# ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

## Ενότητα 3: Ασυμπτωτικός συμβολισμός

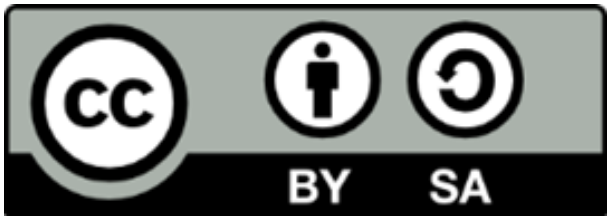
**Μαρία Σατρατζέμη**
**Τμήμα Εφαρμοσμένης Πληροφορικής**



Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

# Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



# Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Μακεδονίας» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

# 3.2 Ασυμπτωτική ανάλυση αλγορίθμων (Ι)

- **Στα προηγούμενες ενότητες παρουσιάσαμε ένα λεπτομερές μοντέλο του υπολογιστή το οποίο περιλαμβάνει αρκετές παραμέτρους χρονομέτρησης και διαπιστώσαμε ότι πρόκειται για λεπτομέρειες που μπορούν να απλουστευθούν. Έτσι στη συνέχεια απλοποιήσαμε το μοντέλο μετρώντας τον χρόνο σε κύκλους ρολογιού και υποθέτοντας ότι κάθε μία από τις παραμέτρους είναι ίση με έναν κύκλο. Σε μεγάλο βαθμό εξακολουθεί το απλοποιημένο μοντέλο να περιλαμβάνει λεπτομέρειες τις οποίες μπορούμε να αγνοήσουμε.**
- **Στην ενότητα αυτή παρουσιάζουμε την έννοια των ασυμπτωτικών φραγμάτων — κυρίως του *O* (μεγάλο όμικρον). Εξετάζοντας αυτά τα φράγματα μπορούμε να πούμε ότι οι κανόνες υπολογισμού και χειρισμού των εκφράσεων *O* απλοποιούν τα μέγιστα την ανάλυση του χρόνου εκτέλεσης ενός προγράμματος, όταν αυτό που μας ενδιαφέρει είναι η ασυμπτωτική του συμπεριφορά.**
- **Προκειμένου να αναλύσουμε έναν αλγόριθμο θα θεωρήσουμε το μοντέλο της μηχανής άμεσης ή τυχαίας προσπέλασης (Random Access Machine - RAM) με έναν επεξεργαστή όπου οι εντολές εκτελούνται ακολουθιακά (η μία κατόπιν της άλλης), δηλ. δεν υπάρχουν πράξεις που να εκτελούνται ταυτόχρονα.**

# Ασυμπτωτική ανάλυση αλγορίθμων (ΙΙ)

- **Το μοντέλο RAM περιλαμβάνει εντολές οι οποίες υπάρχουν σε όλους τους πραγματικούς υπολογιστές:**
- **αριθμητικές πράξεις (πρόσθεση, αφαίρεση, πολλαπλασιασμός, διαίρεση, υπόλοιπο, ακέραιο μέρος)**
- **εντολές μεταφοράς δεδομένων (ανάκληση, αποθήκευση, αντιγραφή)**
- **εντολές ελέγχου (διακλάδωση υπό συνθήκη και άνευ συνθήκης, κλήση υποπρογράμματος και επιστροφή)**
- **Όλες αυτές οι εντολές απαιτούν κάποιον σταθερό χρόνο για την εκτέλεσή τους. Ο χρόνος εκτέλεσης εξαρτάται από τα δεδομένα εισόδου. Γενικά, ο απαιτούμενος χρόνος αυξάνεται καθώς αυξάνεται το μέγεθος της εισόδου και γι' αυτό συνηθίζεται να εκφράζουμε τον χρόνο εκτέλεσης ενός προγράμματος ως συνάρτηση του μεγέθους της εισόδου. Είναι αναγκαίο να διευκρινίσουμε τους όρους ''χρόνος εκτέλεσης'' και ''μέγεθος εισόδου''.**
- **Το ποιο είναι το καταλληλότερο μέτρο για το μέγεθος εισόδου εξαρτάται από το πρόβλημα. Σε κάθε πρόβλημα που εξετάζουμε πρέπει να δηλώνουμε ρητά ποιο μέτρο χρησιμοποιούμε για το μέγεθος της εισόδου.**

