ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΤΜΗΜΑ ΣΤΑΤΙΣΤΙΚΗΣ

Αλγόριθμοι και Δομές Δεδομένων

Διάλεξη 6: Αναδρομικοί αλγόριθμοι

Τσούρος Δήμος

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ

- **Αναδρομική συνάρτηση**: μία συνάρτηση που ορίζεται συναρτήσει του εαυτού της

- **Αναδρομικό πρόγραμμα**: ένα πρόγραμμα που καλεί τον εαυτό του

- Το αναδρομικό πρόγραμμα πρέπει να έχει οπωσδήποτε μία **συνθήκη τερματισμού**

ΑΝΑΔΡΟΜΙΚΟΣ ΑΛΓΟΡΙΘΜΟΣ

- **Αναδρομικός αλγόριθμος (recursive algorithm)** είναι ένας αλγόριθμος που λύνει ένα πρόβλημα λύνοντας ένα ή περισσότερα **μικρότερα στιγμιότυπα** του ίδιου προβλήματος

- Υλοποιούμε **αναδρομικούς αλγόριθμους** με **αναδρομικές συναρτήσεις**

- **Αναδρομικές συναρτήσεις:** συναρτήσεις που καλούν τον εαυτό τους

ΠΑΡΑΓΟΝΤΙΚΟ (1)

$$n! = n * (n-1)!, \text{ για } n \geq 1 \text{ με } 0! = 1$$

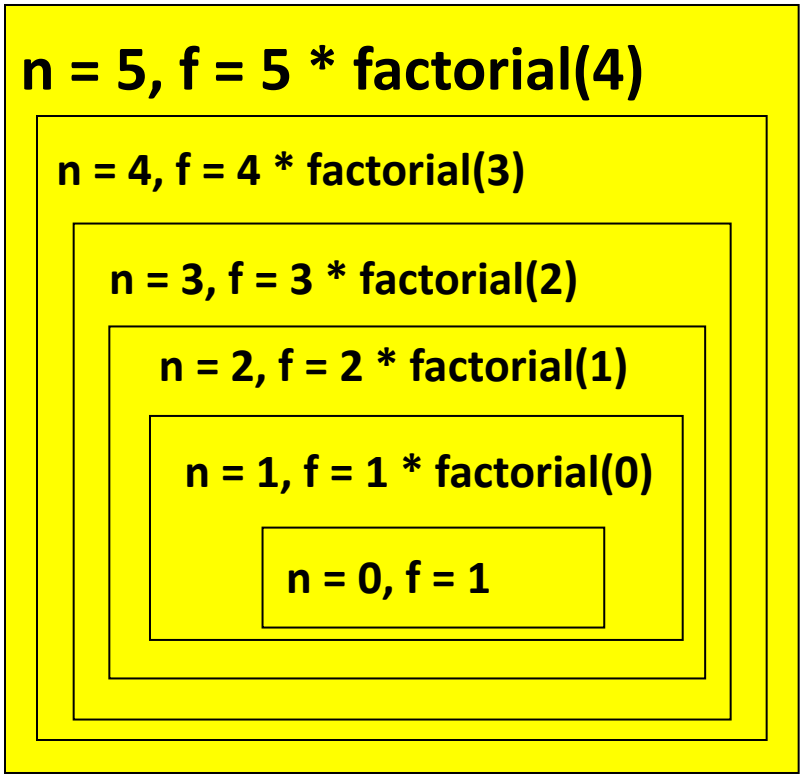
- **Επαναληπτική υλοποίηση**

```
Συνάρτηση Factorial(n), επαναληπτική
1. f ← 1
2. Για i από 1 μέχρι n επανάλαβε
     1. f ← f * i
3. Επιστροφή result
```

