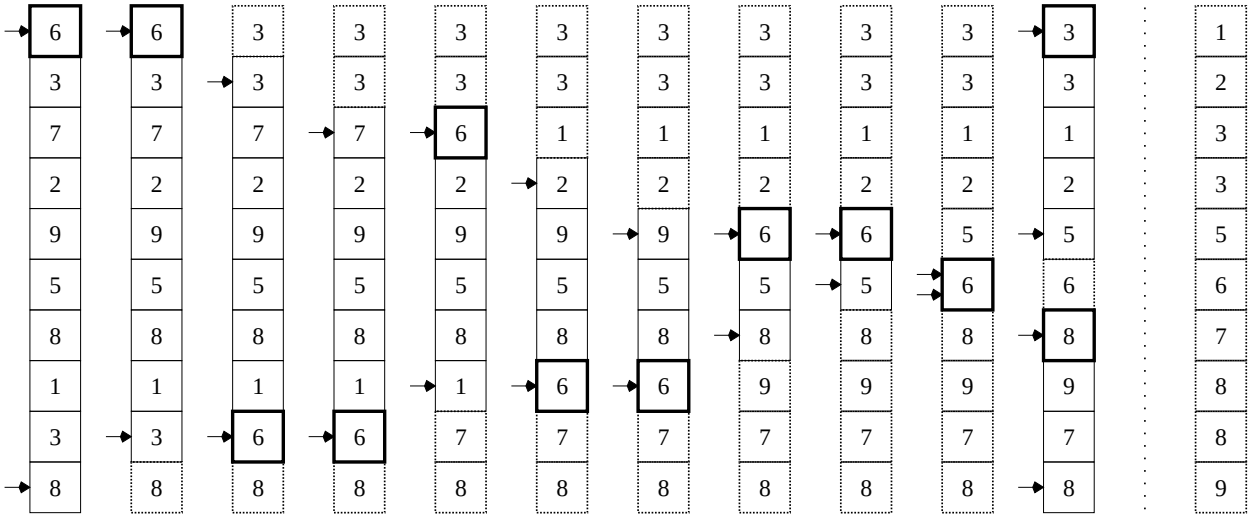


πολυπλοκότητα $O(n^2)$

Για $i = 1, 2, \ldots, n-1$
  $j = i-1$
  Ενόσω $j \geq 0$ **και** $x_j > x_{j+1}$
    Αντιμετάθεσε x_j και x_{j+1}
    $j = j-1$

- *Η γρήγορη μέθοδος*

  Με βάση το πρώτο στοιχείο σαν οδηγό, χώρισε τον πίνακα σε δύο άλλους, έναν που περιέχει στοιχεία μικρότερα ή ίσα με τον οδηγό και έναν με μεγαλύτερα ή ίσα. Ταξινόμησε τους δύο αυτούς πίνακες με την ίδια μέθοδο αναδρομικά (εφ΄ όσον έχουν τουλάχιστον δύο στοιχεία), οπότε το τελικό αποτέλεσμα είναι η παράθεση των ταξινομημένων αυτών πινάκων με ενδιάμεση παρεμβολή του οδηγού στοιχείου.



μέση πολυπλοκότητα $O(n \log n)$

Συνάρτηση $qs(x,\, up,\, down)$
  $start = up$
  $end = down$
  Ενόσω $up < down$
    Ενόσω $x_{down} \geq x_{up}$ **και** $up < down$
      $down = down{-}1$
    Αν $up \neq down$ **τότε**
      Αντιμετάθεσε x_{up} και x_{down}
      $up = up{+}1$
    Ενόσω $x_{up} \leq x_{down}$ **και** $up < down$
      $up = up{+}1$
    Αν $up \neq down$ **τότε**
      Αντιμετάθεσε x_{up} και x_{down}
      $down = down{-}1$
  Αν $start < up{-}1$ **τότε**
    Κάλεσε $qs(x,\, start,\, up{-}1)$
  Αν $end > down{+}1$ **τότε**
    Κάλεσε $qs(x,\, down{+}1,\, end)$

Για να ταξινομήσουμε τον πίνακα x που έχει n στοιχεία, θα πρέπει να καλέσουμε $qs(x,\, 0,\, n{-}1)$.

