ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ & ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

Τμήμα Εφαρμοσμένων Μαθηματικών
Εαρινό εξάμηνο 2007 - 2008,
ΕΜ 202, Θεωρία Αλγορίθμων,
Διδάσκων Παρασκευάς Λεκέας

Τελική Εξέταση

24 - Ιουνίου - 2008

Έχετε στη διάθεσή σας 3 ώρες για να λύσετε τα προβλήματα. Το άριστα για αυτό το διαγώνισμα είναι 80 μονάδες.

Ονοματεπώνυμο : **Α.Μ. :**

Βαθμοί

1. α)
   β)

2. α)
   β)
   γ)
   δ)
   ε)

3. α)
   β)
   γ)

4. α)
   β)

Σύνολο _____

ΘΕΜΑΤΑ

1. α) **(μονάδες 8)** Χρησιμοποιείστε τη μέθοδο της αντικατάστασης για να αποδείξετε ένα αυστηρό ασυμπτωτικό κάτω φράγμα (Ω συμβολισμός) για την πιο κάτω αναδρομική εξίσωση: $T(n)=4T(n/2)+n^2$.

β) **(μονάδες 15)** Υποθέτοντας ότι T(n) σταθερά για $n \le 2$ βρείτε ασυμπτωτικά φράγματα για τις πιο κάτω αναδρομικές εξισώσεις χρησιμοποιώντας τις περιπτώσεις του βασικού θεωρήματος, **β1)** $T(n) = 2T(n/2) + n^3$, **β2)** $T(n) = 16T(n/4) + n^2$, **β3)** $T(n) = 7T(n/2) + n^2$.

Απόδειξη 1α) Από τη βασική μέθοδο έχουμε $T(n)=\Theta(n^2\lg n)$. Άρα η επαγωγική υπόθεση θα είναι $T(m) \ge cm^2\lg m$ για κάθε $m < n$. Για $m=1$, $T(1)=1 > c1^2\lg 1$ για κάθε $c>0$. Για το επαγωγικό βήμα υποθέτουμε ότι για κάθε $m<n$, $T(m) \ge cm^2\lg m$. Έχουμε: $T(n) = 4T(n/2)+n^2 \ge 4c(n/2)^2\lg(n/2)+n^2 = cn^2\lg n - cn^2\lg 2 + n^2 = cn^2\lg n + (1-c)n^2$.

Για $c < 1$, η πιο πάνω ποσότητα είναι πάντα μεγαλύτερη του $cn^2\lg n$. Άρα $T(n)=\Omega(cn^2\lg n)$

Απόδειξη 1β) $\Theta(n^3)$ διότι $a=2$, $b=2$, $f(n)=n^3$ και $n^{\lg_b a} = n^{\lg_2 2} = n$. $n^3=\Omega(n^{\log_2 2+2})$ και $a/b^k = 2/2^3 = 1/4 < 1$ έχουμε την περίπτωση 3. παρόμοια και οι άλλες περιπτώσεις: **β2)** $\Theta(n^2\lg n)$, περίπτωση 2, **β3)** $\Theta(n^{\lg 7})$, περίπτωση 1.

2. α) **(μονάδες 10)** Σας δίνονται 2 σύνολα ακεραίων S και T. Έστω $m=|S|$ και $n=|T|$ και έστω $n \ge m$. Υποθέτοντας ότι οποιαδήποτε πράξη μεταξύ δύο ακεραίων (π.χ. πρόσθεση, πολλαπλασιασμός) απαιτεί μοναδιαίο χρόνο, προτείνετε αλγόριθμο ο οποίος να βρίσκει την τομή $S \cap T$ σε χρόνο $o(n^2)$.

β) **(μονάδες 4)** Ποιά η διαφορά ενός τυχαιοκρατικού από έναν αιτιοκρατικό αλγόριθμο;

γ) **(μονάδες 4)** Ποιό το ελάχιστο και ποιό το μέγιστο πλήθος στοιχείων σε ένα σωρό ύψους h;

δ) **(μονάδες 4)** Ποιός είναι ο χρόνος εκτέλεσης της ταξινόμησης σωρού (HEAPSORT, βλ. παράρτημα) σε συστοιχία A μήκους n η οποία είναι ήδη ταξινομημένη κατά αύξουσα σειρά;

ε) **(μονάδες 7)** Ταξινομήστε τους πιο κάτω αλγορίθμους ως προς τη χρονική τους επίδοση χειρότερης περίπτωσης. Ποιοί από αυτούς είναι ασταθείς αλγόριθμοι; InsertionSort, Heapsort, QuickSort, MergeSort.

