
ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

# ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

*Ωμή βία*

**Κωνσταντίνος Γιαννουτάκης**
*Επίκουρος Καθηγητής*

# ΑΣΚΗΣΗ ΠΡΟΗΓΟΥΜΕΝΗΣ ΔΙΑΛΕΞΗΣ

Προσπαθήστε να βρείτε τα φράγματα των ακόλουθων αναδρομικών σχέσεων:

$$\boldsymbol{T(n) = T\left(\frac{n}{4}\right) + \sqrt{n}}$$

- Έχουμε $a=1$, $b=4$ και $f(n) = \sqrt{n}$. Επομένως $n^{log_b\, a} = n^{log_4\, 1} = n^0 = 1$ με $f(n) > n^{log_b\, a}$.
- Σκεφτόμαστε την Περίπτωση 3:
  - $f(n) = \Omega(n^{log_b\, a+\varepsilon})$ όπου $\varepsilon = 1/2$.
  - Συνθήκη ομαλότητας: $af\left(\frac{n}{b}\right) = \sqrt{\frac{n}{4}} = \frac{1}{2}\sqrt{n} \leq \frac{1}{2}f(n)$

Άρα $T(n) = \Theta(\sqrt{n})$

# ΑΣΚΗΣΗ ΠΡΟΗΓΟΥΜΕΝΗΣ ΔΙΑΛΕΞΗΣ

Προσπαθήστε να βρείτε τα φράγματα των ακόλουθων αναδρομικών σχέσεων:

$$\boldsymbol{T(n) = 2T\left(\frac{n}{2}\right) + n}$$

- Έχουμε $a=2$, $b=2$ και $f(n) = n$. Επομένως $n^{log_b\, \alpha} = n^{log_2\, 2} = n^1 = n$ με $f(n) = n^{log_b\, \alpha}$.
- Άρα βρισκόμαστε στην Περίπτωση 2:

$$T(n) = \Theta(n\log n)$$

# ΣΥΝΟΨΗ ΔΙΑΛΕΞΗΣ

- Αλγόριθμοι Ωμής Βίας

# ΑΛΓΟΡΙΘΜΟΙ ΩΜΗΣ ΒΙΑΣ

- *Σειριακή Αναζήτηση*
- *Ταξινόμηση Φυσαλίδας*
- *Υπολογισμός Πολυωνύμου*
- *Κοντινότερο Ζεύγος Σημείων*
- *Ταίριασμα Συμβολοσειράς*

# ΩΜΗ ΒΙΑ (BRUTE FORCE)

- Εξαντλεί όλες τις δυνατές λύσεις του προβλήματος δοκιμάζοντας κάθε δυνατό συνδυασμό.
- Βασίζεται στον ορισμό του προβλήματος.

**Ωμή Βία**

Τετριμμένος αλγόριθμος επίλυσης προβλήματος – έλεγχος όλων των πιθανών λύσεων (στο χώρο όλων των δυνατών λύσεων του προβλήματος)

# ΣΕΙΡΙΑΚΗ ΑΝΑΖΗΤΗΣΗ (LINEAR SEARCH)

**Αναζήτηση**
Εύρεση στοιχείου σε μη ταξινομημένη λίστα

Σε ποια θέση βρίσκεται το στοιχείο ‘-2’;

| 5 | -3 | 6 | 22 | 0 | -5 | 2 | 45 | -9 | 1 | -2 | 3 | -4 | 66 | 10 | 12 | -8 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

# ΚΩΔΙΚΑΣ ΣΕ C

```
int linearSearch(int[] arr, int n, int x) {
  for (int i = 0; i < n; i++) {
    if (arr[i] == x)
      return i;
  }
  return -1;
}
```

Ποιος είναι ο αριθμός των **βασικών πράξεων** που απαιτούνται για την εύρεση ή μη εύρεση του στοιχείου;

# ΑΝΑΛΥΣΗ ΠΟΛΥΠΛΟΚΟΤΗΤΑΣ

```
int linearSearch(int[] arr, int n, int x) {
  for (int i = 0; i < n; i++) {
    if (arr[i] == x)
      return i;
  }
  return -1;
}
```