- Άλλες μέθοδοι ταξινόμησης
  - Η μέθοδος του σωρού, $O(n \log n)$
  - Η μέθοδος της συγχώνευσης, $O(n \log n)$
  - Η μέθοδος του Shell, $O(n^2)$ (σελ. 94 [KR])

Μέθοδοι ταξινόμησης

```
/* File: sorting.c */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void bubblesort(int, double *);
void selectsort(int, double *);
void insertsort(int, double *);
void quicksort(int, double *);
void quicksort_body(double *, int, int);
void swapd(double *, double *);

int main(int argc, char *argv[])
{ char method = 'b', *name; /* Default sorting method is bubblesort */
  int i, n = 10;                        /* Default array size is 10 */
  long seed;
  double *x, sttime, endtime;
  void (*fun)(int, double *);        /* Pointer to sorting function */
  seed = time(NULL); /* Get current time, in case seed is not given */
  if (argc > 1)                /* First character of first argument */
    method = *argv[1];       /* denotes the employed sorting method */
  if (argc > 2)
    n = atoi(argv[2]);     /* Second argument is number of elements */
  if (argc > 3)                         /* Third argument is seed for */
    seed = atoi(argv[3]);                  /* random number generator */
  switch(method) {        /* Prepare calling the appropriate method */
    case 'b':
      fun = bubblesort; name = "bubblesort"; break;
    case 's':
      fun = selectsort; name = "selectsort"; break;
    case 'i':
      fun = insertsort; name = "insertsort"; break;
    case 'q':
      fun = quicksort; name = "quicksort"; break;
    default:
      printf("Sorry, no such method\n");
      return 1;
  }                                  /* Allocate memory for the array */
  if ((x = malloc(n * sizeof(double))) == NULL) {
    printf("Sorry, not enough memory\n");
    return 1; }
```

```
  srand((unsigned int) seed); /* Initialize random number generator */
  for (i=0 ; i < n ; i++) /* Generate double floating point numbers */
    x[i] = ((double) rand())/RAND_MAX;
  printf("Random numbers\n");
  for (i=0 ; i < n ; i++) {
    printf("%6.4f ", x[i]);                        /* Print them out */
    if (i%10 == 9)                                   /* 10 in a line */
      printf("\n"); }
  printf("\n");
  printf("Sorting by %s\n", name);
  sttime = ((double) clock())/CLOCKS_PER_SEC;       /* Get CPU time */
                                /* consumed since start of program */
  (*fun)(n, x);                                  /* Call sorting method */
  endtime = ((double) clock())/CLOCKS_PER_SEC;    /* Again CPU time */
                             /* Difference endtime-sttime should be */
                                   /* CPU time consumed for sorting */
  for (i=0 ; i < n ; i++) {
    printf("%6.4f ", x[i]);                /* Print out sorted array */
    if (i%10 == 9)
      printf("\n"); }
  printf("\n");             /* Print out CPU time needed for sorting */
  printf("Time: %.2f secs\n", endtime-sttime);
  free(x); return 0;
}

void bubblesort(int n, double *x)
{ int i, j;
  for (i=1 ; i <= n-1 ; i++)          /* Bring appropriate element, */
                                 /* that is the bubble, to place i-1 */
    for (j=n-1 ; j >= i ; j--)
      if (x[j-1] > x[j])     /* Compare pairwise from bottom to top */
        swapd(&x[j-1], &x[j]);                /* and swap if needed */
}

void selectsort(int n, double *x)
{ int i, j, min;
  for (i=1 ; i <= n-1 ; i++) {
    min = i-1;             /* Let current minimum be the i-1 element */
    for (j=i ; j <= n-1 ; j++)
      if (x[j] < x[min])  /* Check if any element after i-1 is less */
        min = j; /* than so far minimum and make it the new minimum */
    swapd(&x[i-1], &x[min]); } /* Exchange minimum with i-1 element */
}
```

```
void insertsort(int n, double *x)
{ int i, j;
  for (i=1 ; i <= n-1 ; i++) {      /* Insert element at place i in */
                       /* its correct position from places 0 to i-1 */
    j = i-1;
    while (j >= 0 && x[j] > x[j+1]) {/* Move repeatedly the element */
      swapd(&x[j], &x[j+1]);                    /* until it reaches */
      j--; } }                               /* its correct position */
}

void quicksort(int n, double *x)
{ quicksort_body(x, 0, n-1);    /* Call recursive quicksort to sort */
}          /* elements of the array from position 0 to position n-1 */

void quicksort_body(double *x, int up, int down)
{ int start, end;
  start = up;              /* Save start position of small elements */
  end = down;                /* Save end position of large elements */
  while (up < down) {            /* Pivot element is at up position */
    while (x[down] >= x[up] && up < down)      /* Let down elements */
      down--;              /* larger than pivot stay where they are */
    if (up != down) {                    /* If pivot is not reached */