Απόδειξη 2α) Ταξινομώ κάθε σύνολο με την Merge Sort. Συγχωνεύω τα αποτελέσματα αγνοώντας όποιο στοιχείο εμφανίζεται σε ένα από τα δύο σύνολα. Ο αλγόριθμος αυτός τρέχει σε χρόνο $\Theta(n\log n)$.

Απόδειξη 2β) Βιβλίο σελ. 93

Απόδειξη 2γ) κατά μέγιστο $2^{h+1}-1$, κατ' ελάχιστο 2^h.

Απόδειξη 2δ) Είναι $\Theta(n\lg n)$, διότι η κατασκευή σωρού μεγίστου (BUILDMAX HEAP) θέλει $\Theta(n)$ χρόνο και η ταξινόμηση (HEAPSORT) συνολικά $\Theta(n\lg n)$ λόγω των n-1 κλήσεων της MAXHEAPIFY (βιβλίο σελ. 134).

Απόδειξη 2ε) Insertion ($O(n^2)$, ευσταθής), Heapsort ($O(n\lg n)$, ασταθής), Mergesort ($O(n\lg n)$, ευσταθής), Quicksort ($O(n^2)$, ασταθής).

3. α) **(μονάδες 5)** Διαθέτουμε άπειρη προμήθεια από νομίσματα αξίας $x_1, x_2, \ldots x_n$. Μπορούμε να σχηματίσουμε ρέστα αξίας v χρησιμοποιώντας το πολύ k από αυτά; Προτείνετε αλγόριθμο δυναμικού προγραμματισμού που λύνει το πιο πάνω πρόβλημα. Ποιά η πολυπλοκότητα του αλγορίθμου σας;

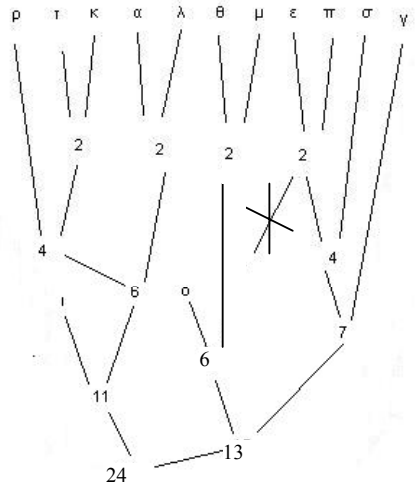
β) **(μονάδες 6)** Ποια βασικά βήματα πρέπει να ακολουθήσει κάποιος για να σχεδιάσει έναν άπληστο αλγόριθμο για ένα πρόβλημα βελτιστοποίησης;

γ) **(μονάδες 5)** Γράψτε βέλτιστο δυαδικό κώδικα Huffman για τη φράση: «ΠΡΟΣΕΓΓΙΣΤΙΚΟΙ ΑΛΓΟΡΙΘΜΟΙ». Κατασκευάστε το δυαδικό δέντρο παραγωγής του κώδικα που προτείνετε και γράψτε τη λέξη «ΑΓΡΙΜΙ» στον κώδικά σας

Απόδειξη 3α) Αν K(w) ο ελάχιστος απαιτούμενος αριθμός νομισμάτων για ρέστα αξίας w, όπου w παίρνει τιμές από 0 έως v, θα ισχύει ότι $K(w)=\min\{K(w-x_i)+1\}$, όπου το min πάνω στα x_i. Το K(v) είναι η λύση του προβλήματος μόνο αν $K(v)>k$, αλλιώς δεν υπάρχει λύση. Πολυπλοκότητα = αριθμός υποπροβλημάτων επί αριθμό επιλογών για κάθε υποπρόβλημα.

