ΠΛΗ30

ΕΝΟΤΗΤΑ 1: ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

Μάθημα 1.1:
Ανάλυση Διαδικαστικών Αλγορίθμων

Δημήτρης Ψούνης

www.psounis.gr

ΠΕΡΙΕΧΟΜΕΝΑ

Α. Σκοπός του Μαθήματος

Επίπεδο Α

- Άριστη γνώση του πως εξάγεται ο Θ συμβολισμός μιας συνάρτησης πολυπλοκότητας
- Τι είναι αλγόριθμος, τι είναι συνάρτηση πολυπλοκότητας ενός αλγορίθμου, πως αποφασίζουμε ποιος αλγόριθμος είναι ο καλύτερος για ένα πρόβλημα

Επίπεδο Β

- Υπολογισμός της συνάρτησης πολυπλοκότητας με χρήση των αθροισμάτων

Επίπεδο Γ

- (-)

Β. Θεωρία

1. Τι είναι αλγόριθμος

- Αλγόριθμος είναι μια διαδικασία του υπολογιστή, που επιλύει ένα πρόβλημα ως εξής:
  - Δέχεται μια είσοδο δεδομένων (το στιγμιότυπο του προβλήματος)
  - Εκτελεί μια σειρά σαφώς καθορισμένων βημάτων (διατυπωμένα σε μία γλώσσα που αναγνωρίζει ο υπολογιστής – στην ΠΛΗ30 η ψευδογλώσσα)
  - Παράγει μία έξοδο δεδομένων (που απεικονίζει τη λύση του προβλήματος)

ΠΑΡΑΔΕΙΓΜΑΤΑ:

ΤΑΞΙΝΟΜΗΣΗ	ΑΝΑΖΗΤΗΣΗ
Πρόβλημα: Η ταξινόμηση μιας ακολουθίας αριθμών	Πρόβλημα: Η αναζήτηση ενός στοιχείου σε μια ταξινομημένη ακολουθία αριθμών
Παραδείγματα Στιγμιότυπων: •[5, 8, 9, 11, 14] •[4,12,7,9]	Παραδείγματα Στιγμιοτύπων: •[5, 8, 9, 11, 14],11 •[3,6,9,14,17],12
Αλγόριθμοι: InsertionSort, BubbleSort, SelectionSort, MergeSort, QuickSort	Αλγόριθμοι: LinearSearch, BinarySearch
Έξοδος στις δύο εισόδους: [5, 8, 9, 11, 14] και [4,7,9,12]	Έξοδος στις δύο εισόδους: ΝΑΙ - ΟΧΙ

Β. Θεωρία

2. Τι είναι διαδικαστικός αλγόριθμος

- Ένας διαδικαστικός αλγόριθμος είναι μια διαδικασία που υλοποιείται με στοιχειώδεις πράξεις που εκτελεί μία συνήθης γλώσσα προγραμματισμού, όπως:
  - Οι εντολές επανάληψης: for, while, do…while
  - Η εντολή συνθήκης: if…else if…else
  - Εντολές καταχώρησης
  - Αριθμητικές πράξεις (όπως π.χ. +,-,*,/,mod)

- …και <u>ΔΕΝ ΧΡΗΣΙΜΟΠΟΙΕΙ ΑΝΑΔΡΟΜΗ</u> (θα την μελετήσουμε εξαντλητικά σε επόμενα μαθήματα)

Β. Θεωρία

3. Πως καταλαβαίνουμε την ποιότητα του αλγορίθμου

1. Χρονική Πολυπλοκότητα

- Χρειαζόμαστε ένα κριτήριο εκτίμησης του πόσο καλός είναι ένας αλγόριθμος.
  - Θα χρησιμοποιήσουμε την χρονική πολυπλοκότητα χειρότερης περίπτωσης.
- Για να φτάσουμε εκεί όμως θα πρέπει πρώτα να δούμε τι είναι η χρονική πολυπλοκότητα ενός αλγορίθμου:

ΧΡΟΝΙΚΗ ΠΟΛΥΠΛΟΚΟΤΗΤΑ ενός αλγορίθμου είναι μια συνάρτηση που υπολογίζει πόσες πράξεις (καταχωρήσεις, συγκρίσεις και αριθμητικές πράξεις) γίνονται ως συνάρτηση του πλήθους των δεδομένων της εισόδου.