      swapd(&x[up], &x[down]);   /* echange it with smaller element */
      up++;     /* Pivot is at down position, move up a bit further */
    }
    while (x[up] <= x[down] && up < down)        /* Let up elements */
    up++;                 /* smaller than pivot stay where they are */
    if (up != down) {                    /* If pivot is not reached */
      swapd(&x[up], &x[down]);   /* exchange it with larger element */
      down--;   /* Pivot is at up position, move down a bit further */
    } }       /* Now up = down is the position of the pivot element */
  if (start < up-1) /* Is there at least one element left of pivot? */
    quicksort_body(x, start, up-1); /* Quick(sort) smaller elements */
  if (end > down+1)/* Is there at least one element right of pivot? */
    quicksort_body(x, down+1, end);  /* Quick(sort) larger elements */
}

void swapd(double *a, double *b)       /* Just exchange two doubles */
{ double temp;
  temp = *a;
  *a = *b;
  *b = temp; }
```

```
% gcc -o sorting sorting.c
% ./sorting b 30
Random numbers
0.0042 0.5007 0.7934 0.6552 0.1657 0.4608 0.4483 0.8092 0.9619 0.8410
0.9418 0.2471 0.0817 0.5173 0.8151 0.7938 0.7822 0.3956 0.0024 0.5245
0.4452 0.0439 0.3564 0.3848 0.2205 0.4797 0.9918 0.0122 0.5947 0.0798

Sorting by bubblesort
0.0024 0.0042 0.0122 0.0439 0.0798 0.0817 0.1657 0.2205 0.2471 0.3564
0.3848 0.3956 0.4452 0.4483 0.4608 0.4797 0.5007 0.5173 0.5245 0.5947
0.6552 0.7822 0.7934 0.7938 0.8092 0.8151 0.8410 0.9418 0.9619 0.9918

Time: 0.00 secs
% ./sorting s 30
Random numbers
0.2815 0.9125 0.8701 0.8467 0.9830 0.5640 0.6997 0.2883 0.4582 0.2020
0.1361 0.5284 0.1718 0.9225 0.6132 0.7087 0.3395 0.3136 0.0601 0.3492
0.3944 0.0827 0.7881 0.3528 0.5681 0.2434 0.4778 0.8673 0.9319 0.3323

Sorting by selectsort
0.0601 0.0827 0.1361 0.1718 0.2020 0.2434 0.2815 0.2883 0.3136 0.3323
0.3395 0.3492 0.3528 0.3944 0.4582 0.4778 0.5284 0.5640 0.5681 0.6132
0.6997 0.7087 0.7881 0.8467 0.8673 0.8701 0.9125 0.9225 0.9319 0.9830

Time: 0.00 secs
% ./sorting i 30
Random numbers
0.3092 0.9536 0.8778 0.5658 0.6647 0.8744 0.1248 0.5362 0.7079 0.9381
0.5555 0.6565 0.3809 0.0630 0.2930 0.2002 0.8953 0.0054 0.6659 0.8317
0.7893 0.8866 0.6313 0.5496 0.3029 0.4197 0.6265 0.2528 0.6657 0.5576

Sorting by insertsort
0.0054 0.0630 0.1248 0.2002 0.2528 0.2930 0.3029 0.3092 0.3809 0.4197
0.5362 0.5496 0.5555 0.5576 0.5658 0.6265 0.6313 0.6565 0.6647 0.6657
0.6659 0.7079 0.7893 0.8317 0.8744 0.8778 0.8866 0.8953 0.9381 0.9536

Time: 0.00 secs
% ./sorting q 30
Random numbers
0.8508 0.5154 0.8893 0.1445 0.1872 0.8399 0.7625 0.4081 0.0823 0.5423
0.6847 0.3487 0.6944 0.2738 0.3127 0.4374 0.2289 0.5431 0.5745 0.5554
0.3817 0.5924 0.3961 0.3448 0.4051 0.1843 0.8494 0.3310 0.2662 0.3955

Sorting by quicksort
0.0823 0.1445 0.1843 0.1872 0.2289 0.2662 0.2738 0.3127 0.3310 0.3448
0.3487 0.3817 0.3955 0.3961 0.4051 0.4081 0.4374 0.5154 0.5423 0.5431
0.5554 0.5745 0.5924 0.6847 0.6944 0.7625 0.8399 0.8494 0.8508 0.8893

Time: 0.00 secs
%
```

```
% hostname
iokasti
% ./sorting b 10000 2006 | tail -4
0.9982 0.9983 0.9985 0.9985 0.9985 0.9986 0.9987 0.9988 0.9989 0.9989
0.9990 0.9994 0.9994 0.9995 0.9995 0.9996 0.9996 0.9997 0.9998 0.9999

Time: 2.25 secs
% ./sorting b 30000 2006 | tail -4
0.9994 0.9994 0.9995 0.9995 0.9995 0.9995 0.9995 0.9996 0.9996 0.9997
0.9997 0.9997 0.9997 0.9998 0.9998 0.9999 0.9999 0.9999 0.9999 1.0000

Time: 20.47 secs
% ./sorting s 10000 2006 | tail -4
0.9982 0.9983 0.9985 0.9985 0.9985 0.9986 0.9987 0.9988 0.9989 0.9989
0.9990 0.9994 0.9994 0.9995 0.9995 0.9996 0.9996 0.9997 0.9998 0.9999

Time: 1.17 secs
% ./sorting s 30000 2006 | tail -4
0.9994 0.9994 0.9995 0.9995 0.9995 0.9995 0.9995 0.9996 0.9996 0.9997
0.9997 0.9997 0.9997 0.9998 0.9998 0.9999 0.9999 0.9999 0.9999 1.0000

Time: 11.89 secs
% ./sorting i 10000 2006 | tail -4
0.9982 0.9983 0.9985 0.9985 0.9985 0.9986 0.9987 0.9988 0.9989 0.9989
0.9990 0.9994 0.9994 0.9995 0.9995 0.9996 0.9996 0.9997 0.9998 0.9999

Time: 1.59 secs
% ./sorting i 30000 2006 | tail -4
0.9994 0.9994 0.9995 0.9995 0.9995 0.9995 0.9995 0.9996 0.9996 0.9997
0.9997 0.9997 0.9997 0.9998 0.9998 0.9999 0.9999 0.9999 0.9999 1.0000

Time: 14.23 secs
% ./sorting q 10000 2006 | tail -4
0.9982 0.9983 0.9985 0.9985 0.9985 0.9986 0.9987 0.9988 0.9989 0.9989
0.9990 0.9994 0.9994 0.9995 0.9995 0.9996 0.9996 0.9997 0.9998 0.9999

Time: 0.00 secs
% ./sorting q 30000 2006 | tail -4
0.9994 0.9994 0.9995 0.9995 0.9995 0.9995 0.9995 0.9996 0.9996 0.9997
0.9997 0.9997 0.9997 0.9998 0.9998 0.9999 0.9999 0.9999 0.9999 1.0000

Time: 0.02 secs
% ./sorting q 100000 2006 | tail -4
0.9998 0.9998 0.9998 0.9998 0.9998 0.9998 0.9999 0.9999 0.9999 0.9999
0.9999 0.9999 0.9999 0.9999 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

Time: 0.09 secs
%
```

