# ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

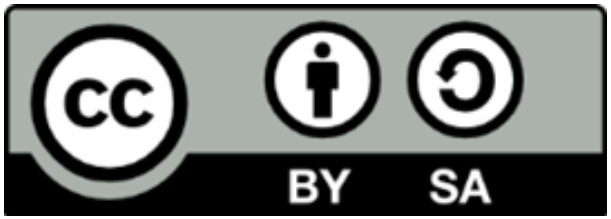
## Ενότητα 9: Στατιστικά Διάταξης-Στατιστικά σε Μέσο Γραμμικό Χρόνο

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

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.

# Στατιστικά Διάταξης

- Με τον όρο **στατιστικά διάταξης** (*order statistics*) εννοούμε την περίπτωση όπου δεδομένου ενός πίνακα *A* με *n* αταξινόμητα στοιχεία, πρέπει να αναζητήσουμε το μικρότερο, το μεγαλύτερο, το μεσαίο ή γενικά το *k*-οστό στοιχείο του πίνακα με βάση την τιμή του κλειδιού του.
- Εστω ότι θέλουμε να αναζητήσουμε ταυτόχρονα το μικρότερο και το μεγαλύτερο στοιχείο ενός πίνακα. Η προφανής λύση είναι να εκτελέσουμε 2 σαρώσεις, μια σάρωση για την εύρεση του μικρότερου και στη συνέχεια μια σάρωση για την εύρεση του μεγαλύτερου στοιχείου.
- Είναι ευνόητο ότι η πολυπλοκότητα αυτής της προσέγγισης είναι Θ(*n*).
- Στη συνέχεια παρουσιάζουμε έναν αλγόριθμο που με μια σάρωση επιτυγχάνουμε ταυτόχρονη εύρεση των δυο στοιχείων.

**Αλγόριθμος maxmin1**

1. $max \leftarrow A[1]$; $min \leftarrow A[1]$;
2. **for** $i \leftarrow 2$ **to** $n$ **do**
3. **if** $A[i] > max$ $max \leftarrow A[i]$;
4. **if** $A[i] < min$ $min \leftarrow A[i]$;
5. **return** *max*, *min*

Η παραπάνω προσέγγιση είναι απλοϊκή αν και γίνεται μια σάρωση του πίνακα, κάθε στοιχείο υποβάλλεται σε 2 συγκρίσεις.

Θα βοηθούσε αν αντικαταστούμε τις εντολές 3-4 με τις εντολές:

3. **if** $A[i] > max$ $max \leftarrow A[i]$;
4. else **if** $A[i] < min$ $min \leftarrow A[i]$;