- Θα δούμε ότι:
  - Η συνάρτηση χρονικής πολυπλοκότητας της SelectionSort είναι:

$$T(n) = n^2 + 3n$$

  - Η συνάρτηση χρονικής πολυπλοκότητας της LinearSearch είναι:

$$T(n) = n$$

Β. Θεωρία

3. Πως καταλαβαίνουμε την ποιότητα του αλγορίθμου

1. Χρονική Πολυπλοκότητα (Ο αλγόριθμος LinearSearch)

- Ας δούμε ένα παράδειγμα (αλγόριθμος γραμμικής αναζήτησης – Linear Search)

```
procedure LinearSearch(A,x)

   for i=1 to n
      if (A[i]==x)
         return «ΝΑΙ»
      end if
   end for

   return «ΌΧΙ»
end procedure
```

- Όπου A είναι ένας πίνακας n στοιχείων, στον οποίο αναζητούμε το στοιχείο x. Αν το στοιχείο βρεθεί απαντάμε ΝΑΙ αλλιώς απαντάμε ΌΧΙ.

Β. Θεωρία

3. Πως καταλαβαίνουμε την ποιότητα του αλγορίθμου

1. Χρονική Πολυπλοκότητα (Ο αλγόριθμος LinearSearch)

- Υπάρχουν τρεις τρόποι να αναλύσουμε την χρονική πολυπλοκότητα του αλγορίθμου:
  - Η ανάλυση χειρότερης περίπτωσης (δηλαδή πότε ο αλγόριθμος κάνει τις περισσότερες δυνατές πράξεις). Στον συγκεκριμένο αλγόριθμο, όταν το στοιχείο δεν υπάρχει στον πίνακα, άρα οι πράξεις είναι:

$$\boxed{T(n)=\sum_{i=1}^{n}1=n}$$

  - Η ανάλυση βέλτιστης περίπτωσης (δηλαδή πότε ο αλγόριθμος κάνει τις λιγότερες δυνατές πράξεις). Στον συγκεκριμένο αλγόριθμο όταν το στοιχείο είναι στην πρώτη θέση του πίνακα, άρα οι πράξεις είναι:

$$\boxed{T(n)=1}$$

  - Η ανάλυση μέσης περίπτωσης είναι προχωρημένη μέθοδος ανάλυσης της πολυπλοκότητας και απαιτεί πιθανοτική ανάλυση των δεδομένων εισόδου. Θα δούμε τέτοιου τύπου αναλύσεις σε επόμενα μαθήματα.

Β. Θεωρία

3. Πως καταλαβαίνουμε την ποιότητα του αλγορίθμου

1. Χρονική Πολυπλοκότητα (Ο αλγόριθμος SelectionSort)

- Ακόμη ένα παράδειγμα (ο αλγόριθμος ταξινόμησης SelectionSort)

```
procedure SelectionSort(A)

   for i=1 to n
      pos=i
      for j=i+1 to n
         if (A[j]<A[pos])
            pos=j
         end if
      end for
      temp=A[i]; A[i]=A[pos]; A[pos]=temp
   end for

end procedure
```

- Όπου Α είναι ένας (αταξινόμητος) πίνακας n στοιχείων

Β. Θεωρία

3. Πως καταλαβαίνουμε την ποιότητα του αλγορίθμου

1. Χρονική Πολυπλοκότητα (Ο αλγόριθμος SelectionSort)

- Όταν έχουμε έναν (πιο περίπλοκο) αλγόριθμο προς μελέτη, καλό θα είναι να τον εκτελούμε βήμα-βήμα με κάποια μικρά στιγμιότυπα εκτέλεσης. Με τον τρόπο αυτό καταλαβαίνουμε πως λειτουργεί ο αλγόριθμος. Π.χ. με είσοδο [4 3 5 1 2] έχουμε βήμα βήμα την εκτέλεση:

