Γιαννουτάκης Παναγιώτης

It1238@uom.edu.gr

It1238

Εισαγωγή στην Ανάλυση Αλγορίθμων

Ασκήσεις (2)

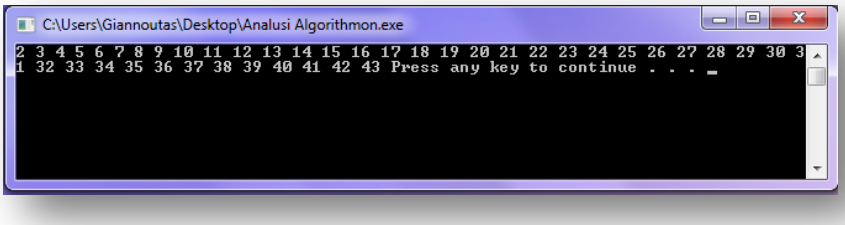
Άσκηση 1

Η ταξινόμηση με εισαγωγή υπερτερεί της ταξινόμησης με συγχώνευση, δηλαδή είναι ταχύτερη, όταν ισχύει η ανίσωση $8n^2 < 64n \cdot \log_2 n \Leftrightarrow n < 8 \cdot \log_2 n$.

Για να λυθεί αυτή η ανίσωση χρησιμοποιήθηκε ένα πρόγραμμα γραμμένο σε γλώσσα C. Η λειτουργία του προγράμματος είναι να βρίσκει για ποιες τιμές του i (που αυξάνεται κατά ένα κάθε φορά) ισχύει η ανίσωση $n < 8 \cdot \log_2 n$, και να τις τυπώνει στο Command Line του συστήματος. Οι συγκρίσεις γίνονται έως έναν αριθμό, το 1.000.000 και αυτό διότι γνωρίζουμε ήδη ότι για παραπάνω τιμές του μετρητή i δεν χρειάζεται να ψάξουμε γιατί δεν θα ισχύει η ανίσωση αφού $8n^2 > 64n \cdot \log_2 n$ για μεγάλες τιμές του n. Έτσι βρίσκουμε ότι οι τιμές του n που ικανοποιούν την σχέση είναι στο διάστημα [2,43] ή αλλιώς $2 \leq n \leq 43$. Παρακάτω φαίνεται το πρόγραμμα

```c
#include <stdio.h>
#include "simpio.h"
#include "genlib.h"
#include <math.h>


main()
{
    int i;

    for(i=0; i<1000000; i++) {
        if(i<8*log2(i)){
            printf("%d\n", i);
        }
    }

    system("Pause");
}

```

Και το αποτέλεσμα που βγάζει.



Άσκηση 2

i)

$f(n) = 3n^2 - n + 4$

$g(n) = n \cdot \log n + 5$

Από κανόνα αθροίσματος του συμπωτικού συμβολισμού έχουμε: $O\big(f(n) + g(n)\big) = max\{O\big(f(n)\big), O(g(n))\} = max\{O(n^2, O(n \cdot \log n\} = O(n^2)$

ii)

$f(n) = \sqrt{n}$

$g(n) = \log n$

Η λύση θα γίνει με χρήση ορίων που υπάρχει στην θεωρία του μαθήματος. Ο παρακάτω κώδικας σε MatLab βγάζει αποτέλεσμα 1 οπότε ισχύει η σχέση $f(n) + g(n) = O(\sqrt{n})$

```
>> limit((sqrt(n)+log(n))/sqrt(n),n,inf)

ans =

1
```

iii)

$$(n+a)^k = n^k + kn^{k-1}a + (k-1)(n^{k-2}a^2) + .. + a^k$$

Οπότε σύμφωνα με τον κανόνα του πολυωνύμου στον ασυμπτωτικό συμβολισμό βρίσκουμε ότι $O(n^k + kn^{k-1}a + (k-1)(n^{k-2}a^2) + .. + a^k) = O(n^k)$.

iv)α)

$$f(n) = 2^{n+1}, g(n) = 2^n$$

Για να ισχύει η υπόθεση της άσκησης θα πρέπει να ισχύει
$f(n) \leq c \cdot g(n)$ *που ισχύει για* $n_0 = 1$ *και* $c = 2$ *γιατί* $2^{n+1} \leq c \cdot 2^n \Leftrightarrow 4 \leq 2 \cdot 2$.