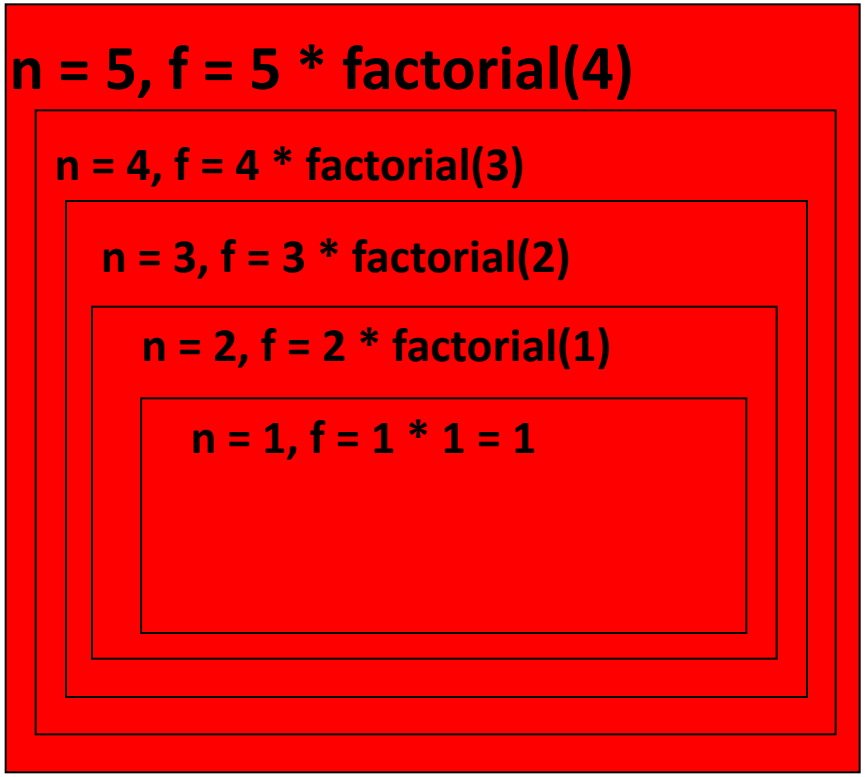
- **Αναδρομική υλοποίηση**

```
Συνάρτηση Factorial(n), αναδρομική
1. Είσοδος n
2. Αν n = 0 τότε
     1. f ← 1
3. Αλλιώς
     1. f ← n * Factorial(n − 1)
4. Επιστροφή f
```

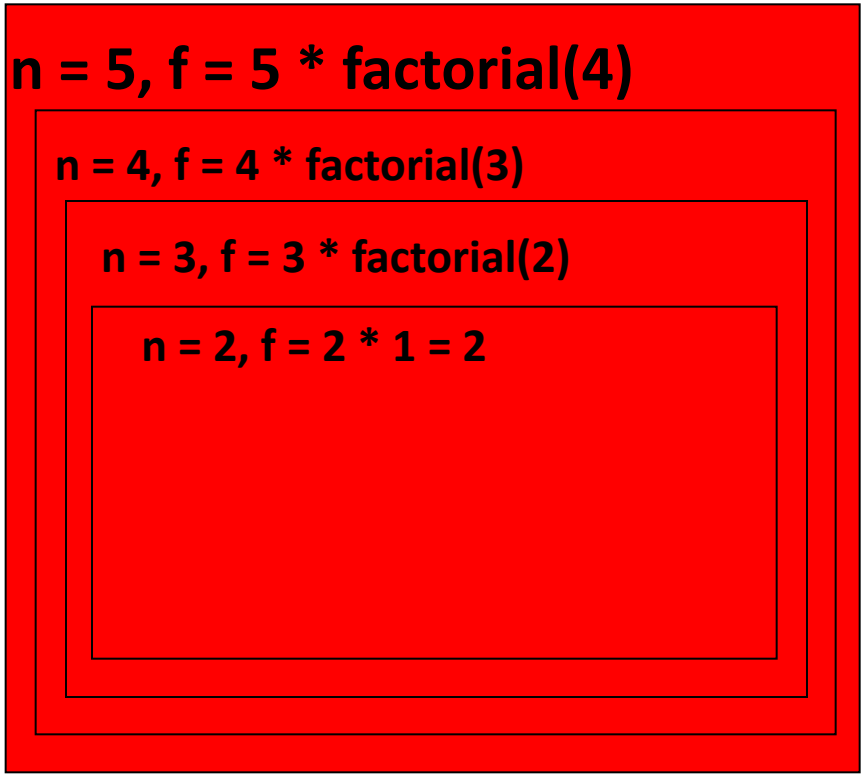
ΠΑΡΑΓΟΝΤΙΚΟ (2)



