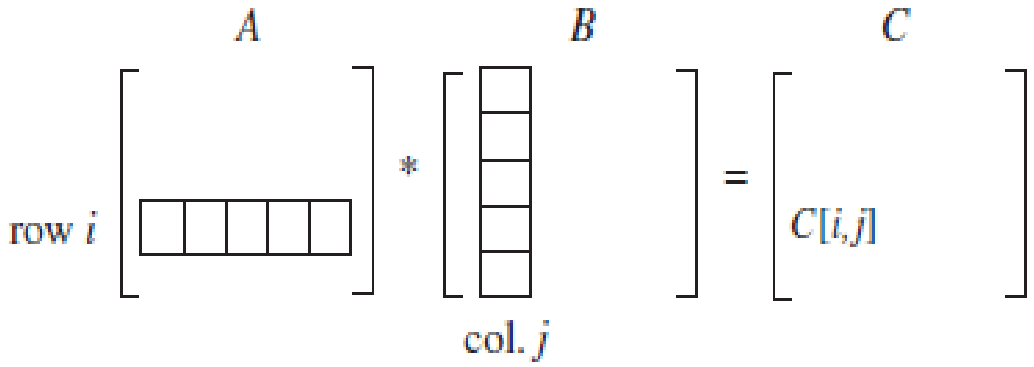
ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

# ΑΛΓΟΡΙΘΜΟΙ

# Πολ/μος Δισδιάστατων Πινάκων (1/3)




$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

$$C[i, j] = A[i, 0]B[0, j] + \cdots + A[i, k]B[k, j] + \cdots + A[i, n-1]B[n-1, j]$$

**ALGORITHM** *MatrixMultiplication*$(A[0..n-1, 0..n-1], B[0..n-1, 0..n-1])$

```
//Input: Two n × n matrices A and B
//Output: Matrix C = AB
for i ← 0 to n − 1 do
    for j ← 0 to n − 1 do
        C[i, j] ← 0.0
        for k ← 0 to n − 1 do
            C[i, j] ← C[i, j] + A[i, k] ∗ B[k, j]
return C
```

# Πολ/μος Δισδιάστατων Πινάκων (2/3)

Στο εσωτερικό loop υπάρχουν πολ/μοι και προσθέσεις

Παραδοσιακά ο πολ/μος θα θεωρηθεί ως η βασική πράξη του αλγορίθμου

Υπολογίζουμε το συνολικό αριθμό των πολ/μων

Η μεταβλητή k πάει από το 0 στο n-1 μέσα στο εσωτερικό loop

Συνεπώς το πλήθος των πολ/μων στο εσωτερικό loop είναι

$$\sum_{k=0}^{n-1} 1$$

και το πλήθος όλων των πολ/μων $$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1$$

# Πολ/μος Δισδιάστατων Πινάκων (3/3)

Οι υπολογισμοί έχουν ως εξής:

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1 = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n = \sum_{i=0}^{n-1} n^2 = n^3$$

# Εύρεση Ψηφίων

Η πιο συχνή λειτουργία είναι η σύγκριση στο while αφού θα εκτελείται πάντα 1 φορά παραπάνω από το πλήθος επαναλήψεων

Οι τιμές του n δεν είναι πάντα οι ίδιες

Οι τιμή της n είναι περίπου μισή σε σχέση με την προηγούμενη

Συνεπώς, το πλήθος των επαναλήψεων θα είναι $\log_2 n + 1$

```
ALGORITHM Binary(n)
    //Input: A positive decimal integer n
    //Output: The number of binary digits in n's binary representation
    count ← 1
    while n > 1 do
        count ← count + 1
        n ← ⌊n/2⌋
    return count
```

# Ασκήσεις (1/5)

Βρείτε την πολυπλοκότητα του ακόλουθου αλγορίθμου:

```
s ← 0
for i ← 1 to n − 4
    for j ← i to i + 4
        for k ← i to j
            s ← s + a_i
```

**Λύση**

Το εξωτερικό loop εκτελείται n-4 φορές.

Το μεσαίο loop εκτελείται 5 φορές.

Το εσωτερικό loop εκτελείται 1, 2, 3, 4 ή 5 φορές για τις τιμές του j ίσες με I, i+1, i+2, i+3 και i+4.