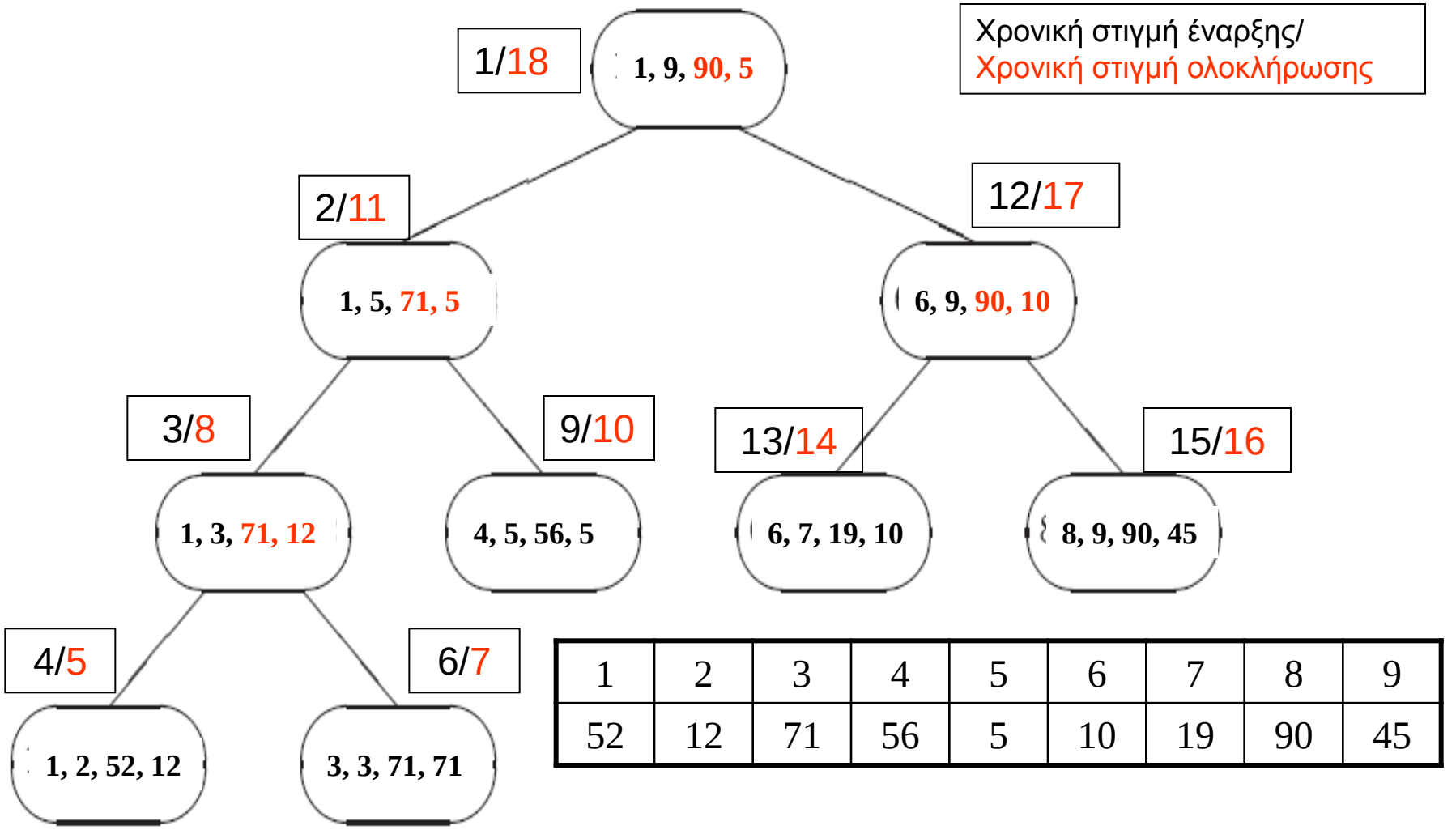
Και αυτή η εκδοχή μπορεί αν αποδειχθεί ότι δε βοηθάει τη χειρότερη περίπτωση. Ο επόμενος αλγόριθμος προσπαθεί να εκμεταλλευτεί τη τεχνική διαίρει και βασίλευε.

Υποθέτει ότι με τα ορίσματα *i* και *j* δίνουμε τα όρια του πίνακα και με τα ορίσματα *fmax*, *fmin* μας επιστρέφονται οι τιμές που μας ενδιαφέρουν. Επίσης υποθέτουμε ότι η συνάρτηση *max* επιστρέφει το μέγιστο μεταξύ 2 στοιχείων και η *min* επιστρέφει το ελάχιστο μεταξύ 2 στοιχείων.

**Αλγόριθμος maxmin2(*i, j, fmax, fmin*)**

```
1.  if i == j
2.      max ← A[i]; min ← A[i];
3.  else if i == j -1  then
4.              if A[i] < A[j]
5.                  fmax ← A[j]; fmin ← A[i];
6.              else
7.                  fmax ← A[i]; fmin ← A[j];
8.      else
9.          middle ← (i + j)/2;
10.         maxmin2(i, middle, gmax, , gmin);
11.         maxmin2(middle+1, j, hmax, , hmin);
12.         fmax ← max(gmax, hmax);
13.         fmin ← min(gmin, hmin);
```