Άρα ισχύει και ότι $2^{n+1} = O(2^n)$

β) Σύμφωνα με τον κώδικα MatLab και την θεωρία τον ορίων για τον ασυμπτωτικό συμβολισμό το όριο βγαίνει $+\infty$ οπότε η σχέση $2^{2n} = O(2^n)$ δεν ισχύει.

```
>> limit((2^(2*x))/(2^x),x,inf)

ans =

Inf
```

Άσκηση 3

Σύμφωνα με την θεωρία του συμπωτικού συμβολισμού για να δούμε αν μια συνάρτηση ανήκει σε μία άλλη και το αντίστροφο, μπορούμε να πάρουμε το όριο του λόγου των δυο συναρτήσεων και βλέποντας το αποτέλεσμα να βγάλουμε τα συμπεράσματα μας. Τα όρια μπορούμε να τα βρούμε με χρήση του MatLab.

Παρακάτω παρατίθεται ο κώδικας ορισμένων περιπτώσεων ενώ στις άλλες περιπτώσεις μόνο το αποτέλεσμα λόγο απλότητας.

- $f(n) = 10n,\ g(n) = n^2 - 10n$. Ο κώδικας είναι:

```
>> limit((10*x)/((x^2)-(10*x)),x,inf)

ans =

0
```

Άρα συμπεραίνουμε ότι $f(n) = O(g(n))\ και\ g(n) \neq O(f(n))$.

- $f(n) = n^3,\ g(n) = n^2 \log n$. Ο κώδικας είναι:

```
>> limit((x^3)/((x^2)*log(x)),x,inf)

ans =

Inf
```

Άρα συμπεραίνουμε ότι $f(n) \neq O(g(n))\ και\ g(n) = O(f(n))$.

- $f(n) = n \log n\,,\ g(n) = n + \log n$. Ο κώδικας είναι:

```
>> limit((x*log(x))/(x+log(x)),x,inf)

ans =

Inf
```

Άρα συμπεραίνουμε ότι $f(n) \neq O(g(n))\ και\ g(n) = O(f(n))$.

- $f(n) = \log n\,,\ g(n) = \sqrt[k]{n}$.
  Το όριο βγαίνει 0 άρα το αποτέλεσμα είναι $f(n) = O(g(n))\ και\ g(n) \neq O(f(n))$.
- $f(n) = \ln n\,,\ g(n) = \log n$.

Το όριο βγαίνει θετικός αριθμός άρα το αποτέλεσμα είναι $f(n) = O(g(n))\ \kappa\alpha\iota\ g(n) = O(f(n))$.

- $f(n) = \log(n+1),\ g(n) = \log n.$
  Το όριο βγαίνει θετικός αριθμός άρα το αποτέλεσμα είναι $f(n) = O(g(n))\ \kappa\alpha\iota\ g(n) = O(f(n))$.
- $f(n) = \log\log n,\ g(n) = \log n.$
  Το όριο βγαίνει 0 άρα το αποτέλεσμα είναι $f(n) = O(g(n))\ \kappa\alpha\iota\ g(n) \neq O(f(n))$.
- $f(n) = 2^n,\ g(n) = 10^n.$
  Το όριο βγαίνει 0 άρα το αποτέλεσμα είναι $f(n) = O(g(n))\ \kappa\alpha\iota\ g(n) \neq O(f(n))$.
- $f(n) = n^m,\ g(n) = m^n.$
  Το όριο βγαίνει 0 άρα το αποτέλεσμα είναι $f(n) = O(g(n))\ \kappa\alpha\iota\ g(n) \neq O(f(n))$.

