Άσκηση 1 – Λύσεις

Πρόβλημα 1 [15 μονάδες] *Χρησιμοποιήστε το κεντρικό θεώρημα για να προσδιορίσετε αυστηρά ασυμπτωτικά φράγματα για τις ακόλουθες αναδρομικές σχέσεις:*

1. $T(n) = 4T(n/2) + n$.
2. $T(n) = 4T(n/2) + n^2$.
3. $T(n) = 4T(n/2) + n^3$.

Λύση:

(α′) Στην περίπτωση αυτή έχουμε $a = 4$ και $b = 2$, οπότε $\log_b a = 2$. Επίσης έχουμε:

$$f(n) = O(n^{3/2}) = O(n^{\log_b a - \epsilon}),$$

για $\epsilon = \frac{1}{2} > 0$, οπότε σύμφωνα με το κεντρικό θεώρημα έχουμε ότι $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$.

(β′) Στην περίπτωση αυτή έχουμε $a = 4$, $b = 2$, οπότε $f(n) = \Theta(n^2) = \Theta(n^{\log_b a})$. Συνεπώς $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(n^2 \lg n)$.

(γ′) Στην περίπτωση αυτή έχουμε $a = 4$, $b = 2$, και

$$f(n) = n^3 = \Omega(n^{5/2}) = \Omega(n^{\log_b a + \epsilon}),$$

για $\epsilon = \frac{1}{2} > 0$. Επίσης

$$af\left(\frac{n}{b}\right) = 4f\left(\frac{n}{2}\right) = 4\left(\frac{n}{2}\right)^3 = 4\frac{n^3}{8} = \frac{n^3}{2} = \frac{1}{2}n^3 \leq cf(n),$$

για οποιοδήποτε $c \in [\frac{1}{2}, 1)$. Συνεπώς από το κεντρικό θεώρημα προκύπτει ότι $T(n) = \Theta(f(n)) = \Theta(n^3)$.

□

Πρόβλημα 2 [20 μονάδες] *Μπορεί να εφαρμοστεί στην αναδρομική σχέση*

$$T(n) = 4T(n/2) + n^2 \lg n$$

η κεντρική μέθοδος; Αιτιολογήστε την απάντησή σας. Προσδιορίστε ένα ασυμπτωτικό άνω φράγμα για τη σχέση αυτή.

Λύση: Στο πρόβλημα αυτό έχουμε $a = 4$, $b = 2$ και $\log_b a = 2$. Στην περίπτωση αυτή δεν μπορεί να εφαρμοστεί το κεντικό θεώρημα γιατί δεν ισχύει καμία από τις σχέσεις $f(n) = O(n^{\log_b a - \epsilon})$, $f(n) = \Theta(n^{\log_b a})$ και $f(n) = \Omega(n^{\log_b a + \epsilon})$ για οποιαδήποτε δυνατή επιλογή του ϵ. Πιο συγκεκριμένα ισχύει ότι $f(n) = \omega(n^{\log_b a})$, και $f(n) = o(n^{\log_b a + \epsilon})$, για κάθε $\epsilon > 0$.

Θα δείξουμε ότι $T(n) = O(n^2 \lg^2 n)$, και ειδικότερα ότι $T(n) \leq cn^2 \lg^2 n)$, για κάποιο c κατάλληλα επιλεγμένο. Αντικαθιστώντας έχουμε

$$\begin{aligned} T(n) &= 4T(n/2) + n^2 \lg n \\ &\leq 4c \left(\frac{n}{2}\right)^2 \lg^2 \left(\frac{n}{2}\right) + n^2 \lg n \\ &= 4c\frac{n^2}{4}(\lg n - 1)^2 + n^2 \lg n \\ &= cn^2(\lg^2 n - 2\lg n + 1) + n^2 \lg n \\ &= cn^2 \lg^2 n - 2cn^2 \lg n + cn^2 + n^2 \lg n \\ &= cn^2 \lg^2 n - [(2c-1)n^2 \lg n - cn^2] \end{aligned}$$

