ARISTOTLE UNIVERSITY OF THESSALONIKI

SCHOOL OF INFORMATICS

DATA&WEB SCIENCE LABORATORY

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Ταξινόμηση

Απόστολος Ν. Παπαδόπουλος
Αναπληρωτής Καθηγητής
Τμήμα Πληροφορικής Α.Π.Θ.

Βασικές Έννοιες

Δίνεται ένα πίνακας (array) με *n* στοιχεία, και θέλουμε να διατάξουμε τον πίνακα σε αύξουσα ή φθίνουσα διάταξη (**χβτγ** υποθέτουμε ότι πάντα ταξινομούμε σε αύξουσα διάταξη).

(*χβτγ: **χ**ωρίς **β**λάβη **τ**ης **γ**ενικότητας*)

Βασικές Έννοιες

Γιατί χρειαζόμαστε την ταξινόμηση;

- Σε έναν ταξινομημένο πίνακα μπορούμε να εφαρμόσουμε δυαδική αναζήτηση.
- Πολλές φορές πρέπει να πάρουμε τα αποτελέσματα σε αύξουσα/φθίνουσα διάταξη ως προς το επίθετο ή τον ΑΜΚΑ.
- Μπορούμε να λύσουμε πιο πολύπλοκα προβλήματα χρησιμοποιώντας την ταξινόμηση ως **υπορουτίνα** (συνάρτηση).
- Και για πολλούς άλλους λόγους ...

Βασικές Έννοιες

Παράδειγμα: δίνεται ένα σύνολο από ακέραιους αριθμούς και θέλουμε να διαγράψουμε τις πολλαπλές εμφανίσεις στοιχείων.

Παράδειγμα2: δίνεται ένα σύνολο από ακέραιους αριθμούς και θέλουμε να υπολογίσουμε πόσες φορές εμφανίζεται το καθένα.

Είσοδος:

1, 2, 1, 3, 6, 5, 1, 2, 6, 4, 7, 7, 8, 7, 9, 6

Έξοδος (διαγραφή πολλαπλών εμφανίσεων):

1, 2, 3, 4, 5, 6, 7, 8, 9

Έξοδος (πόσες φορές εμφανίζεται το καθένα):

(1,3), (2,2), (3,1), (4,1), (5,1), (6,3), (7,3), (8,1), (9,1)

Βασικές Έννοιες

Για την επίλυση των προβλημάτων ταξινομούμε τα στοιχεία και μετά τα πράγματα είναι εύκολα …

1, 1, 1,	2, 2,	3,	4,	5,	6, 6, 6,	7, 7, 7,	8,	9

Βασικές Έννοιες

Γνωρίζουμε ήδη κάποιους βασικούς αλγορίθμους ταξινόμησης:

BubbleSort (ταξινόμηση με ανταλλαγή)

InsertionSort (ταξινόμηση με εισαγωγή)

SelectionSort (ταξινόμηση με επιλογή)

Οι αλγόριθμοι αυτοί έχουν πολυπλοκότητα χειρότερης περίπτωσης $O(n^2)$

Βασικές Έννοιες

Θα απαντήσουμε σε δύο βασικά ερωτήματα:

- Μπορούμε να ταξινομήσουμε σε χρόνο καλύτερο από τετραγωνικό;
- Ποιός είναι ο πιο γρήγορος αλγόριθμος ταξινόμησης;

Ταξινόμηση με Συγχώνευση

Ας δούμε έναν αλγόριθμο ταξινόμησης που είναι στη χειρότερη περίπτωση πιο γρήγορος από BubbleSort/InsertionSort/SelectionSort

Ο αλγόριθμος καλείται **MergeSort** (ταξινόμηση με συγχώνευση).

Ταξινόμηση με Συγχώνευση

Ο αλγόριθμος βασίζεται στη **συγχώνευση**.

Δίδονται δύο πίνακες A και B που είναι ταξινομημένοι. Να κατασκευαστεί πίνακας C που να περιέχει τα στοιχεία του A και του B και να είναι ταξινομημένος.