# Ασυμπτωτική ανάλυση αλγορίθμων (III)

- **Για πολλά προβλήματα, όπως π.χ. η ταξινόμηση, το μέτρο θα είναι το πλήθος των στοιχείων της εισόδου.**
- **Για άλλα προβλήματα, όπως π.χ. ο πολ/σιασμός δύο ακεραίων, το μέτρο θα είναι το συνολικό πλήθος bit που απαιτούνται για την αναπαράσταση της εισόδου στο δυαδικό σύστημα.**
- **Ενίοτε είναι ενδεδειγμένο να περιγράφουμε το μέγεθος της εισόδου με δύο αριθμούς, όπως για παράδειγμα, όταν η είσοδος είναι ένας γράφος (πλήθος κόμβων, πλήθος ακμών).**

- **Ο χρόνος εκτέλεσης ενός αλγορίθμου για δεδομένη είσοδο είναι το πλήθος των στοιχειωδών πράξεων ή ‘‘βημάτων’’ που εκτελούνται, όπου το ‘‘βήμα’’ πρέπει να ορίζεται έτσι ώστε να είναι κατά το δυνατόν πιο ανεξάρτητο από τον τύπο της μηχανής. Θα κάνουμε την παραδοχή (συμβατή με το μοντέλο RAM) ότι ο χρόνος εκτέλεσης για κάθε γραμμή του προγράμματος ή του ψευδοκώδικά μας είναι σταθερός και αν και η μια γραμμή μπορεί να απαιτεί διαφορετικό χρόνο από την άλλη, θα υποθέσουμε ότι κάθε εκτέλεση της i-οστής γραμμής απαιτεί χρόνο $c_i$, όπου $c_i$ = σταθ. (= $O(1)$, με τον ασυμπτωτικό συμβολισμό).**

➢ **Θεωρούμε, για παράδειγμα, την ανάλυση του χρόνου εκτέλεσης του Προγ. 2.1, που είναι ο αλγόριθμος υπολογισμού της τιμής ενός πολυωνύμου με τον κανόνα του Horner:**

# Ασυμπτωτική ανάλυση αλγορίθμων (IV)

**Πρόγραμμα 1.2:** Υπολογισμός του $\sum_{i=0}^{n} a_i x^i$ με τον κανόνα του Horner

```
1. int Horner (int a[ ], int n, int x)
2. {
3.       int result = a[n];
4.       for ( int i = n -1; i >= 0; - -i )
5.            result = result * x + a[i];
6.       return result;
7. }
```

**Πίνακας 3.1:** Υπολογισμός του χρόνου εκτέλεσης για το Προγ. 1.2 (Horner)

| εντολή | λεπτομερές μοντέλο | απλό μοντέλο | $O$ |
|---|---|---|---|
| 3 | $3\tau_{\text{fetch}} + \tau_{[\cdot]} + \tau_{\text{store}}$ | 5 | $O(1)$ |
| 4a | $2\tau_{\text{fetch}} + \tau_{-} + \tau_{\text{store}}$ | 4 | $O(1)$ |
| 4b | $(2\tau_{\text{fetch}} + \tau_{<}) \times (n+1)$ | $3n+3$ | $O(n)$ |
| 4c | $(2\tau_{\text{fetch}} + \tau_{-} + \tau_{\text{store}}) \times n$ | $4n$ | $O(n)$ |
| 5 | $(5\tau_{\text{fetch}} + \tau_{[\cdot]} + \tau_{+} + \tau_{\times} + \tau_{\text{store}}) \times n$ | $9n$ | $O(n)$ |
| 6 | $\tau_{\text{fetch}} + \tau_{\texttt{return}}$ | 2 | $O(1)$ |
| TOTAL | $(9\tau_{\text{fetch}} + 2\tau_{\text{store}} + \tau_{<} + \tau_{[\cdot]} + \tau_{+} + \tau_{\times} + \tau_{-}) \times n + (8\tau_{\text{fetch}} + 2\tau_{\text{store}} + \tau_{[\cdot]} + \tau_{-} + \tau_{<} + \tau_{\texttt{return}})$ | $16n+14$ | $O(n)$ |