Αναζήτηση σε πίνακες

- Πολλές φορές έχουμε ένα πίνακα από στοιχεία (αριθμούς, συμβολοσειρές, οτιδήποτε) και θέλουμε να δούμε αν ένα συγκεκριμένο στοιχείο, ίδιου τύπου με αυτά του πίνακα, ανήκει στον πίνακα. Η διαδικασία αυτή λέγεται αναζήτηση.
- Ο απλούστερος τρόπος να κάνουμε αναζήτηση σ΄ ένα πίνακα είναι με τη λεγόμενη σειριακή αναζήτηση. Διατρέχουμε τα στοιχεία του πίνακα, ένα προς ένα, μέχρι να βρούμε το στοιχείο που ψάχνουμε, αν το βρούμε τελικά.
- Αν ο πίνακας είναι μεγάλος, η σειριακή αναζήτηση μπορεί να είναι πολύ χρονοβόρα. Αν όμως ταξινομήσουμε τον πίνακα, έστω σε αύξουσα σειρά, μπορούμε να εφαρμόσουμε μία πολύ ταχύτερη μέθοδο, τη δυαδική αναζήτηση.
- Με τη δυαδική αναζήτηση, συγκρίνουμε το στοιχείο που ψάχνουμε με το μεσαίο (ή ένα από τα δύο μεσαία) στοιχείο του πίνακα. Αν συμπίπτει, σταματάμε την αναζήτηση αφού το στοιχείο βρέθηκε. Αν όχι, στην περίπτωση που το στοιχείο προηγείται του μεσαίου, ψάχνουμε να το βρούμε στο πρώτο μισό του πίνακα. Αν έπεται του μεσαίου, το ψάχνουμε στο δεύτερο μισό. Η αναζήτηση στον κατάλληλο υποπίνακα γίνεται με τον ίδιο τρόπο, συγκρίνοντας το στοιχείο με το μεσαίο του υποπίνακα και η διαδικασία συνεχίζει με αυτόν τον τρόπο, μέχρις ότου είτε να βρεθεί το στοιχείο είτε να φτάσουμε σε κενό υποπίνακα.
- Ποιες είναι οι πολυπλοκότητες των μεθόδων; [α΄]

[α΄] $O(n)$ και $O(\log n)$, για τη σειριακή και τη δυαδική αναζήτηση, αντίστοιχα.