Ταξινόμηση με Συγχώνευση

A: 1 5 7 10 15

B: 2 6 8 9 20

C: 1 2 5 6 7 8 9 10 15 20

Συγχώνευση ταξινομημένων πινάκων

Ταξινόμηση με Συγχώνευση

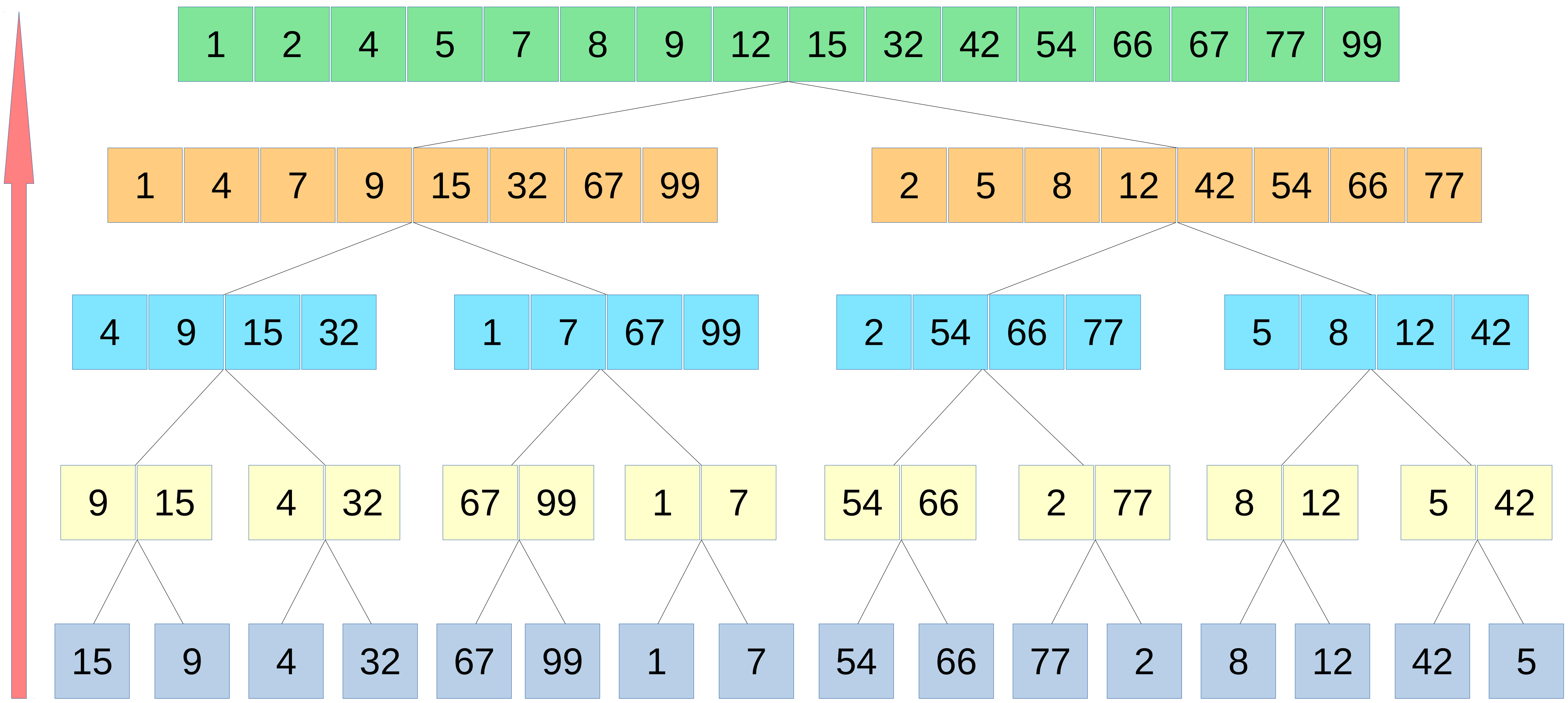
Έχουμε δείξει ότι το κόστος χειρότερης περίπτωσης για τη συγχώνευση δύο πινάκων με n στοιχεία ο καθένας είναι $2n$-1 = Ο(n).

Ταξινόμηση με Συγχώνευση

Ψευδοκώδικας με αναδρομή

```
MergeSort(arr, left, right):
    if left > right
        return
    mid = (left+right)/2
    MergeSort(arr, left, mid)
    MergeSort(arr, mid+1, right)
    merge(arr, left, mid, right)
end
```

Ταξινόμηση με Συγχώνευση



Ανάλυση MergeSort

Πρέπει να απαντήσουμε στα ακόλουθα:

- Πόσες **συγκρίσεις** γίνονται σε κάθε επίπεδο;
- Πόσα **επίπεδα** υπάρχουν;

Με αυτές τις πληροφορίες μπορούμε να προσδιορίσουμε το κόστος της MergeSort στη χειρότερη περίπτωση.