Παρατηρούμε ότι για $c \geq 1$, έχουμε $\frac{c}{2c-1} \leq 1$, ενώ για $n \geq 2$, έχουμε $\lg n \geq 1$. Συνεπώς για $c \geq 1$ και $n \geq 2$, έχουμε:

$$\lg n \geq 1 \geq \frac{c}{2c-1} \quad \Rightarrow \quad (2c-1)\lg n - c \geq 0 \quad \Rightarrow \quad (2c-1)n^2 \lg n - cn^2 \geq 0.$$

Συνεπώς για κάθε $n \geq 2$, $T(n) \leq cn^2 \lg^2 n$, δηλαδή $T(n) = O(n^2 \lg^2 n)$. □

Πρόβλημα 3 [10 μονάδες] *Με χρήση των ορισμών δείξτε ότι:* $f(n) + o(f(n)) = \Theta(f(n))$.

Λύση: Στην πραγματικότητα μας ζητείται να δείξουμε ότι $f(n) + g(n) = \Theta(f(n))$, όπου $g(n) = o(f(n))$.

Εφ' όσον $g(n) = o(f(n))$, για κάθε $c > 0$ υπάρχει $n_0 = n_o(c) > 0$ τέτοιο ώστε για κάθε $n \geq n_0$, έχουμε $0 \leq g(n) \leq cf(n)$. Έστω ότι $c = 1$. Τότε υπάρχει κάποιο $n_0 > 0$, τέτοιο ώστε $0 \leq g(n) \leq f(n)$. Τότε όμως θα έχουμε, για κάθε $n \geq n_0$, και ότι

$$0 \leq f(n) \leq f(n) + g(n) \leq 2f(n).$$

Δηλαδή υπάρχουν σταθερές $c_1 = 1$, $c_2 = 2$ και $n_0 > 0$, τέτοιες ώστε για κάθε $n \geq n_0$ να ισχύει:

$$0 \leq c_1 f(n) \leq f(n) + g(n) \leq c_2 f(n),$$

δηλαδή $f(n) + g(n) = \Theta(f(n))$. □

Πρόβλημα 4 [15 μονάδες] *Θεωρήστε την αναδρομική σχέση* $T(n) = T(\lg n) + 1$. *Δείξτε ότι* $T(n) = O(\lg^\star n)$.

Λύση: Θα δείξουμε ότι $T(n) \leq c \lg^\star n$, για κατάλληλα επιλεγμένο $c > 0$. Αντικαθιστώντας στη δοσμένη αναδρομική σχέση προκύπτει ότι:

$$T(n) = T(\lg n) + 1 \leq c \lg^\star \lg n + 1,$$

η οποία για $c \geq 1$ γίνεται:

$$T(n) \leq c(\lg^\star \lg n + 1).$$

Θα δείξουμε ότι για κάθε $n \geq 3$, ισχύει $\lg^\star \lg n + 1 \leq \lg^\star n$.

Ας θεωρήσουμε την ακολουθία αριθμών α_k, $k \geq 1$, που ορίζεται σύμφωνα με τον παρακάτω τύπο:

$$\alpha_k = \begin{cases} 2, & \text{αν } k = 1 \\ 2^{\alpha_{k-1}}, & \text{αν } k > 1 \end{cases}.$$

Παρατηρούμε ότι για κάθε $k \geq 1$, ισχύει $\lg^\star \alpha_k = k$, ενώ επίσης για κάθε $k \geq 1$ ισχύει ότι $\lg \alpha_{k+1} = \alpha_k$. Τέλος, παρατηρούμε ότι για κάθε n, με $\alpha_k < n \leq \alpha_{k+1}$, έχουμε $\lg^\star n = k + 1$.