**Χειρότερη περίπτωση:** Το στοιχείο να βρίσκεται στη τελευταία θέση του πίνακα, ή να μην υπάρχει καθόλου.
n συγκρίσεις $\Rightarrow \mathcal{O}(n)$

**Καλύτερη περίπτωση:** Το στοιχείο να βρίσκεται στην πρώτη θέση του πίνακα.
1 σύγκριση $\Rightarrow \mathcal{O}(1)$

# ΑΛΓΟΡΙΘΜΟΣ ΦΥΣΑΛΙΔΑΣ (BUBBLE SORT)

**Ταξινόμηση**

Αναδιάταξη στοιχείων μιας λίστας έτσι ώστε να είναι σε αύξουσα ή φθίνουσα διάταξη

| 5 | -3 | 6 | 22 | 0 | -5 | 2 | 45 | -9 | 1 | -2 | 3 | -4 | 66 | 10 | 12 | -8 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| -9 | -8 | -5 | -4 | -3 | -2 | 0 | 1 | 2 | 3 | 5 | 6 | 10 | 12 | 22 | 45 | 66 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

# ΑΛΓΟΡΙΘΜΟΣ ΦΥΣΑΛΙΔΑΣ

- Ο αλγόριθμος φυσαλίδας είναι ο απλούστερος αλγόριθμος ταξινόμησης που λειτουργεί με τη **συνεχή εναλλαγή των γειτονικών στοιχείων εάν βρίσκονται σε λάθος σειρά**

| Πέρασμα 1 | | Πέρασμα 2 | |
|---|---|---|---|
| **5 1** 4 2 8 | → **1 5** 4 2 8 | **1 4** 2 5 8 | → **1 4** 2 5 8 |
| 1 **5 4** 2 8 | → 1 **4 5** 2 8 | 1 **4 2** 5 8 | → 1 **2 4** 5 8 |
| 1 4 **5 2** 8 | → 1 4 **2 5** 8 | 1 2 **4 5** 8 | → 1 2 **4 5** 8 |
| 1 4 2 **5 8** | → 1 4 2 **5 8** | | |

- Ο πίνακας παραμένει ταξινομημένος, αλλά ο αλγόριθμος δεν το γνωρίζει ότι έχει ολοκληρωθεί.
- Ο αλγόριθμος χρειάζεται κάποια περάσματα χωρίς καμία εναλλαγή προκειμένου να ολοκληρωθεί.

# ΚΩΔΙΚΑΣ C

```
void bubbleSort(int arr[], int n)  {
    for (int i = 0; i < n-1; i++)
        for (int j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1]) {
                // swap arr[j+1] and arr[j]
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
}
```

**Πολυπλοκότητα**

# ΑΝΑΛΥΣΗ ΠΟΛΥΠΛΟΚΟΤΗΤΑΣ

```
void bubbleSort(int arr[], int n)  {
    for (int i = 0; i < n-1; i++)
        for (int j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1]) {
                // swap arr[j+1] and arr[j]
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
}
```

n-1 επαναλήψεις

n-i-1 επαναλήψεις

1 σύγκριση

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

$$= (n-1)+(n-2)+(n-3)+\cdots+2+1$$

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

# ΑΝΑΛΥΣΗ ΠΟΛΥΠΛΟΚΟΤΗΤΑΣ

**Χειρότερη Περίπτωση: $\mathcal{O}(n^2)$**
**Καλύτερη Περίπτωση**

```
void bubbleSort(int arr[], int n)   {
    for (int i = 0; i < n-1; i++)
        for (int j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1]) {
                // swap arr[j+1] and arr[j]
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
}
```

n-1 επαναλήψεις

n-i-1 επαναλήψεις

1 σύγκριση

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

$$= (n-1)+(n-2)+(n-3)+\cdots+2+1$$

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

# ΠΙΟ ΑΠΟΔΟΤΙΚΗ ΥΛΟΠΟΙΗΣΗ

```
void bubbleSort(int arr[], int n)  {
    boolean swapped;
    for (int i = 0; i < n-1; i++)  {
        swapped = false;
        for (int j = 0; j < n-i-1; j++)  {
            if (arr[j] > arr[j+1])  {  // swap arr[j] and arr[j+1]
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j+1] = temp;
                swapped = true;
            }
        }
        if (swapped == false)
            break;
    }
}
```

