ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΕΣ ΤΕΧΝΙΚΕΣ

http://www.softlab.ntua.gr/~nickie/Courses/progtech/

Διδάσκοντες:	Γιάννης Μαΐστρος	(maistros@cs.ntua.gr)
	Στάθης Ζάχος	(zachos@cs.ntua.gr)
	Νίκος Παπασπύρου	(nickie@softlab.ntua.gr)
Π.Δ. 407/80	Βασίλης Βεσκούκης	(bxb@softlab.ntua.gr)
	Πέτρος Στεφανέας	(petros@mail.ntua.gr)

Διαφάνειες παρουσίασης #10 (β)

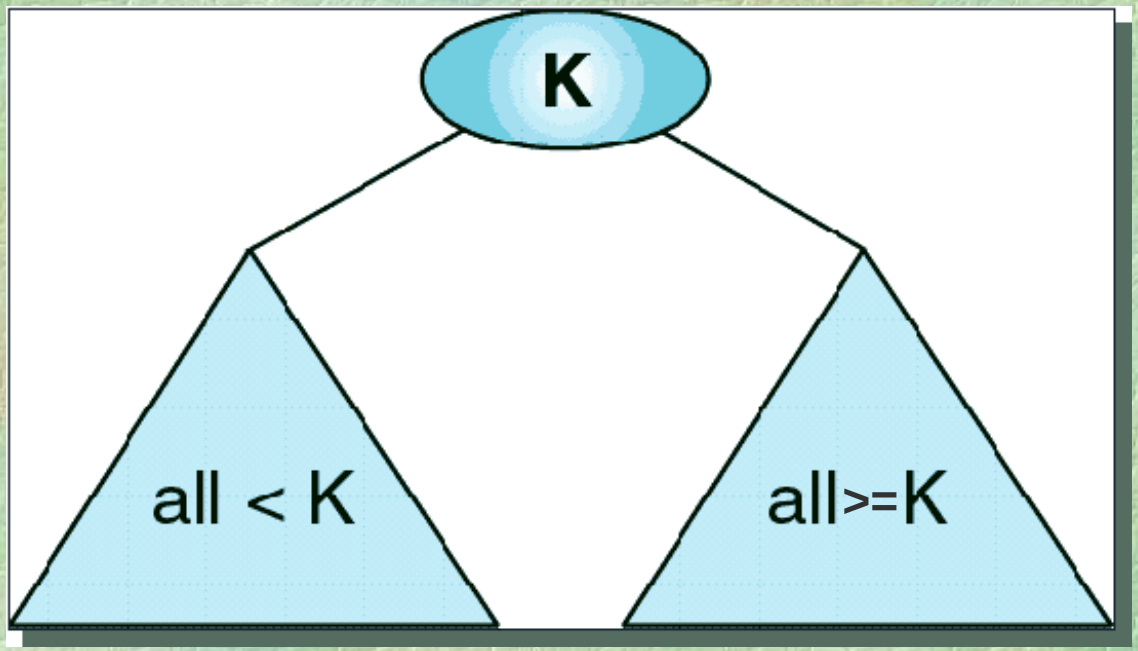
- ✓Δέντρα BST
- ✓Δέντρα AVL
- ✓Περιστροφή - Αναδιάταξη
- ✓Αλγόριθμοι

Δέντρα δυαδικής αναζήτησης

- Δενδρικές δομές δεδομένων στις οποίες
  - Ολα τα στοιχεία στο αριστερό υποδέντρο της ρίζας είναι μικρότερα από την ρίζα
  - Ολα τα στοιχεία στο δεξί υποδέντρο της ρίζας είναι μεγαλύτερα ή ίσα με την ρίζα
  - Το αριστερό και το δεξί υποδέντρο είναι δέντρα δυαδικής αναζήτησης
- Binary Search Trees - BSTs