Απόδειξη 3β) βιβλίο σελ. 382

Απόδειξη 3γ) Οι συχνότητες εμφάνισης είναι Π(1), Ρ(2), Ο(4), Σ(2), Ε(1), Γ(3), Ι(5), Τ(1), Κ(1), Α(1), Λ(1), Θ(1), Μ(1) και μία περίπτωση του δέντρου μπορεί να είναι η:



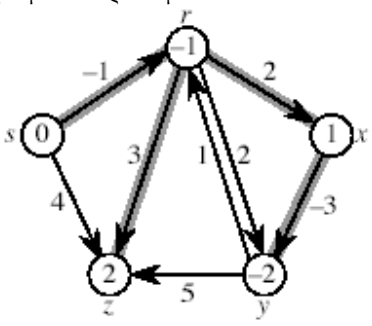
ΑΓΡΙΜΙ = 01101110100001010100

4. ***α)*** **(μονάδες 6)** Χρησιμοποιώντας τον αλγόριθμο Bellman-Ford (παράρτημα) βρείτε τις ελαφρύτατες ομοαφετηριακές διαδρομές από τον κόμβο s του πιο κάτω γράφου.

s	0	−1	0	0	4
r	0	0	2	2	3
x	0	0	0	−3	0
y	0	1	0	0	5
z	0	0	0	0	0

β) (μονάδες 6) Πότε για ένα πρόβλημα μεγιστοποίησης ένας αλγόριθμος Α ονομάζεται ρ-προσεγγιστικός;

Απόδειξη 4α) Τρέχοντας τον αλγόριθμο παίρνουμε:



Στους κύκλους φαίνονται οι ελαφρύτατες ομοαφετηριακές διαδρομές

Απόδειξη 4β) Ένας αλγόριθμος Α ονομάζεται ρ-προσεγγιστικός για ένα πρόβλημα μεγιστοποίησης, αν για κάθε είσοδο μεγέθους n το κόστος C_A της λύσης του αλγορίθμου και το κόστος της βέλτιστης λύσης C_{OPT} συνδέονται με την σχέση $C_A \leq \rho C_{OPT}$.

Παράρτημα

```
HEAPSORT(A, n)
BUILD-MAX-HEAP(A, n)
for i ← n downto 2
    do exchange A[1] ↔ A[i]
       MAX-HEAPIFY(A, 1, i − 1)
```

```
BUILD-MAX-HEAP(A, n)
for i ← ⌊n/2⌋ downto 1
    do MAX-HEAPIFY(A, i, n)
```

```
MAX-HEAPIFY(A, i, n)
l ← LEFT(i)
r ← RIGHT(i)
if l ≤ n and A[l] > A[i]
  then largest ← l
  else largest ← i
if r ≤ n and A[r] > A[largest]
  then largest ← r
if largest ≠ i
  then exchange A[i] ↔ A[largest]
       MAX-HEAPIFY(A, largest, n)
```

```
INSERTION-SORT(A)
for j ← 2 to n
    do key ← A[j]
       ▷ Insert A[j] into the sorted sequence A[1 . . j − 1].
       i ← j − 1
       while i > 0 and A[i] > key
           do A[i + 1] ← A[i]
              i ← i − 1
       A[i + 1] ← key
```

```
QUICKSORT(A, p, r)
if p < r
  then q ← PARTITION(A, p, r)
       QUICKSORT(A, p, q − 1)
       QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
x ← A[r]
i ← p − 1
for j ← p to r − 1
    do if A[j] ≤ x
         then i ← i + 1
              exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

```
MERGE-SORT(A, p, r)
if p < r
  then q ← ⌊(p + r)/2⌋
       MERGE-SORT(A, p, q)
       MERGE-SORT(A, q + 1, r)
       MERGE(A, p, q, r)
```

```
MERGE(A, p, q, r)
n1 ← q − p + 1
n2 ← r − q
create arrays L[1 . . n1 + 1] and R[1 . . n2 + 1]
for i ← 1 to n1
    do L[i] ← A[p + i − 1]
for j ← 1 to n2
    do R[j] ← A[q + j]
L[n1 + 1] ← ∞
R[n2 + 1] ← ∞
i ← 1
j ← 1
for k ← p to r
    do if L[i] ≤ R[j]
         then A[k] ← L[i]
              i ← i + 1
         else A[k] ← R[j]
              j ← j + 1
```

```
BELLMAN-FORD(V, E, w, s)
INIT-SINGLE-SOURCE(V, s)
for i ← 1 to |V| − 1
    do for each edge (u, v) ∈ E
         do RELAX(u, v, w)
for each edge (u, v) ∈ E
    do if d[v] > d[u] + w(u, v)
         then return FALSE
return TRUE
```