- Βήμα 1:

1	2	3	4	5
4	3	5	1	2

- Βήμα 2:

1	2	3	4	5
1	3	5	4	2

- Βήμα 3:

1	2	3	4	5
1	2	5	4	3

- Βήμα 4:

1	2	3	4	5
1	2	3	4	5

- Βήμα 5:

1	2	3	4	5
1	2	3	4	5

- Τέλος:

1	2	3	4	5
1	2	3	4	5

Β. Θεωρία

3. Πως καταλαβαίνουμε την ποιότητα του αλγορίθμου

1. Χρονική Πολυπλοκότητα (Ο αλγόριθμος SelectionSort)

- Η ανάλυση χειρότερης περίπτωσης. Η χειρότερη περίπτωση είναι όταν ο αλγόριθμος κάνει συνεχείς καταχωρήσεις (στο βήμα της if) διότι βρίσκει μικρότερο στοιχείο (αυτό συμβαίνει όταν ο πίνακας είναι ταξινομημένος σε φθίνουσα σειρά). Τότε ο αλγόριθμος κάνει τις εξής πράξεις:

$$T(n)=\sum_{i=1}^{n}[1+(\sum_{j=i+1}^{n}2)+3]=$$

$$=\sum_{i=1}^{n}[4+2(\sum_{j=i+1}^{n}1)]=$$

$$=\sum_{i=1}^{n}[4+2(n-(i+1)+1)]=$$

$$=\sum_{i=1}^{n}[4+2(n-i)]=$$

$$=\sum_{i=1}^{n}[4+2n-2i]=$$

$$=\sum_{i=1}^{n}4+\sum_{i=1}^{n}(2n)-\sum_{i=1}^{n}(2i)=$$

$$=4n+2n^2-2\sum_{i=1}^{n}i=4n+2n^2-2\frac{n(n+1)}{2}=$$

$$=n^2+3n$$

Β. Θεωρία

3. Πως καταλαβαίνουμε την ποιότητα του αλγορίθμου

1. Χρονική Πολυπλοκότητα (Ο αλγόριθμος SelectionSort)

- Η ανάλυση καλύτερης περίπτωσης. Η καλύτερη περίπτωση είναι όταν ο πίνακας είναι ήδη ταξινομημένος σε αύξουσα σειρά, οπότε δεν χρειάζεται να γίνει η καταχώρηση της if. Τότε ο αλγόριθμος κάνει τις εξής πράξεις:

$$T(n) = \sum_{i=1}^{n}[1+(\sum_{j=i+1}^{n}1)+3] =$$

$$= \sum_{i=1}^{n}[4+(n-(i+1)+1)] =$$

$$= \sum_{i=1}^{n}[4+n-i] =$$

$$= \sum_{i=1}^{n}[4+n-i] =$$

$$= \sum_{i=1}^{n}4+\sum_{i=1}^{n}(n)-\sum_{i=1}^{n}(i) =$$

$$= 4n+n^2-\sum_{i=1}^{n}i = 4n+n^2-\frac{n(n+1)}{2} =$$

$$= 0.5n^2+2.5n$$

Β. Θεωρία

3. Πως καταλαβαίνουμε την ποιότητα του αλγορίθμου

2. Ασυμπτωτική Εκτίμηση Χρονικής Πολυπλοκότητας

- Μπορούμε να παρατηρήσουμε ότι:
  - Η πολυπλοκότητα χειρότερης περίπτωσης της LinearSearch είναι σαφώς μεγαλύτερη από την πολυπλοκότητα καλύτερης περίπτωσης.
  - Η πολυπλοκότητα χειρότερης περίπτωσης της SelectionSort είναι πολύ κοντά στην πολυπλοκότητα της καλύτερης περίπτωσης.
- Προκειμένου να απλοποιήσουμε την όλη διαδικασία:

ΑΣΥΜΠΤΩΤΙΚΗ ΕΚΤΙΜΗΣΗ της χρονικής πολυπλοκότητας ενός αλγορίθμου είναι ο μέγιστος από τους όρους του αθροίσματος της συνάρτησης πολυπλοκότητας απαλείφοντας τυχόν σταθερές.