Άσκηση 4

1) Έστω O (1) = 1

$$T(n) = aT(n-1) + 1$$

$$= a(aT(n-2) + 1) + 1$$

$$= a(a(aT(n-3) + 1) + 1) + 1$$

…

$$= a^k T(n-k) + k$$

…

$$= a^n T(0) + n = a^n O(1) + n = O(a^n)$$

2) Έστω O(n) = n και O(1) = 1

$$T(n) = aT(n-1) + n$$
$$= a(aT(n-2) + n - 1) + n$$
$$= a(a(aT(n-3) + n - 2) + n - 1) + n$$
…
$$a^k T(n-k) + \sum_{i=0}^{k-1} (n-i)$$
…

$$= a^n T(0) + \sum_{i=0}^{n-1} (n-i) = a^n$$

3) Έστω $O(1) = 1$ $\kappa\alpha\iota$ $n = a^t \Leftrightarrow t = \log(n-a)$

$$T(n) = aT\left(a^t/_a\right) + 1$$
$$= aT(a^{t-1}) + 1$$
$$= a(aT(a^{t-2}) + 1) + 1$$
$$\dots$$
$$= a^k T(a^{t-k}) + k$$
$$\dots$$
$$= nT(1) + \log(n-a) = O(n)$$

4) Θέτω $O(1) = 1, O(n) = n$ $\kappa\alpha\iota$ $n = a^t$

$$T(n) = aT(a^{t-1}) + a^t$$
$$= a(aT(a^{t-2}) + a^{t-1}) + a^t$$
$$\dots$$
$$= a^k T(a^{t-k}) + \sum_{i=0}^{k-1} a^{t-i}$$
$$\dots$$
$$= a^t T(a^0) + \sum_{i=0}^{t-1} a^{t-i} = nO(1) + O(a^t) = O(n)$$

Άσκηση 5

Πρόγραμμα 1.1

Πίνακας 1.1: Υπολογισμός χρόνου εκτέλεσης για το Πρόγραμμα 1.1

εντολή	χρόνος	κώδικας
3	$\tau_{\text{fetch}} + \tau_{\text{store}}$	`result = 0`
4a	$\tau_{\text{fetch}} + \tau_{\text{store}}$	`i = 1`
4b	$(2\tau_{\text{fetch}} + \tau_{<}) \times (n+1)$	`i <= n`
4c	$(2\tau_{\text{fetch}} + \tau_{+} + \tau_{\text{store}}) \times n$	`++i`
5	$(2\tau_{\text{fetch}} + \tau_{+} + \tau_{\text{store}}) \times n$	`result += i`
6	$\tau_{\text{fetch}} + \tau_{\texttt{return}}$	`return result`
ΣΥΝΟΛΟ	$(6\tau_{\text{fetch}} + 2\tau_{\text{store}} + \tau_{<} + 2\tau_{+}) \times n$ $+ (5\tau_{\text{fetch}} + 2\tau_{\text{store}} + \tau_{<} + \tau_{\texttt{return}})$	

Από τον παραπάνω πίνακα βρίσκουμε για κάθε εντολή η ασυμπτωτική συμπεριφορά της είναι $O(1), O(1), O(n+1), O(n), O(n), O(1)$. Οπότε για να υπολογίσουμε το αυστηρό άνω ασυμπτωτικό φράγμα όλου του προγράμματος και σύμφωνα με τους κανόνες του O(n) θα πάρουμε το μεγαλύτερο ασυμπτωτικό φράγμα των εντολών. Άρα

Σύνολο $11n + 9 = O(n)$.