Εστω ότι το περιεχόμενο του πίνακα Α με τα 9 στοιχεία 52, 12, 71, 56, 5, 10, 19, 90, 45. Στο παρακάτω σχήμα απεικονίζεται ένα δένδρο με κόμβους που περιέχουν τα ορίσματα των κλήσεων. Διασχίζοντας το δένδρο με προτεραιότητα κατά βάθος (dfs) μπορούμε να αποτυπώσουμε την ακριβή σειρά των διαφόρων κλήσεων




| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| 52 | 12 | 71 | 56 | 5 | 10 | 19 | 90 | 45 |

**Πρόταση**

*Η πολυπλοκότητα του maxmin2 είναι Θ(n) στη χειρότερη περίπτωση*

**Απόδειξη**

Η πολυπλοκότητα αυτού του αλγόριθμου μπορεί να βρεθεί επιλύοντας της ακόλουθη αναδρομική σχέση

$$T(n) = \begin{cases} 0 & n = 1 \\ 1 & n = 2 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 2 & n > 2 \end{cases}$$

όπου οι σταθεροί όροι εκφράζουν το κόστος των συγκρίσεων στις εντολές 4 και 12-13.

Θεωρώντας ότι $n = 2^k$ και με διαδοχικές αντικαταστάσεις έχουμε

$$T(n) = 2T\left(\frac{n}{2}\right) + 2 = 2\left(2T\left(\frac{n}{4}\right) + 2\right) + 2 = 4T\left(\frac{n}{4}\right) + 4 + 2$$

$$= 4\left(2T\left(\frac{n}{8}\right) + 2\right) + 4 + 2 = 8T\left(\frac{n}{8}\right) + 8 + 4 + 2$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 2^3 + 2^2 + 2^1$$

$$= 2^k T\left(\frac{n}{2^k}\right) + 2^{k-1} T\left(\frac{n}{2^{k-1}}\right) + \ldots + 2^3 + 2^2 + 2^1$$

$$= 2^k T\left(\frac{n}{2^k}\right) + 2^{k-1} T\left(\frac{n}{2^{k-1}}\right) + \ldots + 2^3 + 2^2 + 2^1$$

$n = 2^k$

$$= 2^k T(1) + 2^{k-1} T\left(\frac{2n}{2^k}\right) + \ldots + 2^3 + 2^2 + 2^1$$

$$= 1 + 2 + 2^2 + \ldots + 2^{k-1} - 1 = \sum_{i=0}^{k-1} 2^i - 1 = \frac{2^{k-1+1} - 1}{2 - 1} - 1 = 2^k - 1 - 1 = 2^k - 2 = n - 2$$

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \ldots + x^n = \frac{x^{n+1} - 1}{x - 1}$$ γεωμετρική σειρά

Ο παραπάνω αλγόριθμος είναι βελτιωμένος και δεν υπερβαίνει το όριο Θ(*n*)

# Στατιστικά με Μέσο Γραμμικό Χρόνο

- Εστω ότι επιθυμούμε να βρούμε το *k*-οστό στοιχείο σε αταξινόμητο πίνακα. Αυτό μπορεί να γίνει ταξινομώντας τον πίνακα και λαμβάνοντας το περιεχόμενο της αντίστοιχης θέσης του πίνακα.
- Ηδη έχουμε αποδείξει ότι το κάτω όριο των αλγόριθμων ταξινόμησης που στηρίζονται σε συγκρίσεις είναι $\Omega(n\log_2 n)$. Επομένως το όριο αυτό προσδιορίζει και τη πολυπλοκότητα της μεθόδου που ανάγει το πρόβλημα της αναζήτησης στοιχείου σε πρόβλημα ταξινόμησης.
- Στη συνέχεια θα επιλύσουμε το πρόβλημα της αναζήτησης του *k*-οστού στοιχείου με μια μέθοδο που στηρίζεται στη γρήγορη ταξινόμηση.