1 σύγκριση

**Χειρότερη και καλύτερη περίπτωση;**

# ΠΙΟ ΑΠΟΔΟΤΙΚΗ ΥΛΟΠΟΙΗΣΗ

```
void bubbleSort(int arr[], int n)  {
    boolean swapped;
    for (int i = 0; i < n-1; i++)  {
        swapped = false;
        for (int j = 0; j < n-i-1; j++)  {
            if (arr[j] > arr[j+1])  {  // swap arr[j] and arr[j+1]
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j+1] = temp;
                swapped = true;
            }
        }
        if (swapped == false)
            break;
    }
}
```

1 σύγκριση

**Χειρότερη περίπτωση**

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

Άρα $\mathcal{O}(\boldsymbol{n^2})$

**Καλύτερη περίπτωση**

**(πίνακας ταξινομημένος)**

$$n - 1 + 1 = n$$

Άρα $\mathcal{O}(\boldsymbol{n})$

# ΥΠΟΛΟΓΙΣΜΟΣ ΠΟΛΥΩΝΥΜΟΥ

**Υπολογισμός Πολυωνύμου**

Δοσμένων των συντελεστών ενός πολυωνύμου βαθμού *n* να υπολογιστεί η τιμή του πολυωνύμου για κάποια τιμή του *x*

$$f(x) = 4x^3 - x^2 + 2x + 1$$

| 1 | 2 | -1 | 4 |
|---|---|---|---|

$$f(x) = 5x^5 - 3x^3 + x + 4$$

| 4 | 1 | 0 | -3 | 0 | 5 |
|---|---|---|---|---|---|

# ΚΩΔΙΚΑΣ C

```
int computePolynomialFunction(double[] arr, int n, double x) {
    double res = 0.0; double pow;
    for (i=n-1; i>=0; i--) { // όπου n είναι ο βαθμός του πολυωνύμου
        pow = 1.0;
        for (j=1; j<=i; j++) {
            pow = pow * x;
        }
        res = res + arr[i] * pow;
    }
    return res;
}
```

| $x^2 + 3x + 2$ με x=5, arr = [2, 3, 1] και n=3 | | | |
|---|---|---|---|
| **i** | **pow** | **res** | **Υπολογισμός τιμής** |
| 2 | 25.0 | 25.0 | $x^2$ |
| 1 | 5.0 | 40.0 | $x^2 + 3x$ |
| 0 | 1.0 | 42.0 | $x^2 + 3x + 2$ |

# ΑΝΑΛΥΣΗ ΠΟΛΥΠΛΟΚΟΤΗΤΑΣ

```
int computePolynomialFunction(double[] arr, int n, double x) {
    double res = 0.0; double pow;
    for (i=n-1; i>=0; i--) { // όπου n είναι ο βαθμός του πολυωνύμου
        pow = 1.0;
        for (j=1; j<=i; j++) {
            pow = pow * x;
        }
        res = res + arr[i] * pow;
    }
    return res;
}
```

n επαναλήψεις

i επαναλήψεις

1 πολλαπλασιασμός

1 πολλαπλασιασμός & 1 πρόσθεση

# ΑΝΑΛΥΣΗ ΠΟΛΥΠΛΟΚΟΤΗΤΑΣ

**Πιο αποδοτική υλοποίηση**

```
int computePolynomialFunction(double[] arr, int n, double x) {
    double res = 0.0; double pow;
    for (i=n-1; i>=0; i--) { // όπου n είναι ο βαθμός του πολυωνύμου
        pow = 1.0;
        for (j=1; j<=i; j++) {
            pow = pow * x;
        }
        res = res + arr[i] * pow;
    }
    return res;
}
```