Ας θεωρήσουμε τώρα κάποιο $n \geq 3$. Τότε υπάρχει $k \geq 1$, τέτοιο ώστε $\alpha_k < n \leq \alpha_{k+1}$, ενώ σύμφωνα με τις παρατηρήσεις της παραπάνω παραγράφου, έχουμε επίσης ότι:

$$\lg^\star \lg n + 1 \leq \lg^\star \lg \alpha_{k+1} + 1 = \lg^\star \alpha_k + 1 = k + 1 = \lg^\star n.$$

Συνεπώς για κάθε $c \geq 1$ και $n \geq 3$, έχουμε ότι

$$T(n) \leq c(\lg^\star \lg n + 1) \leq c \lg^\star n,$$

δηλαδή $T(n) = O(\lg^\star n)$. □

Πρόβλημα 5 [40 μονάδες] *Μία ακολουθία αντικειμένων $a_1, a_2, \ldots, a_n$ θα ονομάζεται κυρτή αν υπάρχει k, με $1 \leq k \leq n$, τέτοιο ώστε για κάθε $1 \leq i \leq k-1$ να έχουμε $a_i \geq a_{i+1}$, ενώ για κάθε $k \leq i \leq n-1$ να έχουμε $a_i \leq a_{i+1}$.*

Σχεδιάστε έναν αλγόριθμο γραμμικού χρόνου ο οποίος ταξινομεί τα στοιχεία μίας κυρτής ακολουθίας αντικειμένων σε γραμμικό χρόνο. Στη συνέχεια υλοποιήστε σε `C` τον αλγόριθμό σας στην περίπτωση που τα αντικείμενά σας είναι αριθμοί τύπου `double`. Η συνάρτησή σας πρέπει να έχει υπογραφή:

```
void dcsort(double a[], int n);
```

όπου `a` είναι ο προς ταξινόμηση πίνακας, και `n` το πλήθος των στοιχείων του `a`. Κατά την επιστροφή ο πίνακας `a` θα πρέπει να περιέχει την ταξινομημένη ακολουθία.

Τέλος, γράψτε γενικό κώδικα για τον αλγόριθμό σας (σε αναλογία με το γενικό κώδικα της `qsort` που παρέχεται από την `stdlib` της `C`). Η συνάρτησή σας πρέπει, στην περίπτωση αυτή, να έχει υπογραφή:

```
void csort(void *a, size_t n, size_t size, int (*cmp)(const void*, const void*));
```

όπου `a` είναι ο πίνακας που περιέχει (στην είσοδο) την κυρτή ακολουθία αντικειμένων, `n` το πλήθος των αντικειμένων, μεγέθους `size` το καθένα, και `cmp` δείκτης σε συνάρτηση σύγκρισης στοιχείων του πίνακα `a`. Θεωρήστε ότι η συνάρτηση `*cmp` επιστρέφει -1 αν το πρώτο στοιχείο είναι μικρότερο από το δεύτερο, 0 αν τα δύο στοιχεία είναι ίσα, και 1 αν το πρώτο στοιχείο είναι μεγαλύτερο από το δεύτερο.

Λύση: Ο αλγόριθμός μας λειτουργεί ως εξής:

1. Δημιούργησε ένα πίνακα b μεγέθους n και αντέγραψε τα στοιχεία του a στον b.
2. Βρες το ελάχιστο στοιχείο του b. Αν το ελάχιστο στοιχείο δεν είναι μοναδικό, διάλεξε εκείνο με το μεγαλύτερο δείκτη. Έστω `imin` ο δείκτης του ελάχιστου στοιχείου στην ακολουθία $b_1, \ldots, b_n$.
3. Συγχώνευσε τις ακολουθίες $b_{\texttt{imin}..1}$ και $b_{\texttt{imin+1}..n}$ βάζοντας το αποτέλεσμα της συγχώνευσης στον πίνακα a.