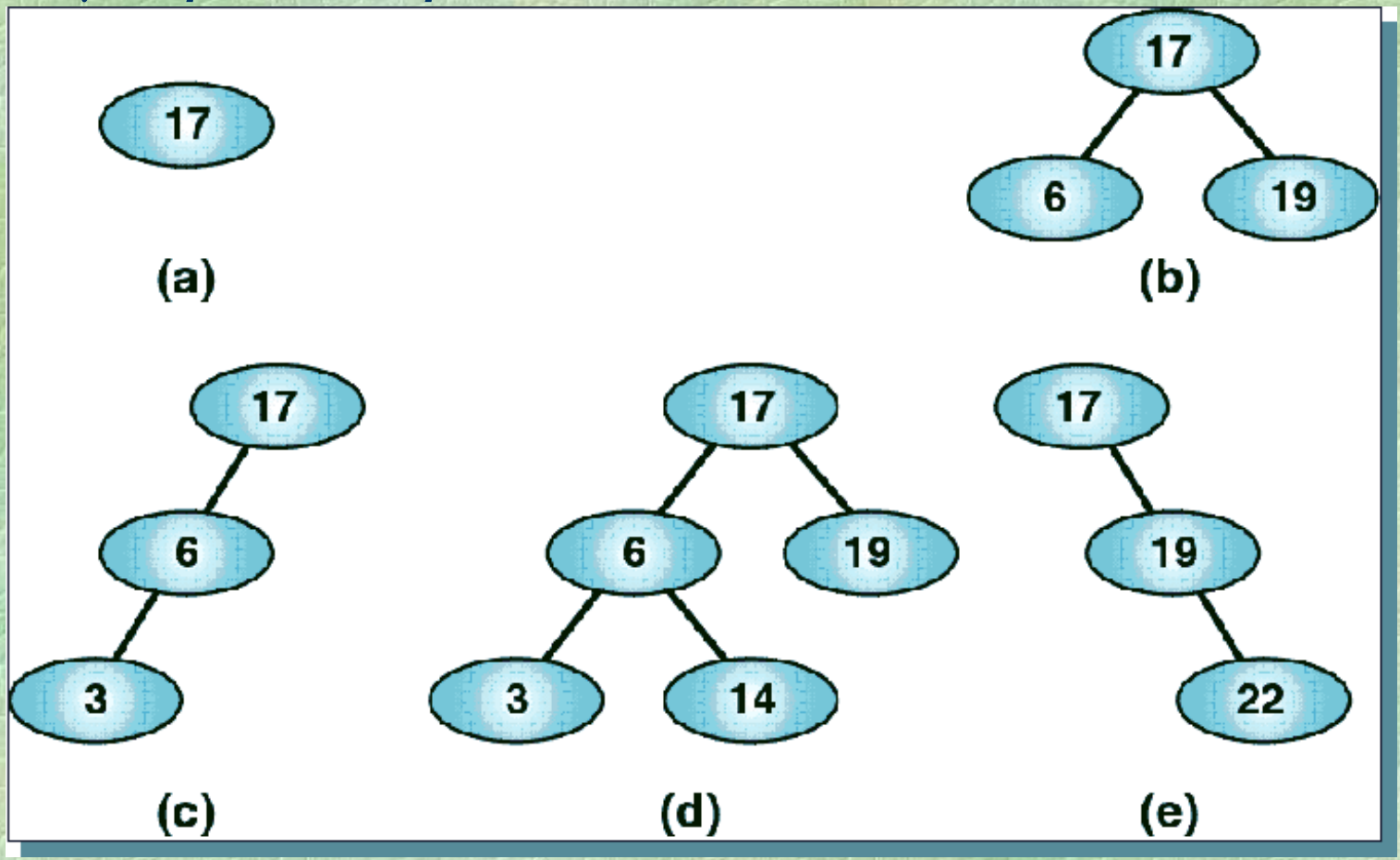
Δέντρα δυαδικής αναζήτησης

- Η γενική εικόνα ενός τέτοιου δέντρου



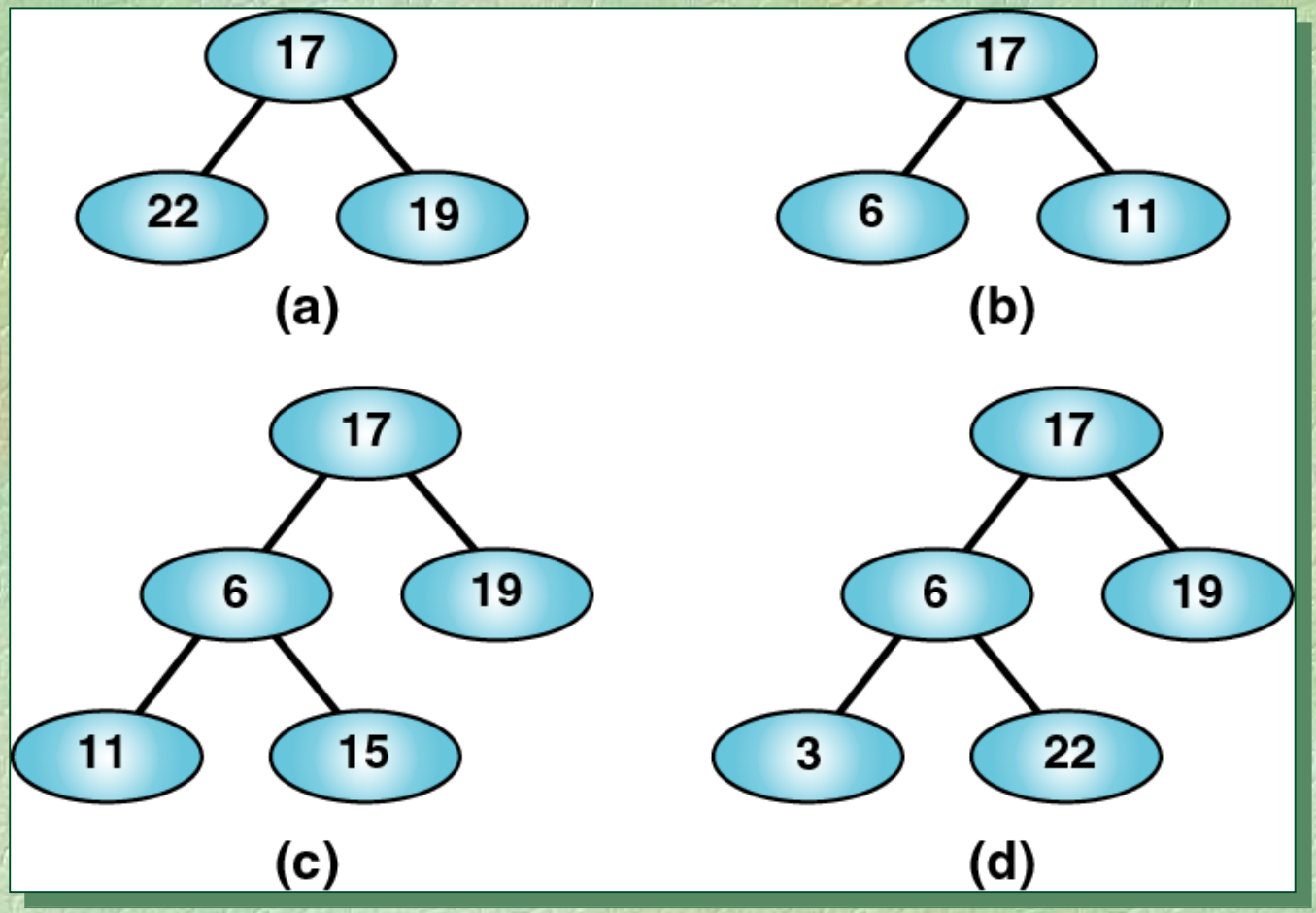
Παραδείγματα δέντρων BST

- Εγκυρα δέντρα

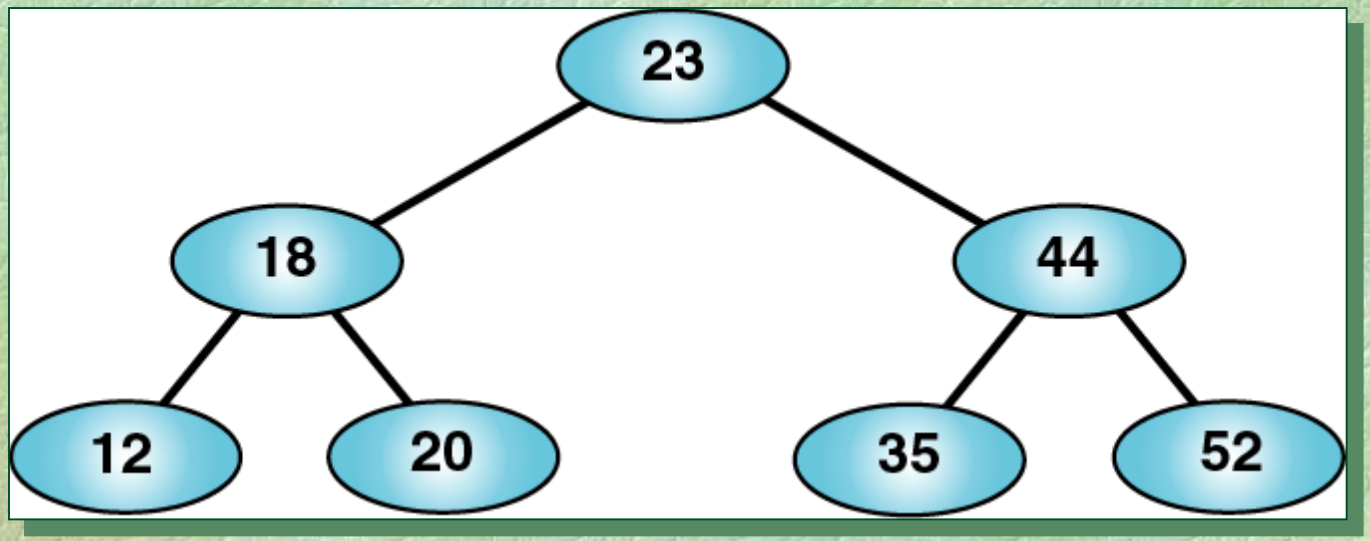


