ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ

**ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ**

# ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

*Υπολογισμός πολυπλοκότητας*

**Κωνσταντίνος Γιαννουτάκης**
*Επίκουρος Καθηγητής*

# ΑΣΚΗΣΗ ΠΡΟΗΓΟΥΜΕΝΗΣ ΔΙΑΛΕΞΗΣ

Έστω δύο αλγόριθμοι A και B με χρόνο εκτέλεσης $\boldsymbol{T_A(n) = 0.1n^2 logn}$ ms και $\boldsymbol{T_B(n) = 2.5n^2}$ ms. Ποιος αλγόριθμος είναι πιο αποδοτικός; Βρείτε το $\boldsymbol{n_0}$ όπου για κάθε $\boldsymbol{n > n_0}$ ο αλγόριθμος που επιλέξατε είναι πιο αποδοτικός από τον άλλο.

## Λύση

- Ο αλγόριθμος B είναι πιο αποδοτικός ως προς την $\mathcal{O}$ πολυπλοκότητα
$$\lim_{n\to\infty} \frac{T_A(n)}{T_B(n)} = \lim_{n\to\infty} \frac{0.1n^2 \log n}{2.5n^2} = \lim_{n\to\infty} \frac{0.1 \log n}{2.5} = \infty$$
- $\boldsymbol{T_B(n) \le T_A(n) \Rightarrow 2.5n^2 \le 0.1n^2 logn \Rightarrow logn \ge 25}$, άρα ο αλγόριθμος B είναι πιο αποδοτικός για $\boldsymbol{n \ge n_0 = 10^{25}}$

# ΑΣΚΗΣΗ ΠΡΟΗΓΟΥΜΕΝΗΣ ΔΙΑΛΕΞΗΣ

Δίνεται ο χρόνος εκτέλεσης ($T(n)$) οκτώ αλγορίθμων. Κατατάξτε τους σε αύξουσα σειρά ως προς την πολυπλοκότητά τους (πρώτα αυτός που έχει την μικρότερη πολυπλοκότητα).

| Αλγόριθμος | Πολυπλοκότητα | |
|---|---|---|
| 1 | $n\log^2 n$ | 2ος |
| 2 | $n^3$ | 6ος |
| 3 | $n^2 \log n$ | 5ος |
| 4 | $8^n$ | 8ος |
| 5 | $n^2$ | 4ος |
| 6 | $n^2/2$ | 3ος |
| 7 | $n \log n$ | 1ος |
| 8 | $2^n$ | 7ος |

# ΣΥΝΟΨΗ ΔΙΑΛΕΞΗΣ

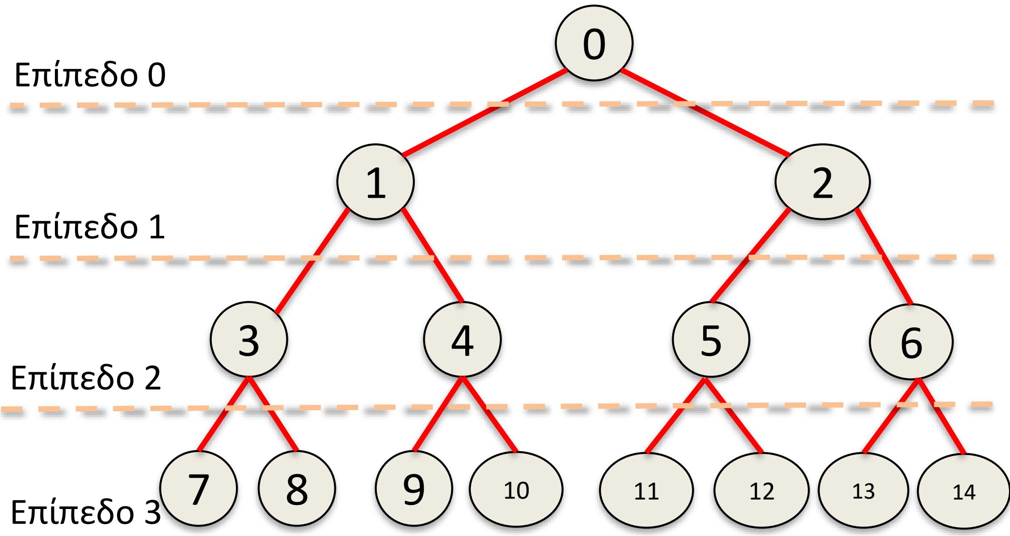
- Συνηθισμένοι χρόνοι εκτέλεσης
- Παραδείγματα Χρόνων Εκτέλεσης και Πολυπλοκότητας

