

# Αλγόριθμοι και Πολυπλοκότητα

## 1η Σειρά Γραπτών Ασκήσεων

Γιαννέλος Γιάννης  
ΑΜ:03108088

8 Δεκεμβρίου 2011

## Άσκηση 1: Ασυμπτωτικός Συμβολισμός, Αναδρομικές Σχέσεις

(α)

Ταξινομώντας τις συναρτήσεις της άσκησης σε αύξουσα σειρά τάξης μεγέθους προκύπτει η εξής διάταξη :

$g_1 = \log n^3$	$g_2 = \sqrt{n}(\log n)^{50}$	$g_3 = \frac{n}{\log \log n}$
$g_4 = \log n!$	$g_5 = n(\log n)^{10}$	$g_6 = n^{1.01}$
$g_7 = 5^{\log n}$	$g_8 = \sum_{k=1}^n k^5$	$g_9 = (\log n)^{\log n} = n^{\log \log n}$
$g_{10} = 2^{(\log n)^4}$	$g_{11} = (\log n)^{\sqrt{n}}$	$g_{12} = e^{\frac{n}{\ln n}}$
$g_{13} = n \cdot 3^n$	$g_{14} = 2^{2n}$	$g_{15} = \sqrt{n!}$

(β)

1. ΧΡΗΣΗ ΤΟΥ MASTER THEOREM:

- $\alpha = 5, \beta = 7$  άρα  $n^{\log_\beta \alpha} = n^{\log_7 5} \simeq n^{0.827}$
- $f(n) = n \log n = \Omega(n^{\log_7 5 + \epsilon})$
- $5f(\frac{n}{7}) = 5\frac{n}{7} \log(\frac{n}{7}) \leq \frac{5}{7}n \log(n)$

Άρα σύμφωνα με την 3η περίπτωση του MASTER THEOREM προκύπτει ότι:  
 $T(n) = \Theta(n \log n)$

2. ΧΡΗΣΗ ΤΟΥ MASTER THEOREM:

- $T(n) = 4T(\frac{n}{5}) + \frac{n}{(\log n)^2}$
- $n^{\log_\beta \alpha} = n^{\log_5 4} \simeq n^{0.861}$

Άρα  $T(n) = \Theta(\frac{n}{(\log n)^2})$

3.  $T(n) = T(\frac{n}{3}) + 3T(\frac{n}{7}) + n \implies T(n) = \Theta(n \log n)$

4. ΧΡΗΣΗ ΤΟΥ MASTER THEOREM:

- $\alpha = 6, \beta = 6$  άρα  $n^{\log_\beta \alpha} = n^{\log_6 6} = n$
- $f(n) = \Theta(n^{\log_6 6})$

Άρα σύμφωνα με την 2η περίπτωση του MASTER THEOREM προκύπτει ότι:  
 $T(n) = \Theta(n \log n)$

5. ΧΡΗΣΗ ΤΟΥ MASTER THEOREM:

- $T(n)$  της μορφής  $T(n) = T(\gamma_1 n) + T(\gamma_2 n) + \Theta(n)$
- $\gamma_1 + \gamma_2 = \frac{1}{3} + \frac{2}{3} = 1$

Άρα ισχύει  $T(n) = \Theta(n \log n)$

6. ΧΡΗΣΗ ΤΟΥ MASTER THEOREM:

- $\alpha = 16, \beta = 4$  άρα  $n^{\log_\beta \alpha} = n^{\log_4 16} = n^2$
- $f(n) = n^3 \log(n)^2 = \Omega(n^{\log_4 16 + \epsilon})$
- $\alpha f(\frac{n}{\beta}) = 16 f(\frac{n}{4}) = 16 \frac{n^3}{4^3} \log(\frac{n}{4})^2 = \frac{16}{4^3} n^3 \log(\frac{n}{4})^2 \leq c f(n)$  με  $c < 1$