Μέθοδοι αναζήτησης

```
/* File: searching.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

void heapsort(int, char **);
void heapify(char **, int, int);
int seqsearch(char *, int, char **);
int binsearch(char *, int, char **);
void swapwords(char **, char **);

int main(int argc, char *argv[])
{ int k = 0;                    /* Counter of words that will be read */
  int nmax = 1000;       /* Default maximum number of words to read */
  double sttime, endtime;
  char search = 's';                /* Default is sequential search */
  char **words, *arg, buf[81];
  while (--argc) {
    arg = *++argv;
    if (!strcmp(arg, "-max")) {               /* Get maximum number */
      if (argc > 1 && --argc)                   /* of words to read */
        nmax = atoi(*++argv); }
    else if (!strcmp(arg, "-seq"))      /* Select sequential search */
      search = 's';
    else if (!strcmp(arg, "-bin"))          /* Select binary search */
      search = 'b';
    else if (!strcmp(arg, "-words")) {  /* Give words to search for */
      argc--;
      break; } }
  argv++;    /* Allocate memory to store addresses of words to read */
  if ((words = malloc(nmax * sizeof(char *))) == NULL) {
    fprintf(stderr, "Not enough memory\n");
    return 1; }
```

```
  while (k < nmax && scanf("%80s", buf) != EOF) {
              /* Read words until EOF or maximum number reached and */
                                  /* allocate memory to store them */
    if ((words[k] = malloc((strlen(buf)+1) * sizeof(char))) == NULL) {
      fprintf(stderr, "Not enough memory\n");
      return 2; }
    strcpy(words[k++], buf); }                    /* Store word read */
  if (search == 'b')                    /* If binary search selected */
    heapsort(k, words);       /* sort words via the heapsort method */
  sttime = ((double) clock())/CLOCKS_PER_SEC;  /* Search start time */
  while (argc--) {
    arg = *argv++;
    switch (search) {
      case 's':                         /* The sequential search case */
        printf("%sfound %s\n",
          seqsearch(arg, k, words) ? "    " : "not ", arg);
        break;
      case 'b':                              /* The binary search case */
        printf("%sfound %s\n",
          binsearch(arg, k, words) ? "    " : "not ", arg);
        break; } }
  endtime = ((double) clock())/CLOCKS_PER_SEC;   /* Search end time */
  printf("Searching time is %.2f seconds\n", endtime-sttime);
  return 0; }

void heapsort(int n, char **x)
{ int i;
  for (i=(n/2)-1 ; i >= 0 ; i--)       /* Transform array to a heap */
/* A heap is an implicit binary tree where each node is not smaller */
    heapify(x, i, n-1);            /* than its immediate successors */
  for (i=n-1 ; i >= 1 ; i--) {/* Move heap root to bottom rightmost */
    swapwords(&x[0], &x[i]);    /* position not already settled and */
    heapify(x, 0, i-1); }/*transform to a heap the rest of the tree */
}
```

```
void heapify(char **x, int root, int bottom)
{ int maxchild; /* Transform to a heap the subtree starting at root */
                                          /* up to element bottom */
  while (2*root < bottom) {         /* Do we have still work to do? */
    if (2*root+1 == bottom)    /* If left child is last to consider */
      maxchild = 2*root+1;             /* this is the maximum child */
    else if (strcmp(x[2*root+1], x[2*root+2]) > 0)     /* Otherwise */
      maxchild = 2*root+1;           /* select maximum between left */
    else
      maxchild = 2*root+2;               /* and right child of root */
    if (strcmp(x[maxchild], x[root]) > 0) {/* Compare maximum child */
      swapwords(&x[maxchild], &x[root]);      /* with root and swap */
      root = maxchild; } /* accordingly, defining also the new root */
    else
      break; } }                              /* OK, we made our heap */

int seqsearch(char *w, int n, char **x)
{ int i;
  for (i=0 ; i < n ; i++)
    if (!strcmp(w, x[i]))       /* So simple, what to comment here! */
      return 1;
  return 0; }

int binsearch(char *w, int n, char **x)
{ int cond, low, high, mid;
  low = 0;                                /* Lower limit for searching */
  high = n-1;                             /* Upper limit for searching */
  while (low <= high) {          /* Do we have space for searching? */
    mid = (low+high)/2;         /* Medium element of search interval */
                          /* to compare with word we are looking for */
    if ((cond = strcmp(w, x[mid])) < 0)   /* Compare medium to word */
      high = mid-1;       /* Not found, word might be at first half */
    else if (cond > 0)
      low = mid+1;       /* Not found, word might be at second half */
    else return 1; }                                  /* We found it! */
  return 0; }                               /* Sorry, word not found */

void swapwords(char **w1, char **w2)
{ char *temp;  /* The well-known swap function for the strings case */
  temp = *w1;
  *w1 = *w2;
  *w2 = temp; }
```

```
% gcc -o searching searching.c
% cat test_words.txt
Hello there! How are you?
Hello!!! I am fine. What about you? Are you OK?
Yes, I am fine. Thank you.
% ./searching -seq -max 50 -words you word fine I < test_words.txt
    found you
not found word
not found fine
    found I
Searching time is 0.00 seconds
% ./searching -bin -max 50 -words you word fine I < test_words.txt
    found you
not found word
not found fine
    found I
Searching time is 0.00 seconds
% ./searching -bin -max 12 -words you word fine I < test_words.txt
not found you
not found word
not found fine
    found I
Searching time is 0.00 seconds
% wc -w /usr/dict/words
   25143 /usr/dict/words
% ./searching -seq -max 26000 \
?             -words `cat KRExcerpt.txt` < /usr/dict/words | tail
    found well
    found to
    found write
    found major
not found programs
    found in
    found many
    found different
not found domains.
Searching time is 0.06 seconds
% ./searching -bin -max 26000 \
?             -words `cat KRExcerpt.txt` < /usr/dict/words | tail
    found well
    found to
    found write
    found major
not found programs
    found in
    found many
    found different
not found domains.
Searching time is 0.00 seconds
%
```

```
% hostname
iokasti
% wc -w large_file.txt
  263901 large_file.txt
% ./searching -seq -max 300000 \
?             -words `cat KRExcerpt.txt` < large_file.txt | tail -6
    found programs
    found in
    found many
    found different
not found domains.
Searching time is 0.23 seconds
% ./searching -bin -max 300000 \
?             -words `cat KRExcerpt.txt` < large_file.txt | tail -6
    found programs
    found in
    found many
    found different
not found domains.
Searching time is 0.00 seconds
% ./searching -bin -max 300000 \
?             -words `cat /usr/dict/words` < large_file.txt \
?                                          | grep 'not found' | wc -l
   23606
% ./searching -bin -max 300000 \
?             -words `cat /usr/dict/words` < large_file.txt \
?                                          | grep '    found' | wc -l
    1537
% ./searching -bin -max 300000 \
?             -words `cat /usr/dict/words` < large_file.txt \
?                                          | grep 'Searching time'
Searching time is 0.04 seconds
% ./searching -seq -max 300000 \
?             -words `cat /usr/dict/words` < large_file.txt \
?                                          | grep 'Searching time'
Searching time is 468.75 seconds
%
```