# Ασυμπτωτική ανάλυση αλγορίθμων (V)

- **Στον Πιν. 3.1 παρουσιάζουμε τον χρόνο εκτέλεσης με τρεις τρόπους: μια λεπτομερή ανάλυση, μια απλουστευμένη ανάλυση και μια ασυμπτωτική ανάλυση.**
- **Και οι τρεις τρόποι είναι σε συμφωνία: οι εντολές 3, 4a και 6 εκτελούνται σε μια σταθερή ποσότητα χρόνου και οι εντολές 4b, 4c και 6 εκτελούνται σε μια ποσότητα χρόνου που είναι ανάλογη του $n$ συν μία σταθερά (γραμμική).**
- **Το αποτέλεσμα της ασυμπτωτικής ανάλυσης δεν εξαρτάται από τις τιμές των διαφόρων σταθερών και ως εκ τούτου το ασυμπτωτικό φράγμα μας λέει κάτι το θεμελιώδες για τον χρόνο εκτέλεσης του αλγορίθμου, το οποίο *δεν εξαρτάται από τα χαρακτηριστικά του υπολογιστή (και του μεταγλωττιστή) που χρησιμοποιείται για την εκτέλεση του προγράμματος*.**
- **Βέβαια, ενώ η ασυμπτωτική ανάλυση μπορεί να είναι σημαντικά απλούστερη, το μόνο που μας δίνει είναι ένα άνω φράγμα του χρόνου εκτέλεσης του αλγορίθμου. Ειδικότερα, δεν μαθαίνουμε τον πραγματικό χρόνο εκτέλεσης ενός προγράμματος.**

# 3.3 Κανόνες για την ανάλυση Ο του χρόνου εκτέλεσης (Ι)

- **Κανόνας 1 (Ακολουθιακή σύνθεση)**

***Ο χρόνος εκτέλεσης χειρότερης περίπτωσης μιας ακολουθίας εντολών***

***$E_1$;***

***$E_2$;***

***…***

***$E_k$;***

***είναι $O(\max(T_1(n), \ldots, T_k(n)))$, όπου ο χρόνος εκτέλεσης της $E_i$, της i-οστής εντολής της ακολουθίας, είναι $O(T_i(n))$.***

- **Ο κανόνας αυτός απορρέει από το Θ.1. Ο συνολικός χρόνος εκτέλεσης μιας ακολουθίας εντολών είναι το άθροισμα των χρόνων εκτέλεσης των επιμέρους εντολών. Με βάση το Θ.1, όταν υπολογίζουμε το άθροισμα μιας σειράς από συναρτήσεις, είναι η μεγαλύτερη (max) απ' αυτές που καθορίζει το φράγμα.**

# Κανόνες για την ανάλυση Ο του χρόνου εκτέλεσης (ΙΙ)

- **Κανόνας 2 (Επανάληψη)**

***Ο χρόνος εκτέλεσης χειρότερης περίπτωσης ενός βρόχου*** `for`**,**

`for (`$E_1$`;` $E_2$`;` $E_3$`;)`

$E_4$`;`

***είναι*** $O(\max(T_1(n), T_2(n)\cdot(I(n)+1), T_3(n)\cdot I(n), T_4(n)\cdot I(n)))$***, όπου ο χρόνος εκτέλεσης της εντολής*** $E_i$ ***είναι*** $O(T_i(n))$ ***για*** $i = 1, 2, 3$ ***και 4, και*** $I(n)$ ***είναι το πλήθος των επαναλήψεων που εκτελούνται στη χειρότερη περίπτωση.***