Άρα σύμφωνα με την 3η περίπτωση του MASTER THEOREM προκύπτει ότι:  
 $T(n) = \Theta(n^3 (\log n)^2)$

7.  $T(n) = T(\sqrt{n}) + \Theta(\log \log n)$ . Έστω  $k = \log n$ ,  $f(k) = T(n)$ . Άρα  
 $T(\sqrt{n}) = f(\log \sqrt{n}) = f(\frac{\log n}{2}) = f(\frac{k}{2}) \implies f(k) = f(k/2) + \Theta(\log k)$ .  
 Επομένως:  $T(n) = \Theta(\frac{(\log k)^2 (\log k + 1)}{2})$

8.  $T(n) = T(n-3) + \log n \implies T(n) = \Theta(n \log n)$

## Άσκηση 2: Ταξινόμηση σε πίνακα με πολλά ίδια στοιχεία

(α)

Ο πίνακας  $A[1 \dots n]$  χαρακτηρίζεται από πολλαπλές εμφανίσεις των στοιχείων του, με  $O(\log(n)^d)$  (για κάποια σταθερά  $d \geq 1$ ) διαφορετικά στοιχεία. Επομένως ο πίνακας που αποτελείται από τα διαφορετικά στοιχεία του  $A$ , έστω  $B[1 \dots k]$ , θα έχει πλήθος στοιχείων  $k \in O(\log(n)^d)$ . Αν εκμεταλευτούμε την ιδιότητα του πίνακα  $A$  μπορούμε να κατασκευάσουμε συγκριτικό αλγόριθμο ταξινόμησης που θα έχει πολυπλοκότητα  $(n \log \log n)$ . Θεωρούμε χωρίς βλάβη της γενικότητας ότι ο  $A$  αποτελείται μόνο από θετικά στοιχεία (δεν θα υπήρχε πρόβλημα στην πολυπλοκότητα του αλγορίθμου αν ο πίνακας περιείχε οποιουσδήποτε ακέραιους, μιας και η διαδικασία να ανάγουμε οποιονδήποτε πίνακα σε πίνακα μόνο θετικών είναι γραμμικής πολυπλοκότητας). Περιγραφικά, θα χρειαστούμε τις εξής διαδικασίες που θα απαρτίζουν τον αλγόριθμό μας:

- `freq_array()`: Συνάρτηση που δέχεται ως είσοδο έναν πίνακα (στην δικιά μας περίπτωση τον  $A$ ) και μας επιστρέφει έναν πίνακα  $B$  μεγέθους `max_array(A)` (μέγιστο στοιχείο του πίνακα) που για κάθε θέση του, έστω  $i$ , δείχνει την αντίστοιχη συχνότητα εμφάνισης του  $i$  στον  $A$ . Παράλληλα δημιουργεί έναν πίνακα  $Unique$  που περιέχει μόνο τα διαφορετικά στοιχεία του  $A$  και επιστρέφει το `diff_num` που είναι το πλήθος των διαφορετικών στοιχείων.
- `qsort_array()`: Συνάρτηση που ταξινομεί έναν πίνακα με την quick sort.
- `reconstruct_array()`: Συνάρτηση που κατασκευάζει τον ζητούμενο ταξινομημένο πίνακα.

Ακολουθούν σε ψευδογλώσσα οι παραπάνω συναρτήσεις:

---

```

1: procedure freq_array( $A$ )
2:    $j \leftarrow 1$ 
3:   initialize( $B$ )                                ▷ Αρχικοποίηση του πίνακα  $B$  με μηδενικά
4:   for  $i$  in  $[1 \dots n]$  do
5:     if  $B[A[i]] = 0$  then
6:        $Unique[j] \leftarrow A[i]$                     ▷ Unique: πίνακας διαφορετικών στοιχείων
7:        $j \leftarrow j + 1$ 
8:     end if
9:      $B[A[i]] \leftarrow B[A[i]] + 1$ 
10:  end for
11:   $diff\_num \leftarrow j - 1$ 
12:  return  $Unique, B, diff\_num$ 
13: end procedure