Ανάλυση MergeSort

Απλοϊκή μέθοδος

Το ύψος του δένδρου συγχωνεύσεων είναι O(log$n$). Σε κάθε επίπεδο απαιτούνται το πολύ n συγκρίσεις για τις συγχωνεύσεις.

Άρα συνολικά έχουμε O(n log$n$).

Ανάλυση MergeSort

Με βάση την **αναδρομική συνάρτηση** της MergeSort θα έχουμε ότι αν το κόστος σε πλήθος συγκρίσεων είναι $T(n)$, το κόστος προσδιορίζεται ως εξής:

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

Ανάλυση MergeSort

Με όποιον τρόπο και αν δουλέψουμε φτάνουμε στο συμπέρασμα ότι η πολυπλοκότητα χειρότερης περίπτωσης για την MergeSort είναι $O(n \log n)$.

Για την ακρίβεια είναι $\Theta(n \log n)$.

Συμπέρασμα: υπάρχει αλγόριθμος ταξινόμησης που έχει πολυπλοκότητα καλύτερη από την τετραγωνική.

QuickSort (Γρήγορη Ταξινόμηση)

Χαρακτηριστικό παράδειγμα αλγορίθμου που ανήκει στην κατηγορία **“Διαίρει και Βασίλευε”**.

Και η MergeSort που είδαμε, επίσης μπορεί να χαρακτηριστεί αλγόριθμος της ίδιας κατηγορίας.

QuickSort

Λογική του αλγορίθμου

- Διάλεξε ένα στοιχείο (ονομάζεται pivot).
- Τοποθέτησε το pivot στη σωστή του θέση.
- Χώρισε τον πίνακα σε δύο μέρη A και B (αριστερά του pivot και δεξιά του pivot).
- Επανέλαβε την ίδια διαδικασία για τα A και B.

QuickSort

15	9	4	32	67	99	1	7	54	66	77	2	8	12	42	5

15	9	4	32	67	99	1	7	54	66	77	2	8	12	42	5

2	9	4	5	12	8	1	7	15	66	77	54	99	67	42	32

2	9	4	5	12	8	1	7	15	66	77	54	99	67	42	32

A **B**

< 15 > 15

QuickSort

Ερώτημα: πως θα μπει το pivot στη σωστή του θέση ώστε αριστερά όλα τα στοιχεία να είναι μικρότερα και δεξιά όλα τα στοιχεία μεγαλύτερα;

QuickSort

15	9	4	32	67	99	1	7	54	66	77	2	8	12	42	5
	i														j

Μετακίνησε το i προς τα δεξιά μέχρι να βρεις το πρώτο στοιχείο που είναι **μεγαλύτερο** του pivot.
Μετακίνησε το j προς τα αριστερά μέχρι να βρεις το πρώτο στοιχείο που είναι **μικρότερο** του pivot.

15	9	4	32	67	99	1	7	54	66	77	2	8	12	42	5
			i												j

Αντιμετάθεση των στοιχείων στις θέσεις i και j.

15	9	4	5	67	99	1	7	54	66	77	2	8	12	42	32
			i												j

Μετακίνησε το i προς τα δεξια μέχρι να βρεις το πρώτο στοιχείο που είναι **μεγαλύτερο** του pivot.
Μετακίνησε το j προς τα αριστερά μέχρι να βρεις το πρώτο στοιχείο που είναι **μικρότερο** του pivot.

15	9	4	5	67	99	1	7	54	66	77	2	8	12	42	32
				i									j		

QuickSort

15	9	4	5	12	99	1	7	54	66	77	2	8	67	42	32
				i									j		

15	9	4	5	12	99	1	7	54	66	77	2	8	67	42	32
					i							j			

15	9	4	5	12	8	1	7	54	66	77	2	99	67	42	32
								i			j				

15	9	4	5	12	8	1	7	2	66	77	54	99	67	42	32
								j	i						

Αντιμετάθεση του pivot=15 με το στοιχείο στη θέση j (2).