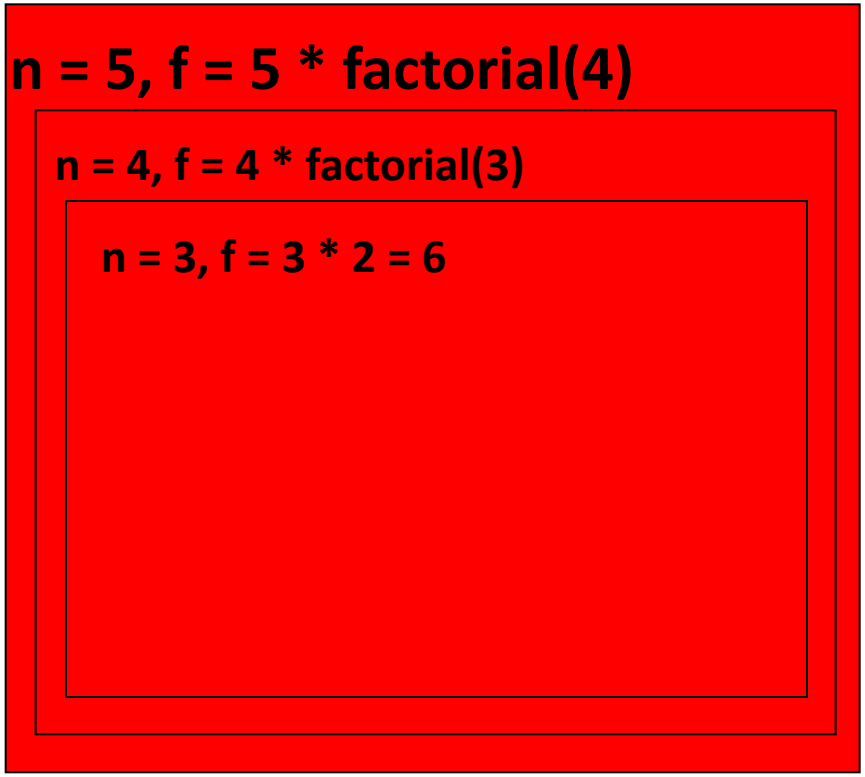
ΠΑΡΑΓΟΝΤΙΚΟ (3)



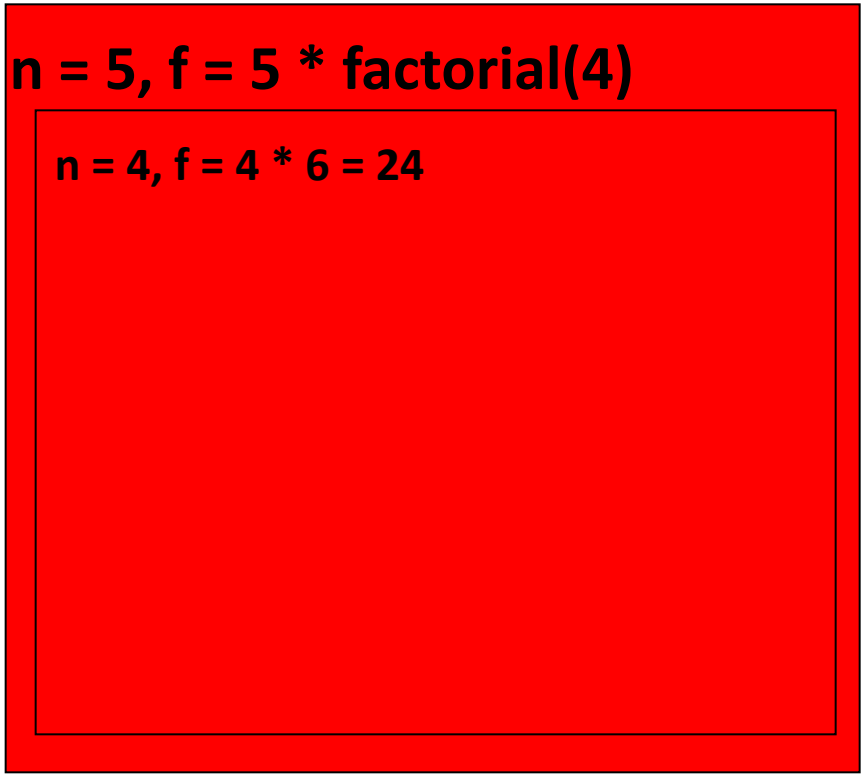
ΠΑΡΑΓΟΝΤΙΚΟ (4)



ΠΑΡΑΓΟΝΤΙΚΟ (5)



ΠΑΡΑΓΟΝΤΙΚΟ (6)



ΠΑΡΑΓΟΝΤΙΚΟ (7)

n = 5, f = 5 * 24 = 120

ΑΚΟΛΟΥΘΙΑ FIBONACCI (2)

$$f(n) = \begin{cases} n, \text{ αν } n \leq 1 \\ f(n-1) + f(n-2), \text{διαφορετικά} \end{cases}$$