# ΣΥΝΗΘΙΣΜΕΝΟΙ ΧΡΟΝΟΙ ΕΚΤΕΛΕΣΗΣ

# ΛΟΓΑΡΙΘΜΙΚΟΣ ΧΡΟΝΟΣ $\mathcal{O}(\lg n)$

Το ύψος $h$ ενός πλήρους δυαδικού δένδρου με $n$ κόμβους είναι $\lg n$. Μπορούμε εύκολα να το αποδείξουμε μετρώντας τους κόμβους ξεκινώντας από τη ρίζα, θεωρώντας ότι κάθε επίπεδο έχει το μέγιστο αριθμό κόμβων.

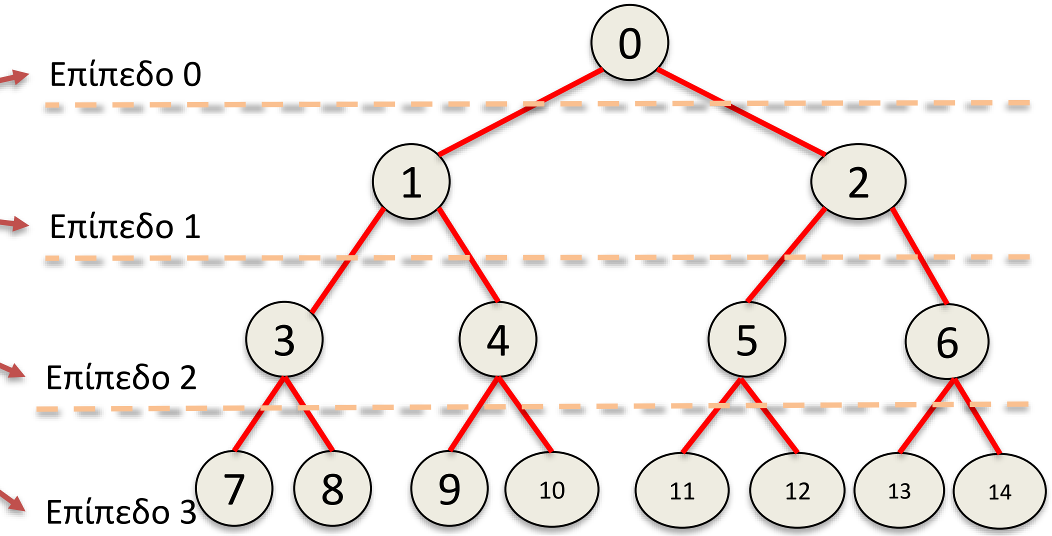
$n = 1 + 2 + 4 + \cdots + 2^{h-1} + 2^h = 2^{h+1} - 1 \Rightarrow$
$n \geq 2^h \Rightarrow h \leq \lg n$
άρα $\boldsymbol{h} = \boldsymbol{\mathcal{O}(\lg n)}$



# ΛΟΓΑΡΙΘΜΙΚΟΣ ΧΡΟΝΟΣ $\mathcal{O}(\lg n)$

```
for(i = 1; i <= n; i = 2 * i) {
        …
}
```

| Επανάληψη | Τιμή i |
|---|---|
| 1η | 1 |
| 2η | 2 |
| 3η | 4 |
| 4η | 8 |
| 5η | 16 |
| ⋮ | ⋮ |




Η **`for`** παράγει τιμές για το i παρόμοιες με το πλήθος των κόμβων ενός δυαδικού δένδρου σε κάθε επίπεδο, δηλ. 1, 2, 4, 8, 16, …

Το πλήθος των επαναλήψεων που θα εκτελεστούν ισούται με το ύψος του πλήρους δυαδικού δένδρου με $n$ κόμβους, δηλαδή $\boldsymbol{\mathcal{O}(\lg n)}$.