Πρόγραμμα 1.2

Πίνακας 1.2: Υπολογισμός του χρόνου εκτέλεσης για το Πρόγραμμα 1.2

εντολή	χρόνος
3	$3\tau_{\text{fetch}} + \tau_{[\cdot]} + \tau_{\text{store}}$
4a	$2\tau_{\text{fetch}} + \tau_{-} + \tau_{\text{store}}$
4b	$(2\tau_{\text{fetch}} + \tau_{<}) \times (n+1)$
4c	$(2\tau_{\text{fetch}} + \tau_{-} + \tau_{\text{store}}) \times n$
5	$(5\tau_{\text{fetch}} + \tau_{[\cdot]} + \tau_{+} + \tau_{\times} + \tau_{\text{store}}) \times n$
6	$\tau_{\text{fetch}} + \tau_{\texttt{return}}$
ΣΥΝΟΛΟ	$(9\tau_{\text{fetch}} + 2\tau_{\text{store}} + \tau_{<} + \tau_{[\cdot]} + \tau_{+} + \tau_{\times} + \tau_{-}) \times n$ $+ (8\tau_{\text{fetch}} + 2\tau_{\text{store}} + \tau_{[\cdot]} + \tau_{-} + \tau_{<} + \tau_{\texttt{return}})$

Από τον παραπάνω πίνακα βρίσκουμε για κάθε εντολή η ασυμπτωτική συμπεριφορά της είναι $O(1), O(1), O(n+1), O(n), O(n), O(1)$. Άρα

Σύνολο $16n + 14 = O(n)$.

Πρόγραμμα 1.3

Πίνακας 1.3: Υπολογισμός του χρόνου εκτέλεσης του Προγράμματος 1.3

	χρόνος	
εντολή	$n = 0$	$n > 0$
3	$2\tau_{\text{fetch}} + \tau_{<}$	$2\tau_{\text{fetch}} + \tau_{<}$
4	$\tau_{\text{fetch}} + \tau_{\texttt{return}}$	—
6	—	$3\tau_{\text{fetch}} + \tau_{-} + \tau_{\text{store}} + \tau_{\times}$ $+ \tau_{\text{call}} + \tau_{\texttt{return}} + T(n-1)$

Από τον παραπάνω πίνακα βρίσκουμε ότι για n=0 έχουμε $O(1), O(1), -$. Ενώ για n>0 έχουμε $O(1), -, O(1) + T(n-1)$.

Ο τύπος του αναδρομικού αλγορίθμου είναι:

$$T(n) = \begin{cases} O(1), & n = 0 \\ O(1) + T(n-1), & n > 0 \end{cases}$$

Για να βρούμε την αναδρομική σχέση του αλγορίθμου θα πάρουμε διαδοχικά.

$$T(n) = T(n-1) + 1$$

$$= (T(n-2) + 1) + 1$$

$$= \big((T(n-3) + 1) + 1\big) + 1$$

$$\dots$$

$$T(n-k) + k$$

$$\dots$$

$$= T(0) + n \;\; \text{όπου } n - k = 0$$

Και από τον αρχικό τύπο έχουμε $T(n) = n + 1$. Άρα είναι $T(n) = O(n)$ και η συνολική πολυπλοκότητα είναι

$$T(n) = \begin{cases} O(1), & n = 0 \\ O(n), & n > 0 \end{cases}$$

Πρόγραμμα 2.5

Πίνακας 2.5: Υπολογισμός του χρόνου εκτέλεσης του Προγ. 2.5

εντολή	χρόνος
3	2
4a	2
4b	$3(n+2)$
4c	$4(n+1)$
6	$2(n+1)$
7a	$2(n+1)$
7b	$3\sum_{i=0}^{n}(i+1)$
7c	$4\sum_{i=0}^{n} i$
8	$4\sum_{i=0}^{n} i$
9	$4(n+1)$
10	2
ΣΥΝΟΛΟ	$\frac{11}{2}n^2+\frac{47}{2}n+24$

Από τον παραπάνω πίνακα βρίσκουμε για κάθε εντολή η ασυμπτωτική συμπεριφορά της είναι $O(1), O(1), O(n+2), O(n+1), O(n+1), O(n+1), O(n^2+1), O(n^2), O(n^2), O(n+1), O(1)$. Άρα

Σύνολο $\frac{11}{2}n^2+\frac{47}{2}n+24 = O(n^2)$.