Έτσι η πολυπλοκότητα είναι ίση με Θ(n)

# Ασκήσεις (2/5)

Βρείτε την πολυπλοκότητα του ακόλουθου αλγορίθμου:

```
a ← 0
for i ← 1 to n
    for j ← i to n
        a ← a + 1
return a
```

**Λύση**

$$\sum_{i=1}^{n}\sum_{j=i}^{n} 1 = \sum_{i=1}^{n}\Big(\underbrace{\sum_{j=1}^{n} 1}_{n} - \underbrace{\sum_{j=1}^{i-1} 1}_{i-1}\Big) = \sum_{i=1}^{n}(n-i+1) = \sum_{i=1}^{n}(n+1) - \sum_{i=1}^{n} i =$$

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

# Ασκήσεις (3/5)

Βρείτε την πολυπλοκότητα του ακόλουθου αλγορίθμου:

```
a ← 0
for i ← 1 to n
    for j ← i+1 to n
        a ← a+1
return a
```

**Λύση**

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n}\Big(\underbrace{\sum_{j=1}^{n} 1}_{n} - \underbrace{\sum_{j=1}^{i} 1}_{i}\Big) = \sum_{i=1}^{n}(n-i) = \sum_{i=1}^{n}(n) - \sum_{i=1}^{n} i =$$

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

# Ασκήσεις (4/5)

Βρείτε την πολυπλοκότητα του ακόλουθου αλγορίθμου:

```
a ← 0
for i ← 1 to n−1
    for j ← i+1 to n
        for k ← 1 to j
            a ← a+1
return a
```

**Λύση**

$$\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\sum_{k=1}^{j}1=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}j=\sum_{i=1}^{n-1}\left(\sum_{j=1}^{n}j-\sum_{j=1}^{i}j\right)=$$

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

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

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

Όμως $\sum_{i=1}^{n}i^2=\frac{1}{6}n(n+1)(2n+1)$ και $\sum_{i=1}^{n-1}i^2=\frac{1}{6}(n-1)n(2n-1)$

# Ασκήσεις (5/5)

Βρείτε την πολυπλοκότητα του ακόλουθου αλγορίθμου:

```
a ← 0
for i ← 1 to n − 1
    for j ← i + 1 to n
        for k ← 1 to j
            a ← a + 1
return a
```

**Λύση (συνέχεια)**

$$\frac{1}{2}n(n-1)(n+1) - \frac{1}{12}n(n-1)(2n-1) - \frac{1}{4}n(n-1) =$$
$$\frac{1}{2}n(n-1)\left(n+1-\frac{1}{6}(2n-1)-\frac{1}{2}\right) =$$
$$\frac{1}{12}n(n-1)(6n+3-2n+1) = \frac{1}{12}n(n-1)(4n+4) =$$
$$\frac{1}{3}n(n-1)(n+1)$$

Άρα ο αλγόριθμος είναι $\Theta(n^3)$

# Αναδρομικά Προβλήματα

# Διαίρει και Βασίλευε

Τρία βήματα για την επίλυση:

- **Διαίρει**: διαιρούμε το πρόβλημα σε ένα πλήθος υπο-προβλημάτων
- **Βασίλευε**: επιλύουμε τα προβλήματα
- **Συνδύασε**: συνδυάζουμε τις λύσεις σε μια γενική λύση για το αρχικό πρόβλημα

Βασιζόμαστε σε αναδρομικές σχέσεις

Παράδειγμα:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

# Μέθοδοι Επίλυσης Αναδρομών

Τρείς μέθοδοι επίλυσης

- **Backward substitution**
- **Δένδρα αναδρομής**
- **Η master μέθοδος**

# Δυαδική Αναζήτηση (1/6)

Αναζήτησε σε ταξινομημένο πίνακα μια τιμή x

```
int binarySearch (int A[], int x, int left, int right){
   int mid;
   if (left>right) return -1;
   mid=(left+right)/2;
   if (A[mid]==x) return mid;
   if (A[mid]<x) return binarySearch(A,x,mid+1,right);
   else return binarySearch(A,x,left,mid-1);
}

void main(){
   int A[10]={0,1,2,3,4,5,6,7,8,9};
   cout << binarySearch(A,8,0,9) << endl;
   cout << binarySearch(A,10,0,9) << endl;
}
```

Find: 9

Array 3 5 7 8 9 12 15

# Δυαδική Αναζήτηση (2/6)

Χειρότερη περίπτωση: Το στοιχείο δεν βρίσκεται στον πίνακα

Μας ενδιαφέρει το πλήθος συγκρίσεων που γίνονται.