Ένα πρόγραμμα C πρέπει να είναι ...

- ... **σωστό**. Προφανώς! Πρόγραμμα που δίνει λάθος αποτελέσματα πρέπει να διορθωθεί.
- ... **αποδοτικό**. Τα προγράμματα πρέπει να δίνουν τα αποτελέσματά τους μέσα σε εύλογο χρονικό διάστημα. Βέβαια, ο ορισμός του "εύλογου" ποικίλει ανάλογα με την περίπτωση και τις συνθήκες. Σε κάθε περίπτωση, θα πρέπει να προσπαθούμε το πρόγραμμά μας να έχει τη μέγιστη δυνατή απόδοση. Επίσης, θα πρέπει να ελέγχουμε τη συμπεριφορά του και για, πιθανώς ασυνήθιστα, μεγάλες εισόδους. Πάντως, ένα πρόγραμμα που θεωρητικά κάποια στιγμή θα τερματίσει, αλλά μπορεί να μην ζούμε για να το δούμε, δεν έχει πρακτική αξία.
- ... **εύρωστο**. Δεν πρέπει το πρόγραμμα να βγάζει λάθη εκτέλεσης, οποιαδήποτε και αν είναι η είσοδος που δέχτηκε. Δεν αρκεί να συμπεριφέρεται ομαλά μόνο για τις αναμενόμενες εισόδους.
- ... **τεκμηριωμένο**. Η τεκμηρίωση συνίσταται στην καλή επιλογή ονομάτων μεταβλητών, συναρτήσεων, κλπ., καθώς και στην προσθήκη σχολίων. Η τεκμηρίωση πρέπει να είναι τόση όση χρειάζεται για να κάνει το πρόγραμμα κατανοητό. Ούτε λιγότερη, αλλά ούτε και περισσότερη.
- ... **ευανάγνωστο**. Πρέπει να έχει συνεπή στοίχιση, να αποφεύγονται γραμμές με περισσότερους από 80 χαρακτήρες, να υπάρχουν κενά γύρω από τελεστές, όπου αυτό βελτιώνει την αναγνωσιμότητα.