- **Ο κανόνας αυτός είναι απόρροια του Θ.3. Έστω π.χ. ο απλός βρόχος**

```
for (int i = 0; i < n; ++i;)
```

$E_4$`;`

**Εδώ** $E_1$ **είναι** `int i = 0` **, οπότε ο χρόνος εκτέλεσής της είναι μια σταθερά (**$T_1(n) = 1$**).** $E_2$ **είναι** `i < n` **, οπότε ο χρόνος εκτέλεσής της είναι μια σταθερά (**$T_2(n) = 1$**) και** $E_3$ **είναι** `++i` **οπότε ο χρόνος εκτέλεσής της είναι μια σταθερά (**$T_3(n) = 1$**). Επίσης, το πλήθος των επαναλήψεων είναι** $I(n) = n$**.**

# Κανόνες για την ανάλυση Ο του χρόνου εκτέλεσης (ΙΙΙ)

- Σύμφωνα με τον Κανόνα 2 ο χρόνος εκτέλεσης θα είναι $O(\max(1, 1\cdot(n+1), 1\cdot n, T_4(n)\cdot n))$, που απλοποιείται σε $O(\max(n, T_4(n)\cdot n))$. Επιπλέον, αν το σώμα του βρόχου κάνει οτιδήποτε, ο χρόνος εκτέλεσής του πρέπει να είναι $T_4(n)=\Omega(1)$. Άρα, το σώμα του βρόχου θα κυριαρχεί στον υπολογισμό του max και ο χρόνος εκτέλεσης του βρόχου είναι απλά $O(T_4(n)\cdot n))$.
- Αν δεν γνωρίζουμε τον ακριβή αριθμό $I(n)$ των επαναλήψεων που εκτελούνται, μπορούμε πάλι να χρησιμοποιήσουμε τον Κανόνα 2, με την προϋπόθεση ότι έχουμε ένα άνω φράγμα, $I(n) = O(f(n))$, του πλήθους των επαναλήψεων που εκτελούνται. Στην περίπτωση αυτή, ο χρόνος εκτέλεσης είναι
  - $O(\max(T_1(n), T_2(n)\cdot(f(n)+1), T_3(n)\cdot f(n), T_4(n)\cdot f(n)))$

# Κανόνες για την ανάλυση Ο του χρόνου εκτέλεσης (IV)

- **Κανόνας 3 (Εκτέλεση υπό συνθήκη)**

  ***Ο χρόνος εκτέλεσης χειρότερης περίπτωσης μιας εντολής if-then-else,***

  `if (`$E_1$`;)`

  $E_2$`;`

  `else`

  $E_3$`;`

  ***είναι*** $O(\max(T_1(n), T_2(n), T_3(n)))$***, με τον χρόνο εκτέλεσης της εντολής*** $E_i$ ***να είναι*** $O(T_i(n))$***, για*** $i$ **= 1, 2, 3.**

- **Ο κανόνας αυτός απορρέει από την παρατήρηση ότι, ο συνολικός χρόνος μιας εντολής *if-then-else* δεν υπερβαίνει το άθροισμα του χρόνου εκτέλεσης του ελέγχου συνθήκης,** $E_1$**, και του μεγαλύτερου από τους χρόνους εκτέλεσης του τμήματος *then*,** $E_2$**, και του τμήματος *else*,** $E_3$**.**

# Παράδειγμα 1 (Ι)

- **Πρόβλημα: Υπολογισμός των όρων της ακολουθίας αθροισμάτων (σειρά) $S_0, S_1, \ldots, S_{n-1}$, όπου**

$$S_j = \sum_{i=0}^{j} a_i$$

- **Ο αλγόριθμος για τον υπολογισμό των όρων της σειράς δίνεται στο Προγ. 3.1 και στον Πιν. 3.2 συνοψίζουμε τον υπολογισμό του χρόνου εκτέλεσης**