2	9	4	5	12	8	1	7	15	66	77	54	99	67	42	32

QuickSort

Πόσες συγκρίσεις στοιχείων θα γίνουν για να μπει το pivot στη σωστή του θέση;

πλήθος συγκρίσεων = n-1 = O(n)

QuickSort

Ψευδοκώδικας QuickSort

```
QuickSort (arr[], low, high) {
    pivot = arr[low]; i = low; j = high;  // το pivot είναι το πρώτο στοιχείο του array
    while (i <= j) {
        while (arr[i] < pivot) i += 1;  // αυξάνουμε το i μέχρι το πρώτο στοιχείο > pivot
        while (arr[j] > pivot) j -= 1;   // μειώνουμε το j μέχρι το πρώτο στοιχείο < pivot
        if (i <= j) { swap(i, j); i += 1; j -= 1; }  // αντιμετάθεση
    }
    swap (low, j); // αντιμετάθεση στοιχείων της θέσης του pivot με τη θέση j
    if (low < j-1) QuickSort (arr, low, j-1);
    if (j+1 < high) QuickSort (arr, j+1, high);
}
```

QuickSort

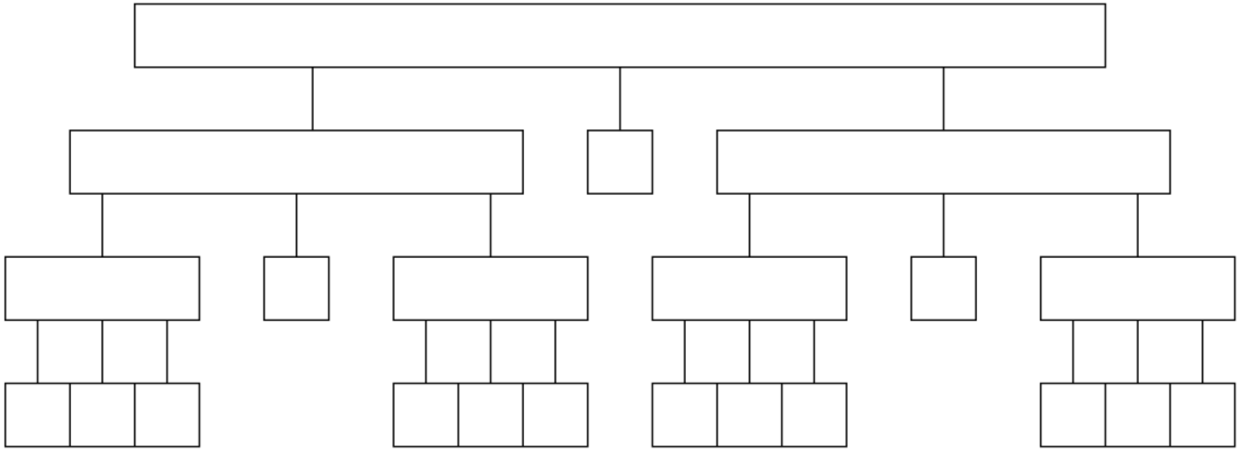
Πόσο καλή είναι η QuickSort;

Αρχικά θα δούμε πως τα πηγαίνει στην καλύτερη περίπτωση και στη χειρότερη περίπτωση.

Μετά θα δούμε ότι με μία μικρή αλλαγή στον αλγόριθμο η QuickSort τα πηγαίνει πολύ καλά και στη **μέση περίπτωση** (δηλαδή σε μία τυπική περίπτωση).

Ανάλυση QuickSort

Αν υποθέσουμε ότι κάθε φορά το pivot χωρίζει το array σε δύο ίσα τμήματα, τότε η δουλειά που θα κάνει η QuickSort θα είναι η λιγότερη δυνατή.



Ανάλυση QuickSort

Η αναδρομική συνάρτηση που προκύπτει είναι ίδια με αυτήν της MergeSort στην περίπτωση αυτή.