Το βήμα 1 του αλγορίθμου απαιτεί χρόνο $\Theta(n)$, καθώς αντιγράφω τον πίνακα a στον b. Το βήμα 2 απαιτεί χρόνο $O(n)$, καθώς πρέπει να κάνουμε στη χειρότερη περίπτωση $n-1$ συγκρίσεις για να βρούμε το ελάχιστο στοιχείο. Τέλος το βήμα 3 απαιτεί και αυτό χρόνο $\Theta(n)$, καθώς συγχωνεύουμε δύο αύξουσες ακολουθίες αριθμών συνολικού μεγέθους n. Συνεπώς ο αλγόριθμός μας τρέχει σε χρόνο $\Theta(n)$.

Παρακάτω παρατίθεται κώδικας για τις συναρτήσεις `dcsort` και `sort`, ενώ επίσης σας δίνεται και κώδικας σε C++ (συνάρτηση `convex_sort`) ο οποίος δέχεται ως όρισμα ένα random access iterator range και προαιρετικά ένα adaptable binary predicate τύπου less και εκτελεί στην ταξινόμηση.

Listing: dcsort.h

```
#ifndef EM202_DCSORT_H
#define EM202_DCSORT_H 1

#include <stdio.h>
#include <stdlib.h>


void dcsort(double a[], int n)
{
  if ( n <= 1 ) return;

  double *b = (double*)malloc(n * sizeof(double));

  if ( b == NULL ) {
    printf("Cannot allocate memory");
    exit(1);
  }

  /* find the index of the minimum element */
  int i = 1, imin = 0;
  while ( a[i] <= a[imin] && i < n ) {
    imin = i;
    ++i;
  }

  /* copy the elements of a in the index range 0..imin into the array
     b in increasing order */
  for (i = 0; i <= imin; ++i) {
    b[i] = a[imin - i];
  }

  /* copy the elements of a in the index range (imin+1)..(n-1) into the
     array b, in the corresponding positions */
  for (i = imin + 1; i < n; ++i) {
    b[i] = a[i];
  }

  /* the array b contains now two sorted lists of numbers in
     increasing order; the two sorted lists span the index ranges 0..imin
     and (imin+1)..(n-1); below we merge the two sorted lists and the
     outcome is written in array a */
  int j = 0, k = 0;
  i = imin + 1;
  while ( j <= imin && i < n ) {
    if ( b[j] <= b[i] ) {
      a[k++] = b[j++];
```

```
    } else {
      a[k++] = b[i++];
    }
  }

  /* one of the two sorted lists has been exhausted; copy the elements
     of the other sorted list into array a */
  while ( j <= imin ) { a[k++] = b[j++]; }
  while ( i < n ) { a[k++] = b[i++]; }

  /* deallocate allocated memory */
  free(b);
}


#endif /* EM202_DCSORT_H */
```

Listing: csort.h

```
#ifndef EM202_CSORT_H
#define EM202_CSORT_H 1

#include <stdio.h>
#include <stdlib.h>

/* the following function performs the assignment to[i] = from[i],
   where to[i] and from[i] are objects of size in bytes equal to 'size' */
void csort_assign(void* to, size_t i_to, void* from, size_t i_from, size_t size)
{
  char *c_to = (char*)to;
  char *c_from = (char*)from;

  size_t i;
  for (i = 0; i < size; ++i) {
    *(c_to + i_to * size + i) = *(c_from + i_from * size + i);
  }
}


void csort(void *a, size_t n, size_t size, int (*cmp)(void*,void*))
{
  if (n <= 1) return;

  void *b = malloc(n * size);

  if ( b == NULL ) {
    printf("Cannot allocate memory");
    exit(1);
  }

  /* find the index of the minimum element */
  size_t i = 1, imin = 0;
  while ( (*cmp)(a + i * size, a + imin * size) <= 0 && i < n ) {
    imin = i;
    ++i;
  }

  /* copy the elements of a in the index range 0..imin into the array
```

```
    b in increasing order */
  for (i = 0; i <= imin; ++i) {
    csort_assign(b, i, a, imin - i, size);
  }

  /* copy the elements of a in the index range (imin+1)..(n-1) into the
     array b, in the corresponding positions */
  for (i = imin + 1; i < n; ++i) {
    csort_assign(b, i, a, i, size);
  }

  /* the array b contains now two sorted lists of numbers in
     increasing order; the two sorted lists span the index ranges 0..imin
     and (imin+1)..(n-1); below we merge the two sorted lists and the
     outcome is written in array a */
  size_t j = 0, k = 0;
  i = imin + 1;
  while ( j <= imin && i < n ) {
    if ( (*cmp)(b + j * size, b + i * size) <= 0 ) {
      csort_assign(a, k, b, j, size);
      ++k, ++j;
    } else {
      csort_assign(a, k, b, i, size);
      ++k, ++i;
    }
  }

  /* one of the two sorted lists has been exhausted; copy the elements
     of the other sorted list into array a */
  while ( j <= imin ) {
    csort_assign(a, k, b, j, size);
    ++k, ++j;
  }
  while ( i < n ) {
    csort_assign(a, k, b, i, size);
    ++k, ++i;
  }

  /* deallocate allocated memory */
  free(b);
}



#endif /* EM202_CSORT_H */
```