Συχνά προγραμματιστικά λάθη στην C

- Να κάνουμε διαίρεση μεταξύ ακεραίων, ευελπιστώντας ότι θα πάρουμε σαν αποτέλεσμα έναν αριθμό κινητής υποδιαστολής, ενώ στην πραγματικότητα παίρνουμε το ακέραιο πηλίκο της διαίρεσης.
- Να νομίσουμε ότι σε μία ακέραια μεταβλητή μπορούμε να φυλάξουμε τιμή οσοδήποτε μεγάλη, ή ότι σε μία μεταβλητή κινητής υποδιαστολής μπορούμε να έχουμε όση ακρίβεια θέλουμε.
- Να κάνουμε λάθος στην οριακή συνθήκη τερματισμού μίας δομής επανάληψης.
- Να χρησιμοποιήσουμε για τελεστή σύγκρισης μέσα σε συνθήκη το =, αντί για το ==.
- Να βάλουμε μετά από μία `for` ή μία `while` ένα `;` πριν την εντολή ή το μπλοκ εντολών που αποτελούν το σώμα τους.
- Να ξεχάσουμε ότι τα στοιχεία ενός πίνακα με διάσταση `N` αριθμούνται από το `0` έως το `N-1`, και όχι από το `1` έως το `N`.
- Να χειριστούμε απρόσεκτα τους δείκτες, για παράδειγμα να δοκιμάσουμε να γράψουμε εκεί που πιστεύουμε ότι δείχνει ένας δείκτης, χωρίς να έχουμε κάνει την απαιτούμενη δέσμευση μνήμης.
- Να δεσμεύουμε δυναμικά μνήμη με επαναληπτικό τρόπο και να μην την αποδεσμεύουμε όταν δεν την χρειαζόμαστε.

- Να βάλουμε `;` στο τέλος μίας οδηγίας `#define`.
- Να ξεχάσουμε να βάλουμε παρενθέσεις γύρω από το κείμενο αντικατάστασης μίας μακροεντολής, που ορίζουμε με `#define`, και όπου αλλού χρειάζεται μέσα σ΄ αυτό.
- Να παρεξηγήσουμε τις προτεραιότητες των τελεστών, για παράδειγμα να γράψουμε

  ```
  while (ch = getchar() != EOF)
  ```

  αντί για

  ```
  while ((ch = getchar()) != EOF)
  ```

- Να κάνουμε σύγχυση ανάμεσα στην έννοια της προτεραιότητας των τελεστών, που έχει να κάνει με το πού εφαρμόζεται ένας τελεστής, με τη σειρά εκτέλεσης των ενεργειών που προσδιορίζουν οι τελεστές. Για παράδειγμα, το `*p++` είναι όντως το ίδιο με το `*(p++)`, το οποίο όμως πολλοί νομίζουν, λάθος, ότι σημαίνει ότι πρώτα θα αυξηθεί ο δείκτης `p` και μετά θα προσπελάσουμε το περιεχόμενο της νέας διεύθυνσης. **ΟΧΙ**. Απλώς, ο τελεστής `++` εδώ είναι μεταθεματικός, δηλώνοντας ότι πρώτα θα προσπελασθεί το περιεχόμενο της διεύθυνσης που δείχνει ο `p` και μετά θα αυξηθεί αυτός. Οι παρενθέσεις γύρω από το `p++` δείχνουν πού εφαρμόζεται ο τελεστής `++` (στον δείκτη `p`) και όχι πότε θα εκτελεσθεί η πράξη που υποδεικνύει.