# ΓΡΑΜΜΙΚΟΣ ΧΡΟΝΟΣ $O(n)$

Ο χρόνος εκτέλεσης είναι το πολύ ένας σταθερός παράγοντας επί το μέγεθος της εισόδου.

Υπολογισμός μέγιστου αριθμού από *n* αριθμούς $a_1, \cdots, a_n$.

```
max ← a1
for i = 2 to n {
   if (ai > max)
      max ← ai
}
```

Διαπέραση/προσπέλαση μονοδιάστατου πίνακα

```
for(i = 0; i < n; i++) {
      …
}
```

# ΓΡΑΜΜΙΚΟΣ ΧΡΟΝΟΣ $O(n)$

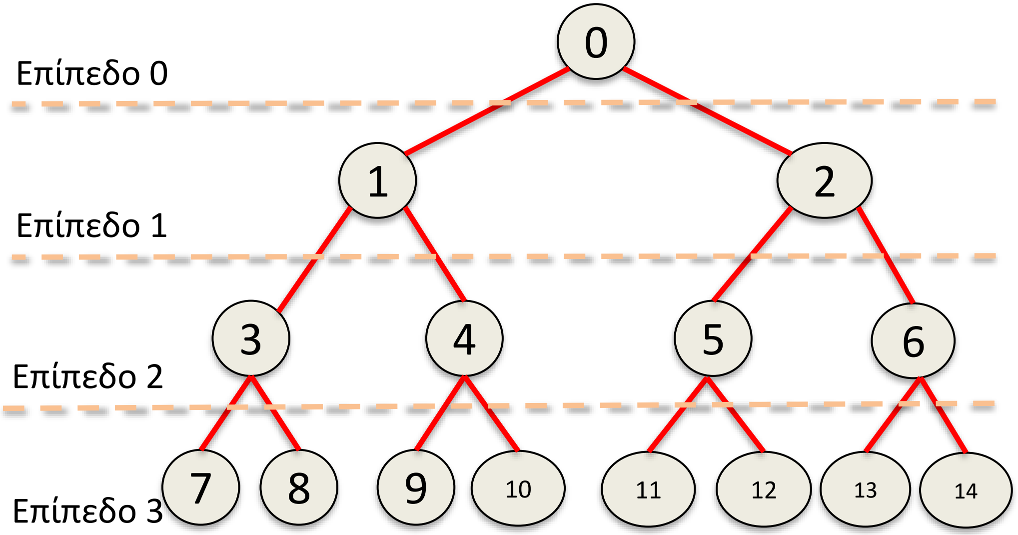
Συγχωνεύστε 2 ταξινομημένες λίστες $A = a_1, a_2, \cdots, a_n$ με $B = b_1, b_2, \cdots, b_n$.

```
i = 1, j = 1
while (both lists are nonempty) {
   if (ai ≤ bj) append ai to output list and increment i
   else append bj to output list and increment j
}
append remainder of nonempty list to output list
```

Η συγχώνευση δύο λιστών μεγέθους $n$ απαιτεί χρόνο $\boldsymbol{O(n)}$.

# ΓΡΑΜΜΟΛΟΓΑΡΙΘΜΙΚΟΣ ΧΡΟΝΟΣ $O(n \lg n)$

```
for(i = 0; i < n; i++) {
        for(j = 1; j <= n; j = 2 * j) {
            …
    }
}
```




Η εσωτερική **for** παράγει τιμές για το j παρόμοιες με το πλήθος των κόμβων ενός δυαδικού δένδρου σε κάθε επίπεδο, δηλ. 1, 2, 4, 8, …

# ΤΕΤΡΑΓΩΝΙΚΟΣ ΧΡΟΝΟΣ $\mathcal{O}(n^2)$

Θεωρώντας μια λίστα από $n$ σημεία στο επίπεδο $(x_1, y_1), \cdots, (x_n, y_n)$, βρες το κοντινότερο ζεύγος. Υπολογισμός ευκλείδειας απόστασης όλων των ζευγαριών σημείων.

```
min ← (x₁ - x₂)² + (y₁ - y₂)²
for i = 1 to n {
    for j = i+1 to n {
        d ← (xᵢ - xⱼ)² + (yᵢ - yⱼ)²
        if (d < min)
            min ← d
    }
}
```