Ειδικά ο μέγιστος όρος θα χαρακτηρίζεται μέσω του συμβολισμού Θ(.)

Β. Θεωρία

3. Πως καταλαβαίνουμε την ποιότητα του αλγορίθμου

2. Ασυμπτωτική Εκτίμηση Χρονικής Πολυπλοκότητας

- Με χρήση των παραπάνω:
- Θα λέμε ότι:
  - Η πολυπλοκότητα της LinearSearch στην χειρότερη περίπτωση:
    - Έχει συνάρτηση πολυπλοκότητας : $T(n) = n$
    - Και ασυμπτωτικά: $T(n) = \Theta(n)$
  - Η πολυπλοκότητα της LinearSearch στην καλύτερη περίπτωση:
    - Έχει συνάρτηση πολυπλοκότητας : $T(n) = 1$
    - Και ασυμπτωτικά: $T(n) = \Theta(1)$
  - Η πολυπλοκότητα της SelectionSort στην χειρότερη περίπτωση:
    - Έχει συνάρτηση πολυπλοκότητας : $T(n) = n^2 + 3n$
    - Και ασυμπτωτικά: $T(n) = \Theta(n^2)$
  - Η πολυπλοκότητα της SelectionSort στην καλύτερη περίπτωση:
    - Έχει συνάρτηση πολυπλοκότητας : $T(n) = 0.5n^2 + 2.5n$
    - Και ασυμπτωτικά: $T(n) = \Theta(n^2)$

Β. Θεωρία

3. Πως καταλαβαίνουμε την ποιότητα του αλγορίθμου

2. Ασυμπτωτική Εκτίμηση Χρονικής Πολυπλοκότητας

- Πως όμως θα ξέρουμε ποιος είναι ο μέγιστος όρος ενός αθροίσματος;
  - Ισχύει η εξής ιεραρχία για τις συναρτήσεις πολυπλοκότητας:

ΣΤΑΘΕΡΕΣ < ΛΟΓΑΡΙΘΜΙΚΕΣ < ΠΟΛΥΩΝΥΜΙΚΕΣ < ΕΚΘΕΤΙΚΕΣ < ΥΠΕΡΕΚΘΕΤΙΚΕΣ

- όπου:
  - Σταθερές είναι συναρτήσεις που δεν υπάρχει το n. Εχουμε: $T(n)=\Theta(1)$
  - Λογαριθμικές είναι συναρτήσεις της μορφής:
    $T(n)=\Theta(\log^k n)$
    - Όπου k είναι σταθερα >0
  - Πολυωνυμικές είναι συναρτήσεις της μορφής:
    $T(n)=\Theta(n^k)$
    - Όπου k είναι σταθερα >0
  - Εκθετικές είναι συναρτήσεις της μορφής:
    $T(n)=\Theta(a^n)$
    - Όπου a είναι σταθερα >1
  - Υπερεκθετικές είναι οι εξής δύο συναρτήσεις:
    $T(n)=\Theta(n!)$ και $T(n)=\Theta(n^n)$ με $n!<n^n$

Β. Θεωρία

3. Πως καταλαβαίνουμε την ποιότητα του αλγορίθμου

3. Χωρική Πολυπλοκότητα

- Σε κάποιες εφαρμογές είναι χρήσιμο να ξέρουμε πόσες θέσεις μνήμης απαιτούνται για την εκτέλεση του αλγορίθμου.

- Στις περιπτώσεις αυτές μετράμε πόσες μεταβλητές απαιτούνται για την εκτέλεση του αλγορίθμου.
  - Προσοχή! Ένας πίνακας μεγέθους n, είναι n μεταβλητές.