Σε κάθε κλήση της αναδρομής ο αλγόριθμος κάνει μια σύγκριση και κατόπιν συνεχίζει για το μισό πίνακα που απομένει.

Έστω T(N) η πολυπλοκότητα για μέγεθος πίνακα N.

Τότε έχω $T(N) <= T(\lfloor N/2 \rfloor) + 1$ με αρχική τιμή T(1)=1

Για τη λύση της αναδρομής διακρίνω 2 περιπτώσεις:

-Το N είναι δύναμη του 2

-Το N δεν είναι

# Δυαδική Αναζήτηση (3/6)

Αν το N είναι δύναμη του 2 τότε έχω:

$T(N) <= T(N/2) +1$

$T(N/2) <= T(N/4) +1$

$T(N/4) <= T(N/8) + 1$

...

$T(N/2^{logN-1}) <= T(N/2^{logn}) + 1$

$T(n) = 1 + T(n/2) = 1 + 1 + T(n/4) = 1 + 1 + 1 + T(n/8)$ .... $T(n) = k + T(n/2^k)$

με $k = \log n$, έχω $n/(2^{\log n})=n/n = 1$ και παίρνω $T(N) <= logN + 1$

# Δυαδική Αναζήτηση (4/6)

Αν το N δεν είναι δύναμη του 2 αποδεικνύω επαγωγικά ότι T(N)<= logN + 1

-Για N=3 ισχύει αφού χρειάζομαι 2 συγκρίσεις (T(N)=2 )

-Έστω ότι ισχύει για k<N, Θα δείξω για k=N

Έχω: $T(N) <= T(\lfloor N/2 \rfloor) + 1$

$<=$ (από επαγωγική υπόθεση) $\lfloor \log \lfloor N/2 \rfloor \rfloor + 1 + 1$

$<= \lfloor \log((N-1)/2) \rfloor + 1 + 1$

$<= \lfloor \log(N-1) \rfloor + 1$

$<= \lfloor \log N \rfloor + 1$

Επομένως ο αλγόριθμος στη χειρότερη περίπτωση (όπου δε βρίσκουμε αυτό που αναζητούμε) έχει πολυπλοκότητα **O(logN)**.

# Δυαδική Αναζήτηση (5/6)

Μέση Περίπτωση:

Στη μέση περίπτωση το στοιχείο μπορεί να βρίσκεται στον πίνακα σε οποιαδήποτε θέση του ή και να μη βρίσκεται.

-Υπάρχουν 2N + 1 καταστάσεις στις οποίες μπορεί να τερματίσει η αναζήτηση.

-Οι N αντιστοιχούν σε επιτυχή αναζήτηση (μία για κάθε θέση του πίνακα που μπορεί να βρεθεί το στοιχείο)

-Οι υπόλοιπες N+1 αντιστοιχούν στις θέσεις που μπορεί να τερματίσει η αναζήτηση ανεπιτυχώς (N-2 ενδιάμεσα των στοιχείων και 2 στις άκρες του πίνακα)

Αν θεωρήσω ισοπίθανα τα 2N+1 ενδεχόμενα αθροίζοντας προκύπτει πάλι O(logN)

# Δυαδική Αναζήτηση (6/6)

Μη αναδρομικός αλγόριθμος:

```
int iterbin2(T[n], x) {
        if (n=1 || x<T[0])
                return 0;
        i=0; j=n-1;
        while (i<j) {
                //αν T[i]≤x<T[j+1]
                int k=(i+j+1) / 2;
                if (x<T[k])
                        j=k-1;
                else if (x≥T[k+1])
                        i=k+1;
                else {i=k; j=k;}
        }
        return i;
}
```