# ΚΥΒΙΚΟΣ ΧΡΟΝΟΣ $O(n^3)$

## Διαπέραση τρισδιάστατου πίνακα

```
for(i = 0; i < n; i++) {
        for(j = 0; j < n; j++) {
              for(k = 0; k < n; k++) {
                  …
              }
        }
}
```

Κάθε for εκτελείται n φορές. Επειδή οι for είναι εμφωλευμένες τότε σχηματίζεται γινόμενο και ο συνολικός χρόνος εκτέλεσης (βήματα) είναι $\boldsymbol{n \cdot n \cdot n = O(n^3)}$

# ΠΑΡΑΔΕΙΓΜΑΤΑ

- Χρόνοι Εκτέλεσης
- Πολυπλοκότητα

# ΠΑΡΑΔΕΙΓΜΑ 1

```
for(i = 0; i < n; i++) {
    for(j = 0; j < n; j++){
      do something;
    }
    for(k = 0; k < n; k++){
      do something;
    }
}
```

$n$ φορές

+

$n$ φορές

$n \cdot 2$n φορές

Οι εσωτερικές **for** εκτελούνται στη σειρά. Δεν ισχύει ο κανόνας του γινομένου αλλά ο κανόνας της πρόσθεσης γιατί οι δύο **for** είναι στη σειρά. Επομένως, ο συνολικός χρόνος εκτέλεσης (βήματα) είναι $n \cdot (n + n) = 2n^2 = \mathcal{O}(n^2)$.

# ΠΑΡΑΔΕΙΓΜΑ 2

```
for(i = 0; i < n; i++) {
    for(j = 1; j <= n; j = 2 * j) {
        …
    }
}
```

$n$ φορές

Όταν έχουμε πολλαπλασιαστικό βήμα τότε έχουμε λογαριθμική πολυπλοκότητα

$\lg n$

Ο πολλαπλασιαστικός όρος είναι το 2 και επομένως έχουμε λογάριθμο με βάση το 2. Αν ήταν 3 τότε θα είχαμε $\log_3 n$, κ.ο.κ.

Ο συνολικός χρόνος εκτέλεσης είναι $n \lg n = \mathcal{O}(n \lg n)$.

# ΠΑΡΑΔΕΙΓΜΑ 3

```
for(int i = n/3; i < n; i++) {
    for(int j = n/2; j <= n; j = j+2) {
        printf(“\n”);
    }
}
int k=0;
while (k < n/2) {
    for (int l = k+1; l <= n; l=l*2) {
        printf(“\n”);
    }
    k = k+2;
}
```

$\mathcal{O}(n)$ φορές

$\mathcal{O}(n)$ φορές

Σταθερός χρόνος εκτέλεσης $\mathcal{O}(1)$

$\mathcal{O}\ (n^2)$ φορές

$\mathcal{O}(n)$ φορές

$\mathcal{O}(\lg n)$ φορές

Σταθερός χρόνος εκτέλεσης $\mathcal{O}(1)$

Σταθερός χρόνος εκτέλεσης $\mathcal{O}(1)$

$\mathcal{O}(n(\lg n + 1))$ φορές

Η πολυπλοκότητα όλου του αλγορίθμου είναι $\mathcal{O}(n^2) + \mathcal{O}(n \lg n + n) = \mathcal{O}(n^2)$

# ΠΑΡΑΔΕΙΓΜΑ 4

```
for(i = n/2 - 1; i < n; i++) {
    for(j = 0; j < n / 2; j++) {
        for(k = 1; k <= n; k = 2 * k) {
            …
        }
    }
}
```

Ποια είναι η χρονική πολυπλοκότητα του αλγορίθμου;

# ΠΑΡΑΔΕΙΓΜΑ 4

```
for(i = n/2 - 1; i < n; i++) {
    for(j = 0; j < n / 2; j++) {
        for(k = 1; k <= n; k = 2 * k) {
            …
        }
    }
}
```