**Αλγόριθμος find(A, *left*, *right*, *k*);**

```
1.    if left == right then return A[left]
2.    else
3.        lo ← left; hi ← right + 1; pivot ← A[left];
4.        do
5.            do lo ← lo+ 1 while A[lo] <= pivot;
6.            do hi ← hi -1 while A[hi] >= pivot;
7.            if lo < hi then swap(A[lo], A[hi]);
8.        while hi >= lo;
9.        swap(A[left], A[hi]);
10.       if (k == hi) return A[hi];
11.       else if (k < hi) find(A, left, hi-1, k);
12.       else find(A, hi+1, right, k-hi)
```

*Οι εντολές 3-9 αντιστοιχούν με τις αντίστοιχες εντολές της γρήγορης αναζήτησης*

- Ο αλγόριθμος find είναι σχεδόν ταυτόσημος με τη γρήγορη αναζήτηση. Συγκεκριμένα οι εντολές 3-9 αντιστοιχούν με τις αντίστοιχες εντολές της γρήγορης αναζήτησης, καθώς αφορούν στις διαδικασίες επιλογής του pivot και του διαμερισμού. Στη συνέχεια διαφοροποιούνται οι εντολές 10-12.
- Όταν το pivot οριστικοποιείται στη θέση *hi* του πίνακα αποφασίζουμε αν θα τερματίσουμε ή θα συνεχίσουμε την αναζήτηση του *k*-οστού στοιχείου στον αριστερό ή στο δεξιό υποπίνακα με αναδρομικό τρόπο.

**Πρόταση**

Η πολυπλοκότητα του find είναι Θ($n^2$), Θ($n$) και *Θ(n)* στη χειρότερη στη μέση και στη καλύτερη περίπτωση.

**Απόδειξη**

Κατ' αρχήν η χειρότερη περίπτωση είναι *Θ($n^2$),* και αυτό συμβαίνει όταν το επιλεγόμενο pivot είναι το μικρότερο ή το μεγαλύτερο στοιχείο.

Θα μελετήσουμε τη μέση περίπτωση για την οποία ισχύει η (1)

$$T(n) = n + \frac{1}{n}\sum_{k=0}^{n-1} T(k) \quad (1)$$

$$T(n) = n + \frac{1}{n}\sum_{k=1}^{n-1} T(k) \quad (2)$$

Ο όρος *n* του δεξιού σκέλους αντιστοιχεί στις συγκρίσεις που θα εκτελεσθούν κατά τη σάρωση του πίνακα (εντολές 5-6) πριν επιτευχθεί η διαμέριση. Με αρχικές συνθήκες T(0) = 0, T(1) = 1, έχουμε τη σχέση:

πολλαπλασιάζουμε επί *n* και προκύπτει η (3)

$$nT(n) = n^2 + \sum_{k=1}^{n-1} T(k) \quad (3)$$

στην ίδια σχέση αντικαθιστούμε το *n* με *n*-1 και πολλαπλασιάζουμε με *n*-1, οπότε προκύπτει η (4):

$$(n-1)T(n-1) = (n-1)^2 + \sum_{k=1}^{n-2} T(k) \quad (4)$$

Αφαιρώντας τα αντίστοιχα σκέλη των σχέσεων (3) & (4)

$$nT(n)-(n-1)T(n-1)=2n-1+T(n-1)\Rightarrow$$

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

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

$$T(2)-T(1)=2-\frac{1}{2}\Rightarrow$$

} $n$ -1 όροι

Αθροίζοντας αντίστοιχα τα αριστερά και δεξιά σκέλη και απλοποιώντας προκύπτει :

$$T(n)=1+2(n-1)-\left(\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}\right)=2n-\left(1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}\right)$$

$$=2n-H_n$$

Έπεται ότι η πολυπλοκότητα της μέσης περίπτωσης της find είναι $\Theta(n)$

---

$$H_n\in\Theta(\ln n)=\Theta(\log_2 n)$$

*See Epp, Example 11.4.6 & 11.4.7*

# Τέλος Ενότητας