- **Αναδρομική υλοποίηση**

Συνάρτηση Fibonacci(n), αναδρομική

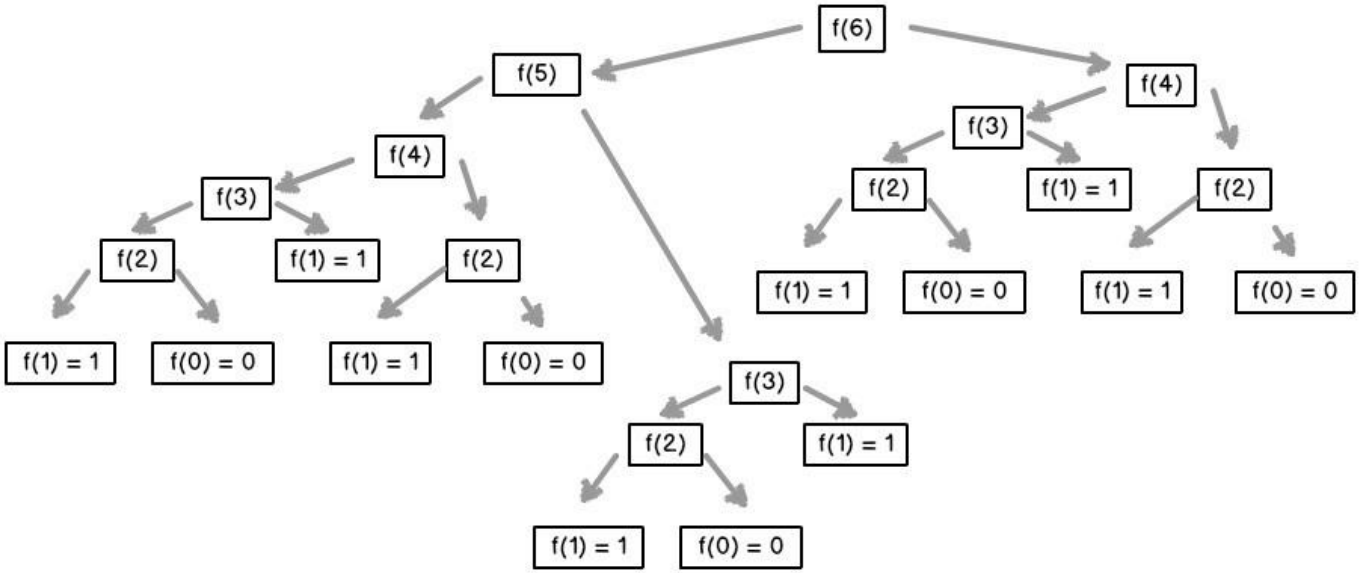
1. Είσοδος n
2. Αν n <= 1 τότε
    1. f ← n
3. Αλλιώς
    1. f ← Fibonacci(n − 1) + Fibonacci(n − 2)
4. Επιστροφή f

ΑΚΟΛΟΥΘΙΑ FIBONACCI (2)

- **Αναδρομική υλοποίηση**

Συνάρτηση Fibonacci(n), αναδρομική

1. Είσοδος n
2. Αν n <= 1 τότε
   1. f ← n
3. Αλλιώς
   1. f ← Fibonacci(n − 1) + Fibonacci(n − 2)
4. Επιστροφή f



ΠΑΡΑΔΕΙΓΜΑ

- **Είναι το παρακάτω πρόγραμμα ένα αναδρομικό πρόγραμμα;**

Συνάρτηση puzzle(n)

1. Είσοδος n
2. Αν n == 1 τότε
   1. f ← n
3. Αλλιώς Αν n % 2 == 0 τότε
   1. f ← puzzle(n /2)
4. Αλλιώς
   1. f ← puzzle(3*n + 1)
5. Επιστροφή f

```
puzzle(3)
  puzzle(10)
    puzzle(5)
      puzzle(16)
        puzzle(8)
          puzzle(4)
            puzzle(2)
              puzzle(1)
```

ΑΛΓΟΡΙΘΜΟΣ ΤΟΥ ΕΥΚΛΕΙΔΗ (1)

- **Εύρεση μέγιστου κοινού διαιρέτη (greatest common divisor)**
- **Ο "Μέγιστος κοινός διαιρέτης" (ΜΚΔ) του Ευκλείδη είναι ένας αλγόριθμος που βρίσκει τον μεγαλύτερο κοινό διαιρέτη δύο ή περισσότερων** ακεραίων, χρησιμοποιώντας τη μέθοδο των διαδοχικών υπολοίπων.