- Συχνά χρησιμοποιείται ο συμβολισμος Θ(.) για να έχουμε μία ασυμπτωτική εκτίμηση του χώρου εκτέλεσης του αλγορίθμου.

Β. Θεωρία

3. Πως καταλαβαίνουμε την ποιότητα του αλγορίθμου

3. Χωρική Πολυπλοκότητα (Ο αλγόριθμος Fibonacci)

- Θα το δούμε με ένα παράδειγμα:
  - Η ακολουθία Fibonacci ορίζεται ως:

$$f_n = \begin{cases} 1, & n=1 \\ 1, & n=2 \\ f_{n-1}+f_{n-2}, & n>2 \end{cases}$$

  - Ζητάμε να υπολογίσουμε τον n-οστό όρο της ακολουθίας με έναν αλγόριθμο. Ας δούμε τους πρώτους όρους της ακολουθίας

1	2	3	4	5	6	7	8	9	10	
1	1	2	3	5	8	13	21	34	55	...

Β. Θεωρία

3. Πως καταλαβαίνουμε την ποιότητα του αλγορίθμου

3. Χωρική Πολυπλοκότητα (Ο αλγόριθμος Fibonacci)

- Μία υλοποίηση υπολογισμού του n-οστού Fibonacci είναι η ακόλουθη:

```
procedure Fibonacci(n)

   A[1]=1
   A[2]=1
   for i=3 to n
      A[i]=A[i-1]+A[i-2]
   end for
   return A[n]
end procedure
```

- Οι μεταβλητές που χρησιμοποιεί το πρόγραμμα είναι:
  - Οι n μεταβλητές του πίνακα A
  - Οι δύο μεταβλητές n και i
- Συνεπώς η χωρική πολυπλοκότητα είναι T(n)=n+2 και ασυμπτωτικά T(n)=Θ(n)

Β. Θεωρία

3. Πως καταλαβαίνουμε την ποιότητα του αλγορίθμου

3. Χωρική Πολυπλοκότητα (Ο αλγόριθμος Fibonacci)

- Μπορούμε να κάνουμε καλύτερα στην διαχείριση της μνήμης;

```
procedure Fibonacci(n)

   if (n=1) return 1
   else if (n=2) return 1
   else
      a=1
      b=1
      for (i=3 to n)
         c=a+b
         a=b
         b=c
      end for
   end if
   return c
end procedure
```

- Οι μεταβλητές που χρησιμοποιεί το πρόγραμμα είναι:
  - Οι πέντε μεταβλητές i,n,a,b,c
- Συνεπώς η χωρική πολυπλοκότητα είναι T(n)=5 και ασυμπτωτικά T(n)=Θ(1)

Β. Θεωρία

3. Πως καταλαβαίνουμε την ποιότητα του αλγορίθμου

4. Σύγκριση Αλγορίθμων

- Ένα πρόβλημα μπορεί να λυθεί από διαφορετικούς αλγόριθμους.
- Για να αποφασίσουμε ποιος αλγόριθμος είναι καλύτερος:
  - Υπολογίζουμε την ασυμπτωτική πολυπλοκότητα (δηλαδή το Θ(.)) στην χειρότερη περίπτωση.
  - Επιλέγουμε τον αλγόριθμο που έχει τη μικρότερη πολυπλοκότητα.
- Στα επόμενα μαθήματα θα δούμε για τα προβλήματα που μελετήσαμε ότι:

	Καλυτ.Περ/ση	**Μέση Περίπτωση**	**Χειρ.Περ/ση**
LinearSearch	$\Theta(1)$	?	$\Theta(n)$
BinarySearch	$\Theta(1)$	?	$\Theta(\log n)$
BubbleSort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
InsertionSort	$\Theta(n)$	?	$\Theta(n^2)$
SelectionSort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
MergeSort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
QuickSort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$