**Πρόγραμμα 3.1:** Υπολογισμός του $\sum_{i=0}^{j} a_i$ για $0 \leq j < n$

```
1. void PrefixSums (int a[], int n)
2. {
3.     for ( int j = n - 1; j >= 0; --j )
4.     {
5.         int sum = 0;
6.         for (int i = 0; i <= j; ++i)
7.             sum += a[i];
8.         a[j] = sum;
9.     }
10.}
```

```
1.  void PrefixSums (int a[], int n)
2.  {
3.      for ( int j = n - 1; j >= 0; --j )
4.      {
5.          int sum = 0;
6.          for (int i = 0; i <= j; ++i)
7.              sum += a[i];
8.          a[j] = sum;
9.      }
10. }
```

**Πίνακας 3.2:**
Υπολογισμός του χρόνου εκτέλεσης

| εντολή | χρόνος |
|---|---|
| 3a | $O(1)$ |
| 3b | $O(1) \times O(n)$ επαναλήψεις |
| 3c | $O(1) \times O(n)$ επαναλήψεις |
| 5 | $O(1) \times O(n)$ επαναλήψεις |
| 6a | $O(1) \times O(n)$ επαναλήψεις |
| 6b | $O(1) \times O(n^2)$ επαναλήψεις |
| 6c | $O(1) \times O(n^2)$ επαναλήψεις |
| 7 | $O(1) \times O(n^2)$ επαναλήψεις |
| 8 | $O(1) \times O(n)$ επαναλήψεις |
| ΣΥΝΟΛΟ | $O(n^2)$ |

# Παράδειγμα 1 (ΙΙ)

- **Συνήθως ο ευκολότερος τρόπος για να αναλύσουμε τον χρόνο εκτέλεσης ενός προγράμματος που περιλαμβάνει ένθετους βρόχους είναι να αρχίσουμε με το σώμα του πλέον εσωτερικού βρόχου.**
- **Στο Προγ. 3.1 τον πλέον εσωτερικό βρόχο συνιστούν οι γραμμές 6 και 7. Συνολικά το κόστος είναι σταθερό: συμπεριλαμβάνει το σώμα του βρόχου (γρ. 7), τον έλεγχο συνθήκης (γρ. 6b) και την προσαύξηση του δείκτη βρόχου (γρ. 6c).**
- **Για μια δεδομένη τιμή του *j*, ο (πλέον) εσωτερικός βρόχος εκτελείται συνολικά *j* + 1 φορές. Και αφού ο εξωτερικός βρόχος εκτελείται για *j* = *n* – 1, *n* – 2, …, 0, στη χειρότερη περίπτωση, ο εσωτερικός βρόχος εκτελείται *n* φορές. Επομένως η συνεισφορά του εσωτερικού βρόχου στον χρόνο εκτέλεσης μιας επανάληψης του εξωτερικού βρόχου είναι *O*(*n*).**
- **Το υπόλοιπο του εξωτερικού βρόχου (γρ. 5 και 8) απαιτεί σταθερό χρόνο σε κάθε επανάληψη. Αυτός ο σταθερός χρόνος υπερκαλύπτεται από τον *O*(*n*) του εσωτερικού βρόχου. Ο εξωτερικός βρόχος κάνει ακριβώς *n* επαναλήψεις. Επομένως ο συνολικός χρόνος εκτέλεσης του προγράμματος είναι *O*($n^2$).**
- **Είναι αυτό ένα αυστηρό *O* φράγμα;**

# Παράδειγμα 1 (III)

- **Είναι λογικό να μην είναι, εξαιτίας της υπόθεσης που κάναμε στην ανάλυση για τη χειρότερη περίπτωση σε σχέση με τον αριθμό των εκτελέσεων του εσωτερικού βρόχου. Ο εσωτερικός βρόχος εκτελείται $j + 1$ φορές για $j = n–1, n–2, \ldots, 0$. Κάναμε όμως τον υπολογισμό υποθέτοντας ότι ο εσωτερικός βρόχος εκτελείται $O(n)$ φορές σε κάθε επανάληψη του εξωτερικού βρόχου. Για να προσδιορίσουμε όμως αν το αποτέλεσμά μας είναι ένα αυστηρό φράγμα, θα πρέπει να προσδιορίσουμε ακριβέστερα τον πραγματικό χρόνο εκτέλεσης του προγράμματος.**
- **Υπάρχει ακριβέστερος τρόπος υπολογισμού που μπορούμε να κάνουμε. Αν παρατηρήσουμε ότι ο χρόνος που καταναλώνεται στον εσωτερικό βρόχο κυριαρχεί (είναι το βαρόμετρο) στον συνολικό χρόνο εκτέλεσης και ότι το κόστος για μια επανάληψη του εσωτερικού βρόχου είναι σταθερό, τότε το μόνο που χρειάζεται να κάνουμε είναι να προσδιορίσουμε ακριβώς το πλήθος των εκτελέσεων του εσωτερικού βρόχου. Αυτό δίνεται από το άθροισμα**