Πρόγραμμα 2.6

Πίνακας 2.6: Υπολογισμός του χρόνου εκτέλεσης του Προγ. 2.6

εντολή	χρόνος
3	2
4a	2
4b	$3(n+2)$
4c	$4(n+1)$
5	$6(n+1)$
6	2
ΣΥΝΟΛΟ	$13n+22$

Από τον παραπάνω πίνακα βρίσκουμε για κάθε εντολή η ασυμπτωτική συμπεριφορά της είναι $O(1), O(1), O(n+2), O(n+1), O(n+1), O(1)$. Άρα

Σύνολο $13n+22 = O(n)$.

Άσκηση 6

Για την άσκηση κάνω ένα συμβιβασμό και θεωρώ ότι στο τμήμα του αλγορίθμου όταν λέει <<to n>> σημαίνει $\leq n$.

$\boldsymbol{i}$)

Εντολή	Πράξεις
$term := term \cdot x$	$1\ x\ K$
$polyval := polyval + term$	$1\ x\ K$

Όπου

$$K = \sum_{i=1}^{n}\sum_{j=1}^{i} 1 = \sum_{i=1}^{n}(i-1+1) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{n^2+n}{2}$$

Άρα $S(n) = n^2 + n$

$\boldsymbol{ii}$)

Η πολυπλοκότητα του αλγορίθμου για το $S(n) = n^2 + n$ είναι $O(n^2)$. Αυτό σημαίνει ότι το ασυμπτωτικό άνω φράγμα θα είναι της μορφής:

$$O\big(g(n)\big) = \{S(n): \exists c \in R, \exists n_0 \in N: \forall n \geq n_0, S(n) \leq c \cdot g(n)$$

Όπου $g(n) = n^2$.

$$\forall n \geq n^0, n^2 + n \leq c \cdot n^2 \Leftrightarrow \frac{1}{n} \leq c$$

Που ισχύει αν $c \geq 1\ \kappa\alpha\iota\ n \geq 1 \Rightarrow c = 1\ \kappa\alpha\iota\ n = 1$

Και αφού υπάρχουν τα n_0 και c η υπόθεση $S(n) \leq c \cdot g(n)$ ισχύει. Άρα

$S(n) = O(n^2)$.

Άσκηση 7

Για την άσκηση κάνω ένα συμβιβασμό και θεωρώ ότι στο τμήμα του αλγορίθμου όταν λέει <<το n>> σημαίνει $\leq n$.

$\boldsymbol{i})$

Εντολή	Πράξεις
$polyval \coloneqq polyval \cdot x + a[n-i]$	$2\ x\ n$

Άρα $M(n) = 2n$

$\boldsymbol{ii})$

Η πολυπλοκότητα του $M(n) = 2n$ είναι $O(n)$.

$\boldsymbol{iii})$

Η πολυπλοκότητα του αλγορίθμου για την επίλυση ενός πολυωνύμου με την μέθοδο Horner είναι σαφώς ταχύτερη και αποδοτικότερη από τον αντίστοιχο αλγόριθμο της άσκησης 6, για ορισμένες τιμές του n. Συγκεκριμένα ο αλγόριθμος της άσκησης 7 είναι πιο γρήγορος του αντίστοιχου της άσκησης 6 για $n \geq 1$. Πράγματι, $n^2 + n \geq 2n \Leftrightarrow n + 1 \geq 2 \Leftrightarrow n \geq 1$.

Γιαννουτάκης Παναγιώτης
It1238@uom.edu.gr
It1238