Γ. Μεθοδολογία Ασκήσεων

1. Υπολογισμός Πολυπλοκότητας

1. Σειριακά τμήματα κώδικα

- Αν έχουμε διαδοχικά τμήματα κώδικα προσθέτουμε τις αντίστοιχες πολυπλοκότητες:

```
....
  A
  B
  Γ
....
```

- Η πολυπλοκότητα θα είναι A+B+Γ

Γ. Μεθοδολογία Ασκήσεων

1. Υπολογισμός Πολυπλοκότητας

2. Υπολογισμός της for

- Κάθε for γίνεται και ένα άθροισμα με κάτω όριο την αρχή του for και πάνω όριο το τέλος του for.
- Π.χ.:

```
Π.χ.:

for (i=A to B)
    ... Εδω γίνονται Κ πράξεις ...
end for
```

- Η πολυπλοκότητα θα είναι

$$T(n)=\sum_{i=A}^{B}K$$

- Χρήσιμα θα φανούν τα εξής αθροίσματα:

$$\boxed{\sum_{i=1}^{n} i = n(n+1)/2}$$

$$\boxed{\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6}$$

$$\boxed{\sum_{i=A}^{B} c = c\sum_{i=A}^{B}1,\ \ c:\sigma\tau\alpha\theta.}$$

$$\boxed{\sum_{i=A}^{B} 1 = \mathrm{B}-\mathrm{A}+1}$$

$$\boxed{\sum_{i=1}^{n}(A+B)=\sum_{i=1}^{n}A+\sum_{i=1}^{n}B}$$

$$\boxed{\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}}$$

Γ. Μεθοδολογία Ασκήσεων

1. Υπολογισμός Πολυπλοκότητας

3. Εμφωλιασμένοι Βρόχοι

- Σε εμφωλιασμένους βρόχους, τηρούμε τους ίδιους κανόνες που είδαμε και για έναν απλό βρόχο.
- Π.χ.:

```
for (i=A to B)
   for (j=C to D)
       ... Εδώ γίνονται K πράξεις ...
   end for
end for
```

- Η πολυπλοκότητα θα είναι

$$T(n)=\sum_{i=A}^{B}\sum_{j=C}^{D}K$$

Γ. Μεθοδολογία Ασκήσεων

2. Συμβολισμός Θ(.)

- Για να εξάγουμε το Θ(.) μιας συνάρτησης πολυπλοκότητας, θα πρέπει να κάνουμε τις όποιες επιμεριστικές ιδιότητες έτσι ώστε να έχουμε «καθαρά» αθροίσματα.
- Έπειτα επιλέγουμε τον μέγιστο από τους όρους του αθροίσματος, και τον εισάγουμε στο Θ(.)
- Παραδείγματα
  1. $\mathrm{T}(n) = n(n+1) = n^2 + n = \Theta(n^2)$
  2. $\mathrm{T}(n) = \dfrac{n(n+1)(2n+1)}{6} = \dfrac{(n^2+n)(2n+1)}{6} = \dfrac{2}{6}n^3 + \dfrac{3}{6}n^2 + \dfrac{1}{6}n = \Theta(n^3)$
- Προσοχή ότι απαλείφονται οι σταθερές που είναι πολλαπλασιασμένες με τους όρους του αθροίσματος.

Γ. Μεθοδολογία Ασκήσεων

3. Ιδιότητες Δυνάμεων

- Ιδιαίτερα στην εξαγωγή του Θ(.) συχνά θα προκύπτει η ανάγκη για τον υπολογισμό δυνάμεων. Ακολουθούν οι σημαντικότερες ιδιότητες δυνάμεων και ριζικών:

$$a^0 = 1$$
$$a^1 = a$$
$$a^{-1} = 1/a$$
$$a^{-k} = 1/a^k$$

$$(a^m)^n = (a^n)^m = a^{nm}$$
$$a^{m^n} = a^{(m^n)}$$
$$a^m a^n = a^{m+n}$$
$$a^m / a^n = a^{m-n}$$
$$a^m / b^m = (a/b)^m$$