```

---



---

```

1: procedure reconstruct_array( $B, Unique, diff\_num$ )
2:    $init \leftarrow 1$ 
3:    $k \leftarrow diff\_num$ 
4:   for  $i$  in  $[1 \dots k]$  do
5:     for  $j$  in  $[1 \dots B[Unique[i]]]$  do
6:        $A[init] \leftarrow Unique[i]$ 
7:        $init \leftarrow init + 1$ 
8:     end for
9:   end for
10:  return  $A$ 

```

---

11: **end procedure**

---

Η πολυπλοκότητα του αλγόριθμου μας θα είναι  $O(\log n^d \log \log n^d)$  που όμως είναι ίση με  $O(n \log \log n)$  αφού  $(\log n)^d \leq n$  (ακόμα και στην χειρότερη περίπτωση, το πλήθος των διαφορετικών στοιχείων είναι μικρότερο από το πλήθος όλων των στοιχείων). Αυτό συμβαίνει αφού αν καλέσουμε κατάλληλα τις παραπάνω συναρτήσεις για την ταξινόμηση του  $A$ , όλες είναι γραμμικής πολυπλοκότητας, εκτός από την quick sort όπου θέλει  $n \log n$  χρόνο. Όμως το μέγεθος του πίνακα που ταξινομεί είναι πολυλογαριθμικά μικρότερο από αυτό του αρχικού πίνακα. Έτσι στην περίπτωση ταξινόμησης σαν και αυτή της άσκησης δεν ισχύει το κάτω φράγμα  $\Omega(n \log n)$  γιατί εκμεταλλευόμαστε την ιδιότητα των στοιχείων μειώνοντας την χρονική πολυπλοκότητα αλλά αυξάνοντας πιθανά την χωρική (ο πίνακας  $B$  είναι μεγέθους  $\max\_array(A)$ , δηλαδή μέγεθος ίσο με το μέγιστο στοιχείο).

### Άσκηση 3: Δυαδική Αναζήτηση

(α)

Ο αλγόριθμος για την επίλυση του προβλήματος θα αποτελείται από τα εξής βήματα:

- Εύρεση της τιμής του  $n$  (μέγεθος του πίνακα), σε λογαριθμικό χρόνο.
- Εκτέλεση δυαδικής αναζήτησης με δεδομένο το μέγεθος του πίνακα, πάλι σε λογαριθμικό χρόνο.

Έτσι συνολικά η πολυπλοκότητα του αλγορίθμου μας θα είναι  $\log n + \log n$  δηλαδή  $O(\log n)$ . Για το 1ο βήμα:

- Ξεκινάμε με δείκτη  $i = 1$  και τον αυξάνουμε με βήμα  $2, 2^2, 2^3, 2^4, \dots$  μέχρις ότου μας επιστραφεί το μήνυμα σφάλματος  $\infty$ .
- Έστω  $2^k$  η θέση του πίνακα όπου κατά την προσπέλαση της εμφανίστηκε το μήνυμα σφάλματος.
- Εκτελούμε δυαδική αναζήτηση μεταξύ των θέσεων  $2^{k-1}$  και  $2^k$  μέχρι να βρούμε το ζητούμενο  $n$ , γνωρίζοντας ότι θα είναι το τελευταίο στοιχείο που δεν θα εμφανιστεί το μήνυμα λάθους.

Στη συνέχεια, για το 2ο βήμα, εκτελούμε δυαδική αναζήτηση για το στοιχείο  $x$  στον πίνακα  $A[1..n]$ , αφού γνωρίζουμε πλέον και το μέγεθος του πίνακα  $A$ .