Listing: convex_sort.hpp

```
#ifndef EM202_CONVEX_SORT_HPP
#define EM202_CONVEX_SORT_HPP 1

#include <vector>
#include <algorithm>
#include <functional>

template<typename RandomAccessIterator, class Less>
void convex_sort(const RandomAccessIterator& begin,
```

```
                const RandomAccessIterator& beyond,
                const Less& less)
{
  typedef std::iterator_traits<RandomAccessIterator>   Iterator_traits;
  typedef typename Iterator_traits::value_type         value_type;
  typedef typename Iterator_traits::difference_type    Distance;

  // the number of elements in the interator range
  Distance n = (beyond - begin);

  if ( n <= 1 ) return;

  // auxiliary storage used by the algorithm
  std::vector<value_type> aux;

  // copy the elements in the given interator range to the vector aux
  std::copy(begin, beyond, std::back_inserter(aux));

  // find the index of the minimum element in aux
  Distance i = 1, imin = 0;
  while ( !less(aux[imin], aux[i]) && i < n ) {
    imin = i;
    ++i;
  }

  // the array aux contains now two sorted lists of numbers in
  // increasing order; the two sorted lists span the index ranges imin..0
  // and (imin+1)..(n-1); below we merge the two sorted lists and the
  // outcome is written in the given random access iterator range */
  Distance j = imin, k = 0;
  i = imin + 1;
  while ( j >= 0 && i < n ) {
    if ( less(aux[j], aux[i]) ) {
      *(begin + k) = aux[j--];
    } else {
      *(begin + k) = aux[i++];
    }
    ++k;
  }

  // one of the two sorted lists has been exhausted; copy the elements
  // of the other sorted list into the given random access iterator range
  while ( j >= 0 ) {
    *(begin + k) = aux[j--];
    ++k;
  }
  while ( i < n ) {
    *(begin + k) = aux[i++];
    ++k;
  }
}



template<typename RandomAccessIterator>
void convex_sort(const RandomAccessIterator& begin,
                const RandomAccessIterator& beyond)
{
```

```
  typedef std::iterator_traits<RandomAccessIterator>    Iterator_traits;
  typedef typename Iterator_traits::value_type          value_type;

  convex_sort(begin, beyond, std::less<value_type>());
}


#endif // EM202_CONVEX_SORT_HPP
```

Τα αρχεία που περιέχουν τους σχετικούς κώδικες μπορείτε να τα κατεβάσετε χρησιμοποιώντας τα παρακάτω links:

- Συνάρτηση `dcsort`: αρχείο `dcsort.h`
- Συνάρτηση `csort`: αρχείο `csort.h`
- Συνάρτηση `convex_sort`: αρχείο `convex_sort.hpp`

Τέλος, σας δίνονται επίσης και τα αρχεία `csort.c` και `convex_sort.cpp` που χρησιμοποιούν τις παραπάνω υλοποιήσεις σε συγκεκριμένα παραδείγματα. □

Σύνολο μονάδων: 100