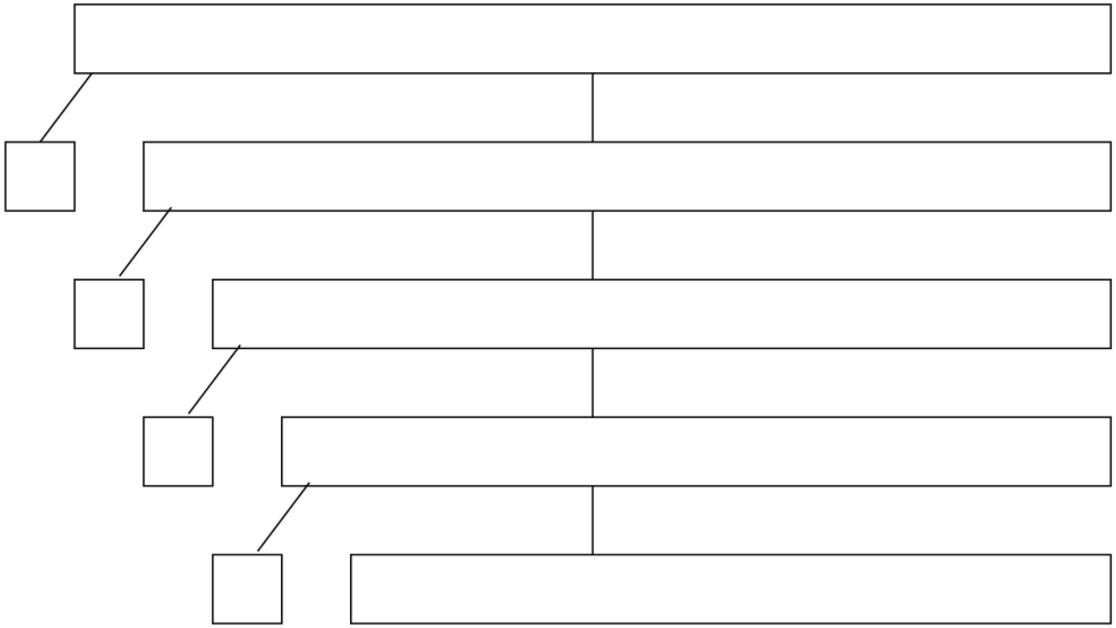
$$T(n) = \begin{cases} O(1) & \text{αν } n = 1 \\ 2T(n/2) + O(n) & \text{αν } n > 1 \end{cases}$$

Άρα το κόστος σε συγκρίσεις είναι: $O(n \log n)$

Αυτό ισχύει όμως στην καλύτερη περίπτωση! Άρα δεν είμαστε ακόμη χαρούμενοι. Ας δούμε τι γίνεται στη χειρότερη περίπτωση.

Ανάλυση QuickSort

Η χειρότερη περίπτωση της QuickSort εμφανίζεται όταν το array είναι π.χ. ταξινομημένο ή περίπου ταξινομημένο.



Ανάλυση QuickSort

Στην περίπτωση αυτή η αναδρομική συνάρτηση θα είναι λίγο διαφορετική:

$$T(n) = \begin{cases} O(1) & \text{αν } n = 1 \\ T(n-1) + O(n) & \text{αν } n > 1 \end{cases}$$

Αν παρατηρήσουμε προσεκτικά, η συνάρτηση μας δίνει ένα κόστος τετραγωνικό ως προς το *n*, άρα έχουμε: $O(n^2)$

Ανάλυση QuickSort

Αν μείνουμε στη χειρότερη περίπτωση, φαίνεται ότι η QuickSort δεν τα πάει και τόσο καλά!

Το πρόβλημα δημιουργείται όταν μας δώσουν να ταξινομήσουμε έναν σχεδόν ταξινομημένο πίνακα ή έναν σχεδόν ανάποδα ταξινομημένο πίνακα.

Μπορούμε να παρακάμψουμε αυτό το πρόβλημα;

Ανάλυση QuickSort

Η βασική αδυναμία του αλγορίθμου είναι ότι το pivot είναι πάντα το πρώτο στοιχείο του πίνακα (και του κάθε υποπίνακα στις αναδρομικές κλήσεις).

Αν κάποιος γνωρίζει ότι επιλέγουμε ως pivot το πρώτο στοιχείο, τότε μπορεί να μας δυσκολέψει!

Ανάλυση QuickSort

Randomized QuickSort

Ιδέα: να διαλέγουμε το pivot με τυχαίο τρόπο!

Σε κάθε αναδρομική κλήση της QuickSort επιλέγουμε το pivot **τυχαία** από όλα τα στοιχεία του πίνακα.

Θεώρημα: αν το pivot επιλέγεται τυχαία με ομοιόμορφη κατανομή, τότε το μέσο πλήθος συγκρίσεων της QuickSort είναι: $O(n \log n)$

Randomized QuickSort

Σε επόμενη διάλεξη θα δείξουμε ότι στη μέση περίπτωση η QuickSort εκτελεί O(n log$n$) συγκρίσεις.

Άρα σε μία τυπική περίπτωση, τα πηγαίνει πολύ καλά και για το λόγο αυτό έχει τη φήμη της **γρήγορης ταξινόμησης**.

Επίσης θα δούμε ότι το n log$n$ αποτελεί και **κάτω φράγμα για το πρόβλημα της ταξινόμησης**: δεν υπάρχει αλγόριθμος ταξινόμησης που βασίζεται σε συγκρίσεις στοιχείων που να μπορεί να ταξινομεί για οποιαδήποτε είσοδο σε χρόνο μικρότερο από n log$n$.