n επαναλήψεις

i επαναλήψεις

1 πολλ

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

$$=1+2+3+\cdots+(n-1)+n=\frac{n(n+1)}{2}$$

$$=\frac{1}{2}n^2+\frac{1}{2}n$$

# ΠΙΟ ΑΠΟΔΟΤΙΚΗ ΥΛΟΠΟΙΗΣΗ

```
int computePolynomialFunction(double[] arr, int n, double x) {
    double res = 0.0; double pow = 1.0;
    for (i=0; i<n; i++) { // όπου n είναι ο βαθμός του πολυωνύμου
        if ( i > 0)
            pow = pow * x;
        res = res + arr[i] * pow;
    }
    return res;
}
```

n επαναλήψεις

1 σύγκριση

1 πολλαπλασιασμός

1 πολλαπλασιασμός & μια πρόσθεση

| $x^2 + 3x + 2$ με x=5, arr = [2, 3, 1] και n=3 | | | |
|---|---|---|---|
| i | pow | res | Υπολογισμός τιμής |
| 0 | 1.0 | 2.0 | 2 |
| 1 | 5.0 | 17.0 | 3x +2 |
| 2 | 25.0 | 42.0 | $x^2 + 3x + 2$ |

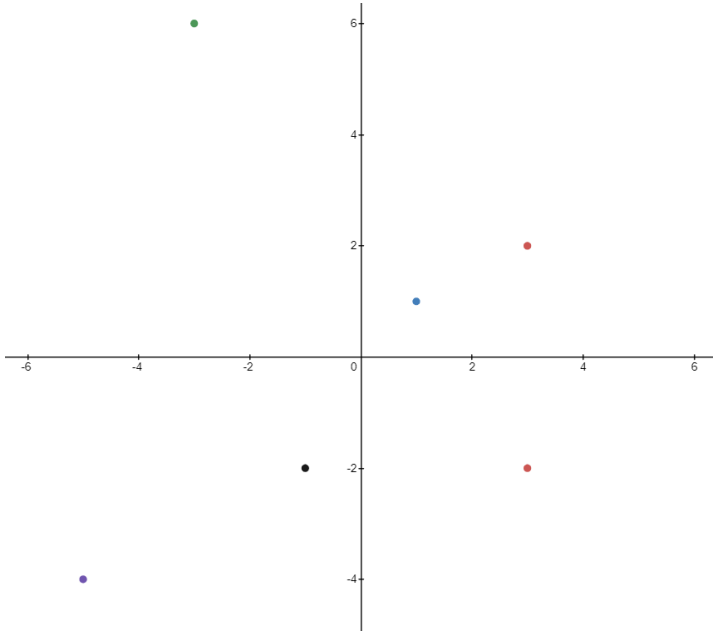
$$1 + \sum_{i=1}^{n-1} 2 = 1 + 2(n-1)$$
$$= 2n - 1$$

Άρα $\boldsymbol{O(n)}$

# ΚΟΝΤΙΝΟΤΕΡΟ ΖΕΥΓΟΣ ΣΗΜΕΙΩΝ

**Κοντινότερο Ζεύγος Σημείων**

Να βρεθούν τα δυο πλησιέστερα σημεία μεταξύ n σημείων στον k-διάστατο χώρο




| |
|---|
| (3,2) |
| (1,1) |
| (-3,6) |
| (-5,-4) |
| (-1,-2) |
| (3,-2) |

# ΚΩΔΙΚΑΣ C

```
minDistance = FLT_MAX;
for (int i = 0; i < n-1; i++) {
      for (int j = i + 1; j < n; j++) {
            float x = arr[i], y = arr[j];
            if (dist(x, y) < minDistance) {
                  minDistance = dist(x, y);
                  nearestPair.p1 = x;
                   nearestPair.p2 = y;
             }
      }
}
```