Βήματα του αλγόριθμου του Ευκλείδη

1. Διαίρεσε τον μεγαλύτερο αριθμό με τον μικρότερο και βρες το υπόλοιπο.
2. Αν το υπόλοιπο είναι 0, τότε ο μικρότερος αριθμός είναι ο ΜΚΔ.
3. Αν το υπόλοιπο δεν είναι 0, επανέλαβε τη διαδικασία:
    1. τώρα θα διαιρέσεις τον μικρότερο αριθμό με το υπόλοιπο που βρήκες στο προηγούμενο βήμα.
4. Συνέχισε αυτή τη διαδικασία μέχρι να βρεις υπόλοιπο 0.
5. Ο διαιρέτης που οδήγησε σε υπόλοιπο 0 είναι ο ΜΚΔ

ΑΛΓΟΡΙΘΜΟΣ ΤΟΥ ΕΥΚΛΕΙΔΗ (1)

Βήματα του αλγόριθμου του Ευκλείδη

1. Διαίρεσε τον μεγαλύτερο αριθμό με τον μικρότερο και βρες το υπόλοιπο.
2. Αν το υπόλοιπο είναι 0, τότε ο μικρότερος αριθμός είναι ο ΜΚΔ.
3. Αν το υπόλοιπο δεν είναι 0, επανέλαβε τη διαδικασία:
    1. τώρα θα διαιρέσεις τον μικρότερο αριθμό με το υπόλοιπο που βρήκες στο προηγούμενο βήμα.
4. Συνέχισε αυτή τη διαδικασία μέχρι να βρεις υπόλοιπο 0.
5. Ο διαιρέτης που οδήγησε σε υπόλοιπο 0 είναι ο ΜΚΔ

Αναδρομική υλοποίηση

```
Συνάρτηση ΜΚΔ(n1, n2)
1. Είσοδος n1, n2, με n1 > n2
2. Αν n1 % n2 == 0 τότε
     1. f ← n2
3. Αλλιώς
     1. f ← gcd(n2, n1%n2)
4. Επιστροφή f
```

ΑΛΓΟΡΙΘΜΟΣ ΤΟΥ ΕΥΚΛΕΙΔΗ (2)

- **Αναδρομική υλοποίηση**

Συνάρτηση gcd(n1, n2)

1. Είσοδος n1, n2, με n1 > n2
2. Αν n1 % n2 == 0 τότε
   1. f ← n2
3. Αλλιώς
   1. f ← gcd(n2, n1%n2)
4. Επιστροφή f

```
gcd(314159, 271828)
  gcd(271828, 42331)
    gcd(42331, 17842)
      gcd(17842, 6647)
        gcd(6647, 4458)
          gcd(4458, 2099)
            gcd(2099, 350)
              gcd(350, 349)
                gcd(349, 1)
                  gcd(1, 0)
```

ΔΙΑΙΡΕΙ ΚΑΙ ΒΑΣΙΛΕΥΕ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΤΜΗΜΑ ΣΤΑΤΙΣΤΙΚΗΣ

- **Οι αποτελεσματικότεροι αναδρομικοί αλγόριθμοι υλοποιούν την αρχή του διαίρει και βασίλευε**
  - **Διαιρούμε το πρόβλημα σε υποπροβλήματα**
  - **Λύνουμε τα υποπροβλήματα αναδρομικά**
  - **Συνθέτουμε τις λύσεις των υποπροβλημάτων για να λύσουμε το αρχικό πρόβλημα**

ΠΑΡΑΔΕΙΓΜΑ (1)

- **Εύρεση μέγιστου στοιχείου πίνακα**
- **Επαναληπτική υλοποίηση**

Συνάρτηση max(A)

1. Είσοδος: πίνακας A με n στοιχεία
2. max <- A[1]
3. Για i από 2 μέχρι n
    1. Αν A[i] > max τότε
        1. max <- A[i]
4. Επιστροφή max // μέγιστη τιμή και θέση της

ΠΑΡΑΔΕΙΓΜΑ (2)