$n - 1 - \left(\frac{n}{2} - 1\right) + 1 = \frac{n}{2} + 1$ φορές

$\frac{n}{2} - 1 - 0 + 1 = \frac{n}{2}$ φορές

$\lg n$ φορές

$$\left(\frac{n}{2} + 1\right)\frac{n}{2}\lg n = \boldsymbol{O}\left(\boldsymbol{n^2 \lg n}\right)$$

# ΠΑΡΑΔΕΙΓΜΑ 5

```
for(i = n/2 - 1; i < n; i++) {
    for(j = 1; j <= n; j = 2 * j) {
        for(k = 1; k <= n; k = 2 * k) {
            …
        }
    }
}
```

Ποια είναι η χρονική πολυπλοκότητα του αλγορίθμου;

$$O(n\lg^2 n)$$

# ΕΣΩΤΕΡΙΚΟ ΓΙΝΟΜΕΝΟ

Εάν $u = [u_1, u_2, \cdots, u_n]^T$ και $v = [v_1, v_2, \cdots, v_n]^T$ διανύσματα του $\mathbb{R}^3$. Ορίζουμε το εσωτερικό γινόμενο των διανυσμάτων τη ποσότητα:

$$u \cdot v = u_1\, v_1 + u_2\, v_2 + \cdots + u_n\, v_n$$

```
dot_prod=0.0;
for(i = 0; i < n; i++)
   dot_prod+=u[i]*v[i];
```

# ΕΣΩΤΕΡΙΚΟ ΓΙΝΟΜΕΝΟ

Εάν $u = [u_1, u_2, \cdots, u_n]^T$ και $v = [v_1, v_2, \cdots, v_n]^T$ διανύσματα του $\mathbb{R}^3$.
Ορίζουμε το εσωτερικό γινόμενο των διανυσμάτων τη ποσότητα:

$$u \cdot v = u_1\, v_1 + u_2\, v_2 + \cdots + u_n\, v_n$$

```
dot_prod=0.0;
for(i = 0; i < n; i++)
    dot_prod+=u[i]*v[i];
```

1 πολλαπλασιασμός
1 άθροισμα

$n$ φορές

***Πλήθος πράξεων***: $n$ πολλαπλασιασμοί και $n$ αθροίσματα
***Ασυμπτωτική πολυπλοκότητα***: $\boldsymbol{O}(\boldsymbol{n})$ πολλαπλασιασμοί και $\boldsymbol{O}(\boldsymbol{n})$ αθροίσματα

# ΓΙΝΟΜΕΝΟ ΠΙΝΑΚΩΝ

Εάν $A, B \in M_{n,n}$, τότε ορίζουμε το γινόμενό τους

$$C \in M_{n,n}\colon\ c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$$

```
for(i = 0; i < n; i++)
 for(j = 0; j < n; j++)
   for(k = 0; k < n; k++)
      c[i][j]+=a[i][k]*b[k][j];
```

1 πολλαπλασιασμός
1 άθροισμα

$n$ φορές

$n$ φορές

$n$ φορές

***Πλήθος πράξεων***: $n^3$ πολλαπλασιασμοί και $n^3$ αθροίσματα

***Ασυμπτωτική πολυπλοκότητα***: $\boldsymbol{O}(\boldsymbol{n^3})$ πολλαπλασιασμοί

# ΠΑΡΑΓΟΝΤΙΚΟ

Παραγοντικό ενός μη-αρνητικού ακεραίου $n$ είναι:

$$n! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot n$$

```
long factorial( int n){
    if(n==0)
        return 1;
    else
        return (factorial(n-1)*n)
}
```

Αριθμός Πολλαπλασιασμών:

$$T(n) = \underbrace{T(n-1)}_{\text{Χρόνος υπολογισμού } T(n-1)} + \underbrace{1}_{\text{Χρόνος πολ/σμού του } \texttt{factorial(n-1)} \text{ με το } \texttt{n}}$$

Πως θα λύσουμε την αναδρομική σχέση;

# ΠΑΡΑΓΟΝΤΙΚΟ

Παραγοντικό ενός μη-αρνητικού ακεραίου $n$ είναι:

$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$$

```
long factorial( int n){
    if(n==0)
        return 1;
    else
        return (factorial(n-1)*n)
}
```

Αριθμός Πολλαπλασιασμών:

$$T(n) = \underbrace{T(n-1)}_{\substack{\text{Χρόνος υπολογισμού} \\ T(n-1)}} + \underbrace{1}_{\substack{\text{Χρόνος πολ/σμού του} \\ \texttt{factorial(n-1)} \text{ με το } \texttt{n}}}$$

**Μέθοδος της Επανάληψης:**

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

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

$$[T(n-3)+1] + 2 = T(n-3) + 3 =$$

…

$$T(0) + n = n$$

# ΤΑΞΙΝΟΜΗΣΗ ΕΠΙΛΟΓΗΣ (SELECTION SORT)

- Ταξινόμηση ενός πίνακα με την ταξινόμηση επιλογής
  - Είσοδος: πίνακας A[0…n-1] (πίνακας προς ταξινόμηση)
  - Έξοδος: A[0…n-1] (ταξινομημένος πίνακας)

Στιγμιότυπο: 89, 45, 68, 90, 29, 34,17

Εκτέλεση:

```
|89  45  68  90  29  34  17
17|  45  68  90  29  34  89
17  29|  68  90  45  34  89
17  29  34|  90  45  68  89
17  29  34  45|  90  68  89
17  29  34  45  68|  90  89
17  29  34  45  68  89|  90
```

```
void selectionsort(int t[], int n)  {
      for (int i = 0; i < n-1; i++)  {
         // Εύρεση του μικρότερου στοιχείου του πίνακα
           int min = i;
           for (int j = i+1; j < n; j++) {
                if (t[j] < t[min])
                     min = j;
           }
           // Εναλλαγή του min στοιχείου με το πρώτο
           int tmp = t[min];
           t[min] = t[i];
           t[i] = tmp;
      }
}
```

# ΤΑΞΙΝΟΜΗΣΗ ΕΠΙΛΟΓΗΣ (SELECTION SORT)

- Δοθέντος μιας λίστας αριθμών, **πάρε το τρέχον στοιχείο και αντάλλαξέ το με το μικρότερο στη δεξιά πλευρά** του τρέχοντος

| | | | | | | |
|---|---|---|---|---|---|---|
| 8 | 5 | 7 | 1 | 9 | 3 | $(\boldsymbol{n}$-1) βήματα για την εύρεση του μικρότερου |
| 1 | 5 | 7 | 8 | 9 | 3 | $(\boldsymbol{n}$-2) βήματα για την εύρεση του μικρότερου |
| 1 | 3 | 7 | 8 | 9 | 5 | $(\boldsymbol{n}$-3) βήματα για την εύρεση του μικρότερου |
| 1 | 3 | 5 | 8 | 9 | 7 | 2 |
| 1 | 3 | 5 | 7 | 9 | 8 | 1 |
| 1 | 3 | 5 | 7 | 8 | 9 | 0 |

Συνολικά χρειαζόμαστε $\boldsymbol{n}(\boldsymbol{n}\text{-}1)/2$ βήματα άρα $\boldsymbol{O}(\boldsymbol{n}^2)$

# ΤΑΞΙΝΟΜΗΣΗ ΕΠΙΛΟΓΗΣ (SELECTION SORT)

- Η βασική πράξη του αλγορίθμου είναι η σύγκριση των στοιχείων A[j] < A[min]. Το πλήθος των φορών που εκτελείται είναι:

Εφαρμογή του τύπου

$$\sum_{i=j}^{n} 1 = n - j + 1$$

$$T(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1]$$

$$= \sum_{i=0}^{n-2} (n-i-1) = \sum_{k=1}^{n-1} k = \frac{(n-1)n}{2} = \frac{1}{2}n^2 - \frac{1}{2}n \rightarrow$$

$$T(n) = \mathcal{O}(\mathrm{n}^2)$$

**Κωνσταντίνος Γιαννουτάκης**

Επίκ. Καθηγητής

kgiannou@uom.edu.gr

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