## (β)

Για την επίλυση του προβλήματος σε αυτό το υποερώτημα θα χρειαστεί να εφαρμόσουμε την ιδέα της δυαδικής αναζήτησης στην ένωση 2 πινάκων. Αφού οι 2 πίνακες A,B είναι ταξινομημένοι και τα στοιχεία τους διαφορετικά, αρχικά, δεν είναι δυνατό το ζητούμενο στοιχείο να βρίσκεται σε θέση μεγαλύτερη της  $k$ . Άρα η αναζήτηση περιορίζεται στα στοιχεία  $A[1...k]$ ,  $B[1...k]$ . Ακολουθεί ο αλγόριθμος για την επίλυση του προβλήματος:

---

```
1: procedure  $k\_smallest(A[n], B[n])$ 
2:    $i \leftarrow k \text{ div } 2$ 
3:    $j \leftarrow k - i$ 
4:    $step \leftarrow k \text{ div } 4$ 
5:   while  $step > 0$  do
6:     if  $A[i] > B[j]$  then
7:        $i \leftarrow i - step$ 
8:        $j \leftarrow j + step$ 
9:     else
10:       $i \leftarrow i + step$ 
11:       $j \leftarrow j - step$ 
12:    end if
13:     $step \leftarrow step \text{ div } 2$ 
14:  end while
15:  if  $A[i] > B[j]$  then
16:     $x \leftarrow A[i]$ 
17:  else
18:     $x \leftarrow B[j]$ 
19:  end if
20:  return  $x$ 
21: end procedure
```

---

## Άσκηση 4: Συλλογή comics

Η λύση του προβλήματος της Συλλογής Comics στηρίζεται εν μέρη στην ιδέα της δυαδικής αναζήτησης. Συγκεκριμένα, η ιδέα που θα βασιστούμε για την κατασκευή του αλγορίθμου είναι ότι μετά από κάθε ανάγνωση των ψηφίων ενός comic θα ελλατώνουμε στο μισό τα ψηφία που θα πρέπει να εξεταστούν (δηλ. σύμφωνα με την εκφώνηση, τις ερωτήσεις της μορφής "Ποιό είναι το  $i$ -οστό  $bit$  στην αρίθμηση του τεύχους  $j$ ") στην επόμενη επανάληψη. Έτσι σε κάθε βήμα

χρειάζεται να διαβάζουμε  $n$  ψηφία τα οποία στην επόμενη αναζήτηση μειώνονται στο μισό άρα:

- $T(n) = n + T(n/2) \implies T(n) = 2n - 2 + c1$
- ΠΟΛΥΠΛΟΚΟΤΗΤΑ:  $n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \dots = \sum_{k=0}^{\log n} \frac{n}{2^k} \implies O(n)$

Θεωρούμε ότι ο δείκτης  $i$  είναι αρχικά 0 και το σύνολο των τευχών που εξετάζουμε είναι το *ComicsSet* με αρχικά όλα τα τεύχη. Ακόμα, έστω *ComicsNumber* κάθε φορά το σύνολο των στοιχείων του *ComicsSet*. Τα βήματα του αλγορίθμου μας θα είναι τα εξής παρακάτω:

1. Διάβάζουμε το  $i$ -οστό (απο το τέλος) ψηφίο σε κάθε τεύχος του *ComicsSet* και ανάλογα με το αν είναι 1 ή 0 χωρίζουμε τα τεύχη σε 2 σύνολα.
2. Αν υπάρχουν περισσότερα 0 απο 1 τότε το τεύχος που λείπει βρίσκεται στα  $\frac{1}{2} \cdot \text{ComicsNumber}$  μεγαλύτερα στοιχεία του τρέχοντος *ComicsSet*. Το τρέχον *ComicsSet* γίνεται το σύνολο με τα τεύχη με 1 στο  $i$ -οστο ψηφίο.
3. Αν υπάρχουν περισσότερα 1 απο 0 τότε το τεύχος που λείπει βρίσκεται στα  $\frac{1}{2} \cdot \text{ComicsNumber}$  μικρότερα στοιχεία του τρέχοντος *ComicsSet*. Το τρέχον *ComicsSet* γίνεται το σύνολο με τα τεύχη με 0 στο  $i$ -οστο ψηφίο.
4. Πήγαινουμε ξανά στο βήμα 1 για  $i = i + 1$  μέχρι το *ComicsNumber* να γίνει 1.
5. Τέλος το στοιχείο που ψάχνουμε είναι το στοιχείο με αριθμό τεύχους ίδιο με αυτό του τελευταίου στοιχείου του *ComicsSet*, αν αλλάξουμε το Most Significant Bit στο συμπληρωματικό του.