$$\sqrt{x} = x^{\frac{1}{2}} = x^{0.5}$$
$$\sqrt[A]{x^B} = x^{\frac{B}{A}}$$

Δ. Ασκήσεις
Ασκηση Κατανόησης 1

- Υπολογίστε μία ασυμπτωτική εκτίμηση των εξης συναρτήσεων πολυπλοκότητας:

$$f_1(n) = n(2n + n^2 + 1)$$

$$f_2(n) = 2^n(2^n + 1)$$

$$f_3(n) = 5^0(n^2 + 4^n) + \log n$$

$$f_4(n) = (2^{\frac{n}{2}})^2 + n^n$$

$$f_5(n) = \log n + n^6 + n! + 1000^n + 14$$

$$f_6(n) = 1000 + n^{0.01}$$

$$f_7(n) = 4^n / 2^n$$

$$f_8(n) = \sqrt[4]{n^2} + \sqrt[6]{n^4} + 4n$$

Δ. Ασκήσεις
Ασκηση Κατανόησης 2

- Υπολογίστε την ακριβή πολυπλοκότητα του παρακάτω τμήματος κώδικα:

```
for i=1 to n
   for j=1 to n
      a=a+1
   end for
   b=a+a*a
end for
```

Δ. Ασκήσεις
Ασκηση Κατανόησης 3

- Υπολογίστε την ακριβή πολυπλοκότητα του παρακάτω τμήματος κώδικα:

```
for i=1 to n
   a=a/2
   for j=1 to n
      a=a*10
   end for
   b=a+a*a/2
   for j=i+1 to n
      a=a+9
   end for
end for
```

Δ. Ασκήσεις
Εφαρμογή 1

- Η εύρεση του ελάχιστου αριθμού σε έναν πίνακα αριθμό μπορεί να υλοποιηθεί με την εξής ρουτίνα:

```
procedure minArray(A)

   min=A[1]
   for i=2 to n
      if (A[i]<min)
         min=A[i]
      end if
   end for

end procedure
```

Δ. Ασκήσεις
Εφαρμογή 1

1. Για την χειρότερη περίπτωση
   1. Υπολογίστε την ακριβή πολυπλοκότητα
   2. Δώστε μία ασυμπτωτική εκτίμηση της πολυπλοκότητας
2. Για την καλύτερη περίπτωση
   1. Υπολογίστε την ακριβή πολυπλοκότητα
   2. Δώστε μια ασυμπτωτική εκτίμηση της πολυπλοκότητας

Δ. Ασκήσεις
Εφαρμογή 2

- Ενας δέυτερος αλγόριθμος ταξινόμησης είναι ο αλγόριθμος ταξινόμησης με εισαγωγή (InsertionSort). Παρακάτω φαίνεται μια υλοποίηση του αλγορίθμου αυτού σε ψευδογλώσσα:

```
procedure InsertionSort(A)

   for i=2 to n
      for j=i-1 to 1
          if (A[j]>A[j+1])
             temp=A[j]
             A[j]=A[j+1]
             A[j+1]=temp
          else
             break
          end if
      end for
   end for

end procedure
```

Δ. Ασκήσεις
Εφαρμογή 2

1. Εκτελέστε ένα μικρό στιγμιοτυπο π.χ. το [5 4 3 1 2] για να γίνει αντιληπτό πως δουλεύει ο αλγόριθμος.
2. Πότε έχουμε την χειρότερη περίπτωση της εκτέλεσης του αλγορίθμου;
3. Ποια η πολυπλοκότητα της χειρότερης περίπτωσης;
4. Δώστε μια ασυμπτωτική εκτίμηση της πολυπλοκότητας της χειρότερης περίπτωσης.
5. Πότε έχουμε την καλύτερη περίπτωση της εκτέλεσης του αλγορίθμου;
6. Ποια η πολυπλοκότητα της καλύτερης περίπτωσης;
7. Δώστε μια ασυμπτωτική εκτίμηση της πολυπλοκότητας της καλύτερης περίπτωσης.