$$\sum_{j=0}^{n-1}(j+1) = \sum_{j=1}^{n} j = \frac{n(n+1)}{2} = \Theta(n^2)$$

- **Επομένως το αποτέλεσμά μας, $T(n) = O(n^2)$, είναι ένα αυστηρό $O$ φράγμα.**

# Παράδειγμα 2

- **Θα συγκρίνουμε τους χρόνους εκτέλεσης δύο διαφορετικών προγραμμάτων που υπολογίζουν τους όρους της ακολουθίας Fibonacci (αριθμοί Fibonacci) οι οποίοι δίνονται από τη σχέση**

$$F_n = \begin{cases} 0 & n = 0, \\ 1 & n = 1, \\ F_{n-1} + F_{n-2} & n \geq 2. \end{cases}$$

- **Το Προγ. 3.2 είναι μια άμεση υλοποίηση του ορισμού του $F_n$ ως αθροίσματος των δύο προηγούμενων όρων:**

**Πρόγραμμα 3.2:** Μη αναδρομικό πρόγραμμα για τον υπολογισμό των αριθμών Fibonacci

```
1. int Fibonacci (int n)
2. {
3.       int previous = -1;
4.       int result = 1;
5.       for ( int i = 0; i <= n; ++i )
6.       {
7.            int sum = result + previous;
8.            previous = result;
9.            result = sum;
10.      }
11.      return result;
12.}
```

**Πίνακας 3.3:** Υπολογισμός του χρόνου εκτέλεσης του Προγράμματος 3.2

| εντολή | χρόνος |
|---|---|
| 3 | $O(1)$ |
| 4 | $O(1)$ |
| 5a | $O(1)$ |
| 5b | $O(1) \times (n+2)$ επαναλήψεις |
| 5c | $O(1) \times (n+1)$ επαναλήψεις |
| 7 | $O(1) \times (n+1)$ επαναλήψεις |
| 8 | $O(1) \times (n+1)$ επαναλήψεις |
| 9 | $O(1) \times (n+1)$ επαναλήψεις |
| 11 | $O(1)$ |
| ΣΥΝΟΛΟ | $O(n)$ |

- **Ο χρόνος εκτέλεσης αυτού του αλγορίθμου είναι *O*(*n*) όπως αναλυτικά παρουσιάζεται στον Πιν. 3.3.**
- **Η ακολουθία Fibonacci είναι αναδρομική αλλά ο αλγόριθμος του Προγ. 3.2 είναι *επαναληπτικός*. Ένας *αναδρομικός* αλγόριθμος δίνεται στο Προγ. 3.3 και ο χρόνος εκτέλεσής του παρουσιάζεται στον Πιν. 3.4:**

**Πρόγραμμα 3.3:** Αναδρομικό πρόγραμα για τον υπολογισμό των αριθμών Fibonacci

```
1. int Fibonacci (int n)
2. {
3.     if (n == 0 || n == 1)
4.         return n;
5.     else
6.         return Fibonacci(n - 1) * Fibonacci (n - 2);
7. }
```