## Άσκηση 5: Πολυκατοικίες χωρίς θέα

Για την επίλυση του προβλήματος «ΠΟΛΥΚΑΤΟΙΚΙΕΣ ΧΩΡΙΣ ΘΕΑ» θα χρησιμοποιήσουμε μια παραλλαγμένη μορφή του αλγόριθμου που βρίσκει τις κοντινότερες μικρότερες τιμές «ALL NEAREST SMALLER VALUES» σε μια ακολουθία αριθμών (πολυπλοκότητα  $O(n)$ ). Στην προκειμένη περίπτωση χρειάζεται να βρούμε όλες τις κοντινότερες μεγαλύτερες τιμές κάθε στοιχείου του πίνακα  $A$ . Ένας αποδοτικός τρόπος για να υλοποιήσουμε το παραπάνω θα είναι χρησιμοποιώντας τη δομή δεδομένων "στοίβα" (χρησιμοποιούνται χωρίς να οριστούν οι συναρτήσεις *push()*, *pop()*, *top()* για την εισαγωγή και εξαγωγή στοιχείων απο την στοίβα και για το κορυφαίο στοιχείο της). Ακολουθεί υλοποίηση του παραπάνω αλγόριθμου σε ψευδογλώσσα:

- ΠΟΛΥΠΛΟΚΟΤΗΤΑ ΑΛΓΟΡΙΘΜΟΥ:  $O(n)$

---

```

1: procedure NearestLargerValues( $A[n]$ )
2:   initialize( $S$ )                                ▷ Αρχικοποίηση άδειας στοίβας  $S$ 
3:    $i \leftarrow 1$ 
4:    $j \leftarrow 1$ 
5:   for  $x$  in  $A[1..n]$  do
6:     while  $S$  is nonempty and  $top(S) \leq x$  do
7:       pop( $S$ )
8:        $j \leftarrow j - 1$ 
9:     end while
10:    if empty( $S$ ) then
11:       $x$  has no preceeding larger value
12:       $B[i] \leftarrow 0$ 
13:    else
14:       $top(S)$  is the nearest larger value to  $x$ 
15:       $B[i] \leftarrow j$ 
16:       $i \leftarrow i + 1$ 
17:    end if
18:    push( $x$ )
19:     $j \leftarrow j + 1$ 
20:  end for
21:  return  $B[n]$ 
22: end procedure

```

---

Έτσι ο αλγόριθμος έχει ως έξοδο ένα πίνακα που για κάθε πολυκατοικία  $i : 1..n$  αντιστοιχίζει στην θέση  $B[i]$  τον αριθμό της πολυκατοικίας που βρίσκεται πιο κοντά (απο αριστερά) και είναι ψηλότερη (αν δεν υπάρχει τέτοια πολυκατοικία, το  $B[i]$  γίνεται 0). Ο αλγόριθμος εκτελείται σε γραμμικό χρόνο, αν και αυτό δεν είναι εμφανές λόγω των εμφωλευμένων επαναλήψεων. Είναι γραμμικής πολυπλοκότητας όμως, αφού κάθε επανάληψη του εσωτερικού βρόγχου αφαιρεί ένα στοιχείο το οποίο είχε προστεθεί σε μια προηγούμενη επανάληψη του εξωτερικού βρόγχου.