- **Αναδρομική υλοποίηση**

Συνάρτηση max(A), αναδρομική

1. Είσοδος: πίνακας A, αρχή l, τέλος r
2. middle <- (l + r) // 2
3. Αν l = r τότε
    1. Επιστροφή A[l]
4. u <- max(A, l, m)
5. v <- max(A, m+1, r)
6. Αν u > v τότε
    1. Επιστροφή u
7. Αλλιώς
    1. Επιστροφή v

ΔΥΝΑΜΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

- **Δυναμικός προγραμματισμός (dynamic programming) είναι μια υπολογιστική μέθοδος που εφαρμόζεται όταν τα υποπροβλήματα δεν είναι ανεξάρτητα μεταξύ τους**

- **Ένας αλγόριθμους δυναμικού προγραμματισμού επιλύει μία φορά κάθε υποπρόβλημα και αποθηκεύει την λύση του σε έναν πίνακα, στον οποίο μπορεί να βρίσκει τη λύση κάθε φορά που συναντά κάποιο επιλυμένο υποπρόβλημα**

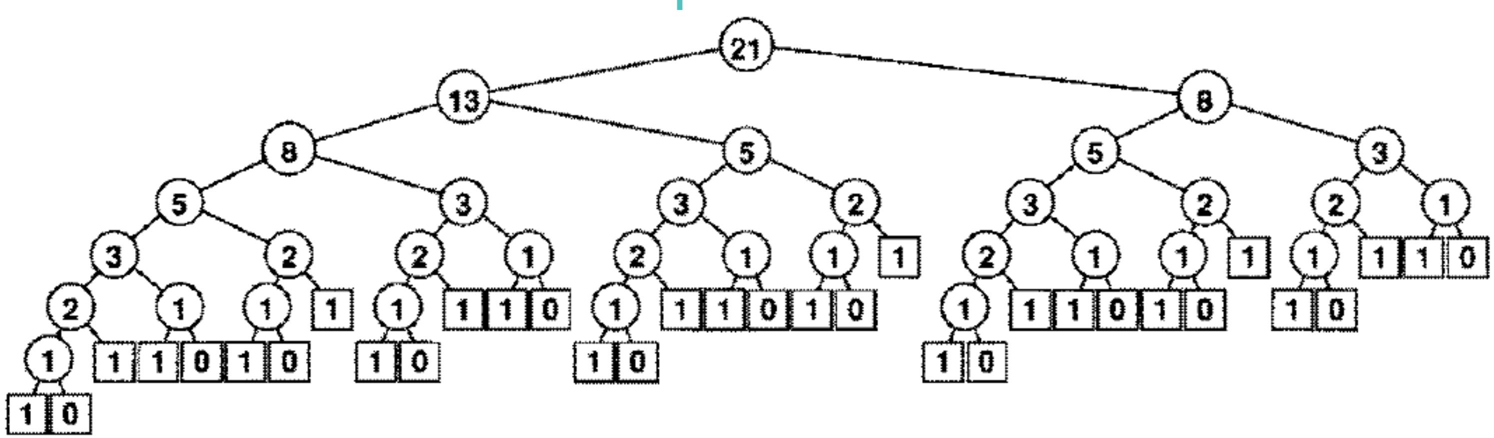
ΑΚΟΛΟΥΘΙΑ FIBONACCI (2)

$$f(n) = \begin{Bmatrix} n,\ \text{αν}\ n \leq 1 \\ f(n-1) + f(n-2),\ \text{διαφορετικα} \end{Bmatrix}$$

- **Αναδρομική υλοποίηση**

Συνάρτηση Fibonacci(n), αναδρομική

1. Είσοδος n
2. Αν n <= 1 τότε
   1. f ← n
3. Αλλιώς
   1. f ← Fibonacci(n − 1) + Fibonacci(n − 2)
4. Επιστροφή f



ΑΚΟΛΟΥΘΙΑ FIBONACCI (3)

$$f(n) = \begin{cases} n, \text{ αν } n \leq 1 \\ f(n-1) + f(n-2), \text{διαφορετικά} \end{cases}$$

- **Υλοποίηση δυναμικού προγραμματισμού**

Συνάρτηση Fibonacci(n), Δυν. Πρόγρ
1. Είσοδος αριθμός n, πίνακας Known
2. Αν Known[n] != -1 τότε
    1. Επιστροφή Known[n]
3. Αν n <= 1
    1. f ← n
4. Αλλιώς
    1. f ← Fibonacci(n − 1) + Fibonacci(n − 2)
5. Known[n] ← f
6. Επιστροφή f