**Πίνακας 3.4:** Υπολογισμός του χρόνου εκτέλεσης του Προγράμματος 3.3

| | χρόνος | |
|---|---|---|
| εντολή | $n < 2$ | $n \geq 2$ |
| 3 | $O(1)$ | $O(1)$ |
| 4 | $O(1)$ | — |
| 6 | — | $T(n-1) + T(n-2) + O(1)$ |
| ΣΥΝΟΛΟ | $O(1)$ | $T(n-1) + T(n-2) + O(1)$ |

- **Από τον Πιν. 3.4 βρίσκουμε ότι ο χρόνος εκτέλεσης του αναδρομικού αλγορίθμου για τους αριθμούς Fibonacci δίνεται από την αναδρομική σχέση**

$$T(n) = \begin{cases} O(1) & n < 2 \\ T(n-1) + T(n-2) + O(1) & n \geq 2 \end{cases}$$

- **Επειδή μας ενδιαφέρει η ασυμπτωτική συμπεριφορά (το ασυμπτωτικό φράγμα) του αποτελέσματος, απλά αγνοούμε προς στιγμήν τα *O* και λύνουμε την αναδρομική σχέση**

$$T(n) = \begin{cases} 1 & n < 2 \\ T(n-1) + T(n-2) + 1 & n \geq 2 \end{cases}$$

- **Πρόκειται για τη μη ομογενή γραμμική αναδρομική σχέση 2ου βαθμού**

$$\alpha_n = \alpha_{n-1} + \alpha_{n-2} + 1 \quad (1)$$

**με $\alpha_0 = 1$ και $\alpha_1 = 1$ (αρχικές συνθήκες)**

- **Για να τη λύσουμε εφαρμόζουμε τη γνωστή, από το μάθημα των Διακριτών Μαθηματικών, διαδικασία επίλυσης γραμμικών αναδρομικών εξισώσεων με σταθερούς συντελεστές.**

- **Η αντίστοιχη ομογενής εξίσωση είναι**

$$a_n - a_{n-1} - a_{n-2} = 0$$

- **με χαρακτηριστική εξίσωση**

$$x^2 - x - 1 = 0$$

- **οι ρίζες της χαρακτηριστικής εξίσωσης βρίσκουμε ότι είναι**

$$\rho_1 = \left(1 + \sqrt{5}\right)/2 \ \text{και} \ \rho_2 = \left(1 - \sqrt{5}\right)/2 = -1/\rho_1$$

- **η γενική λύση της ομογενούς αναδρομικής εξίσωσης είναι της μορφής**

$$a_n^{(H)} = c_1 \cdot \rho_1^n + c_2 \cdot \rho_2^n$$

- **Επειδή το μη ομογενές τμήμα είναι 1, αναζητούμε μερική λύση της μη ομογενούς της μορφής**

$$a_n^{(P)} = c$$

- **Με αντικατάσταση στην (1) βρίσκουμε *c* = -1, οπότε η γενική λύση της (1) είναι**

$$a_n = a_n^{(H)} + a_n^{(P)} = c_1 \cdot \rho_1^n + c_2 \cdot \rho_2^n - 1$$

- **Οι σταθερές βρίσκονται με αντικατάσταση στις αρχικές συνθήκες:**

$$c_1 + c_2 = 2$$

$$c_1\rho_1 + c_2\,\rho_2 = 2$$

- **Λύνοντας το σύστημα βρίσκουμε**

$$c_1 = \left(1+\sqrt{5}\right)/\sqrt{5} \quad \text{και} \quad c_2 = -\left(1-\sqrt{5}\right)/\sqrt{5}$$

- **Τελικά η λύση της αναδρομικής σχέσης είναι**

$$a_n = \left(2/\sqrt{5}\right)\cdot \rho_1^{n+1} - \left(2/\sqrt{5}\right)\cdot \rho_2^{n+1} - 1$$

$$\Rightarrow T(n) = \left(2/\sqrt{5}\right)\left(\rho_1^{n+1} - \rho_2^{n+1}\right) - 1$$

- **Είναι $\rho_1 = \phi \approx 1.622$ και $\rho_2 = -1/\phi \approx -0.62$ οπότε**
  - $T(n) = \Theta(\phi^n)$

# Τέλος Ενότητας