- Να χειριστούμε απρόσεκτα μεταβλητές με το ίδιο όνομα, παρεξηγώντας τις εμβέλειες καθεμιάς, και να χρησιμοποιούμε κάποια απ΄ αυτές, ενώ πραγματικά χρησιμοποιούμε κάποια άλλη.
- Να θεωρήσουμε ότι τοπικές μεταβλητές είναι αρχικοποιημένες (για παράδειγμα, σε 0), παρότι εμείς δεν έχουμε κάνει ρητά κάτι τέτοιο.
- Να διατυπώσουμε απρόσεκτα εμφωλευμένες εντολές `if`, παραλείποντας άγκιστρα { και } σε περιπτώσεις που χρειάζονται, με αποτέλεσμα η εντολή να είναι μεν συντακτικά σωστή, οπότε ο μεταγλωττιστής δεν διαμαρτύρεται, αλλά να σημαίνει κάτι άλλο από αυτό που είχαμε πρόθεση να γράψουμε.
- Να ξεχάσουμε κρίσιμα `break` στις περιπτώσεις μίας εντολής `switch`.
- Να ταυτίσουμε το `'A'` με το `"A"`.
- Να παραβλέψουμε το γεγονός ότι μία συμβολοσειρά τερματίζει με τον χαρακτήρα `'\0'`.
- Να συγκρίνουμε συμβολοσειρές με τον τελεστή ==, αντί με τη συνάρτηση `strcmp`.
- Να μην βάλουμε συνθήκη τερματισμού μίας αναδρομής.
- Να καλέσουμε την `scanf` περνώντας σ΄ αυτήν τις μεταβλητές που θέλουμε να διαβάσουμε, αντί για τις διευθύνσεις τους.

Βιβλιογραφία

- Brian W. Kernighan, Dennis M. Ritchie: "*Η Γλώσσα Προγραμματισμού C*", Prentice Hall (Ελληνική μετάφραση, εκδόσεις Κλειδάριθμος), 1988.
- Herbert Schildt: "*Οδηγός της C*", 3η έκδοση, McGraw-Hill (Ελληνική μετάφραση, εκδόσεις Γκιούρδας), 2004.
- Γ. Σ. Τσελίκης, Ν. Δ. Τσελίκας: "*C: Από τη Θεωρία στην Εφαρμογή*", 2010.
- Νικόλαος Μισυρλής: "*Εισαγωγή στον Προγραμματισμό με την C*", 2003.
- Νίκος Μ. Χατζηγιαννάκης: "*Η Γλώσσα C σε Βάθος*", Εκδόσεις Κλειδάριθμος, 2009.
- Eric S. Roberts: "*Η Τέχνη και Επιστήμη της C*", Addison-Wesley (Ελληνική μετάφραση, εκδόσεις Κλειδάριθμος), 2004.
- Brian W. Kernighan, Rob Pike: "*The Practice of Programming*", Addison-Wesley, 1999.
- Jon Bentley: "*Programming Pearls*", 2nd edition, Addison-Wesley, 2000.
- Steven S. Skiena: "*The Algorithm Design Manual*", Springer-Verlag, 1998.
- H. M. Deitel, P. J. Deitel: "*C: How to Program*", 3rd edition, Prentice Hall, 2001.
- Νικόλαος Θ. Αποστολάτος: "*Προγραμματισμός – Εφαρμογές*", 1995.
- Διομήδης Σπινέλλης: "*Ανάγνωση Κώδικα – Η Προοπτική του Ανοικτού Λογισμικού*", εκδόσεις Κλειδάριθμος, 2005.
- Brian W. Kernighan, Rob Pike: "*Το Περιβάλλον Προγραμματισμού Unix*", Prentice Hall (Ελληνική μετάφραση, εκδόσεις Κλειδάριθμος), 1989.
- Donald E. Knuth: "*The Art of Computer Programming: Sorting and Searching (Vol. 3)*", 2nd edition, Addison-Wesley, 1998.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: "*Introduction to Algorithms*", 2nd edition, The MIT Press, 2001.

Ευχαριστίες ...

- ... στην Επίκουρη Καθηγήτρια *Ιζαμπώ Καράλη* για τη μετάδοση πολύτιμης εμπειρίας, ιδεών και υλικού από το μάθημα του Αντικειμενοστραφούς Προγραμματισμού που διδάσκει στο Τμήμα μας από το ακαδημαϊκό έτος 2002–03.
- ... επίσης στην *Ιζαμπώ* και στους απόφοιτους του Τμήματος (προπτυχιακού και μεταπτυχιακού κύκλου) *Στέφανο Σταμάτη* και *Νίκο Ποθητό* που διάβαζαν με απεριόριστη υπομονή τα πρωτόλεια αυτών των διαφανειών/σημειώσεων, ενόσω γραφόντουσαν, και υπέδειξαν ένα πλήθος από λάθη και παραλείψεις που είχαν, κάνοντας ταυτόχρονα και προτάσεις για τη διόρθωσή τους.

Παναγιώτης Σταματόπουλος