**Πολυπλοκότητα**

# ΑΝΑΛΥΣΗ ΠΟΛΥΠΛΟΚΟΤΗΤΑΣ

```
minDistance = FLT_MAX;
for (int i = 0; i < n-1; i++) {
     for (int j = i + 1; j < n; j++) {
          float x = arr[i], y = arr[j];
          if (dist(x, y) < minDistance) {
               minDistance = dist(x, y);
               nearestPair.p1 = x;
               nearestPair.p2 = y;
          }
     }
}
```

n-1 επαναλήψεις

n-i-1 επαναλήψεις

1 σύγκριση & 1 υπολογισμός απόστασης

1 υπολογισμός απόστασης

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

$$= (n-1) + (n-2) + \cdots + 2 + 1$$

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

Άρα $\boldsymbol{\mathcal{O}(n^2)}$ συγκρίσεις

# ΤΑΙΡΙΑΣΜΑ ΣΥΜΒΟΛΟΣΕΙΡΑΣ (STRING MATCHING)

**Ταίριασμα Συμβολοσειράς**
Εύρεση μιας συμβολοσειράς σε ένα κείμενο

- Συμβολοσειρά (pattern): σε
  Κείμενο (text): Εύρεση μιας συμβολο**σε**ιράς **σε** ένα κείμενο

- Συμβολοσειρά (pattern): 01110
  Κείμενο (text): 00101010000101**0111**00001

# ΚΩΔΙΚΑΣ C

```c
void patternSearch(char* pat, char* txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    // Βρόχος για ολίσθηση στον πίνακα κατά μια θέση
    for (int i = 0; i <= N - M; i++) {
        int j;
        // Για το τρέχον i, έλεγξε για ταίριασμα
        for (j = 0; j < M; j++)
            if (txt[i + j] != pat[j])
                break;

        if (j == M) // Εάν pat[0...M-1] = txt[i, i+1, ...i+M-1]
            printf("Pattern found at index %d \n", i);
    }
}
```

**Πολυπλοκότητα**

# ΑΝΑΛΥΣΗ ΠΟΛΥΠΛΟΚΟΤΗΤΑΣ

**Χειρότερη Περίπτωση:** $\mathcal{O}(nm - m^2)$
**Καλύτερη Περίπτωση**

```
void patternSearch(char* pat, char* txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    // Βρόχος για ολίσθηση στον πίνακα κατά μια θέση
    for (int i = 0; i <= N - M; i++) {
        int j;
        // Για το τρέχον i, έλεγξε για ταίριασμα
        for (j = 0; j < M; j++)
            if (txt[i + j] != pat[j])
                break;

        if (j == M) // Εάν pat[0...M-1] = txt[i, i+1,
            printf("Pattern found at index %d \n", i);
    }
}
```

n-m+1 επαναλήψεις

m επαναλήψεις

1 σύγκριση

1 σύγκριση

$$\sum_{i=0}^{n-m}\left(\left(\sum_{j=0}^{m-1}1\right)+1\right)=\sum_{i=0}^{n-m}(m+1)$$

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

$$=nm+n-m^2-m+m+1$$

$$=nm+n-m^2+1$$

Άρα $\mathcal{O}(nm - m^2)$ συγκρίσεις

# ΣΥΜΠΕΡΑΣΜΑΤΑ

**Πλεονεκτήματα**

- Μεγάλη εφαρμοσιμότητα και απλότητα
- Δίνει λογικούς αλγορίθμους για σημαντικά προβλήματα
  - αναζήτηση, ταξινόμηση, πολλαπλασιασμός πινάκων
- Δίνει πρότυπους αλγορίθμους για απλά προβλήματα
  - Άθροισμα/γινόμενο αριθμών, εύρεση μέγιστου/ελάχιστου σε λίστα

**Μειονεκτήματα**

- Δίνει σπάνια αποτελεσματικούς αλγορίθμους
- Μερικοί αλγόριθμοι ωμής βίας είναι απαράδεκτα αργοί
- Δεν είναι τόσο δημιουργική τεχνική όσο άλλες σχεδιαστικές τεχνικές

**Κωνσταντίνος Γιαννουτάκης**

Επίκ. Καθηγητής

kgiannou@uom.edu.gr