Παραδείγματα δέντρων BST

- Μη-έγκυρα δέντρα



Διάσχιση δυαδικών δέντρων



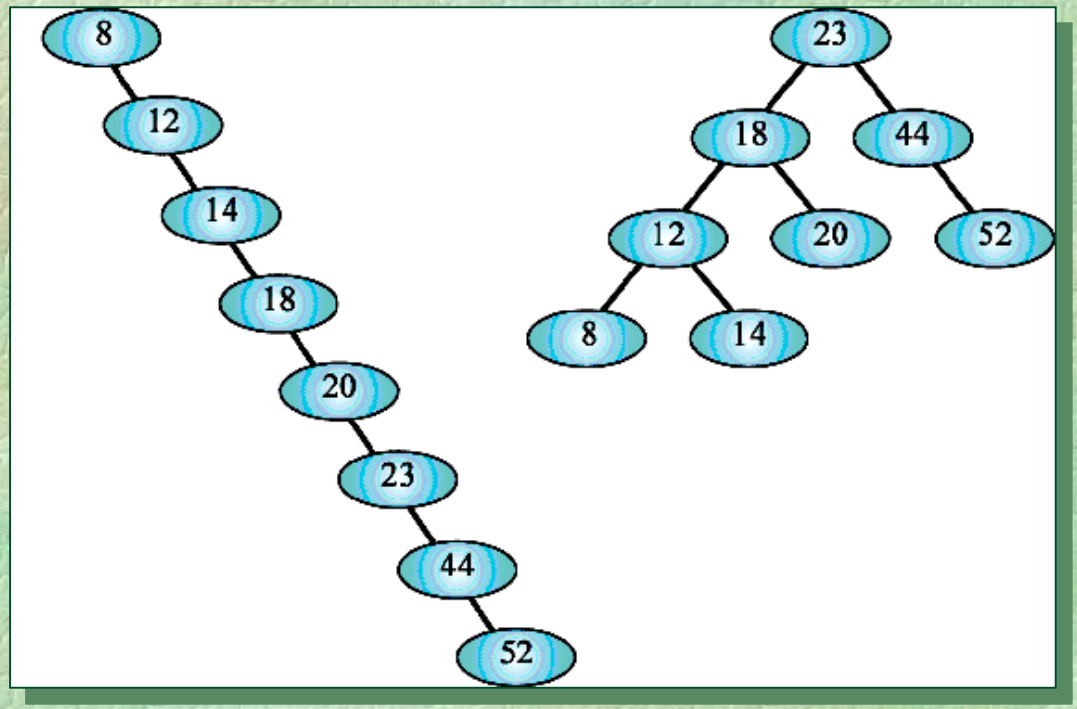
- Preorder: 23 18 12 20 44 35 52
- Postorder: 12 20 18 35 52 44 23
- Inorder: 12 18 20 23 35 44 52

Ζύγισμα δέντρων BST

- Ανάλογα με τη σειρά άφιξης των στοιχείων και τον τρόπο δημιουργίας του δέντρου BST, για τα ίδια στοιχεία δεν προκύπτει πάντα το ίδιο δέντρο
- Το ύψος του δέντρου σχετίζεται με το χρόνο αναζήτησης ενός στοιχείου μέσα στο δέντρο

Ζύγισμα δέντρων BST

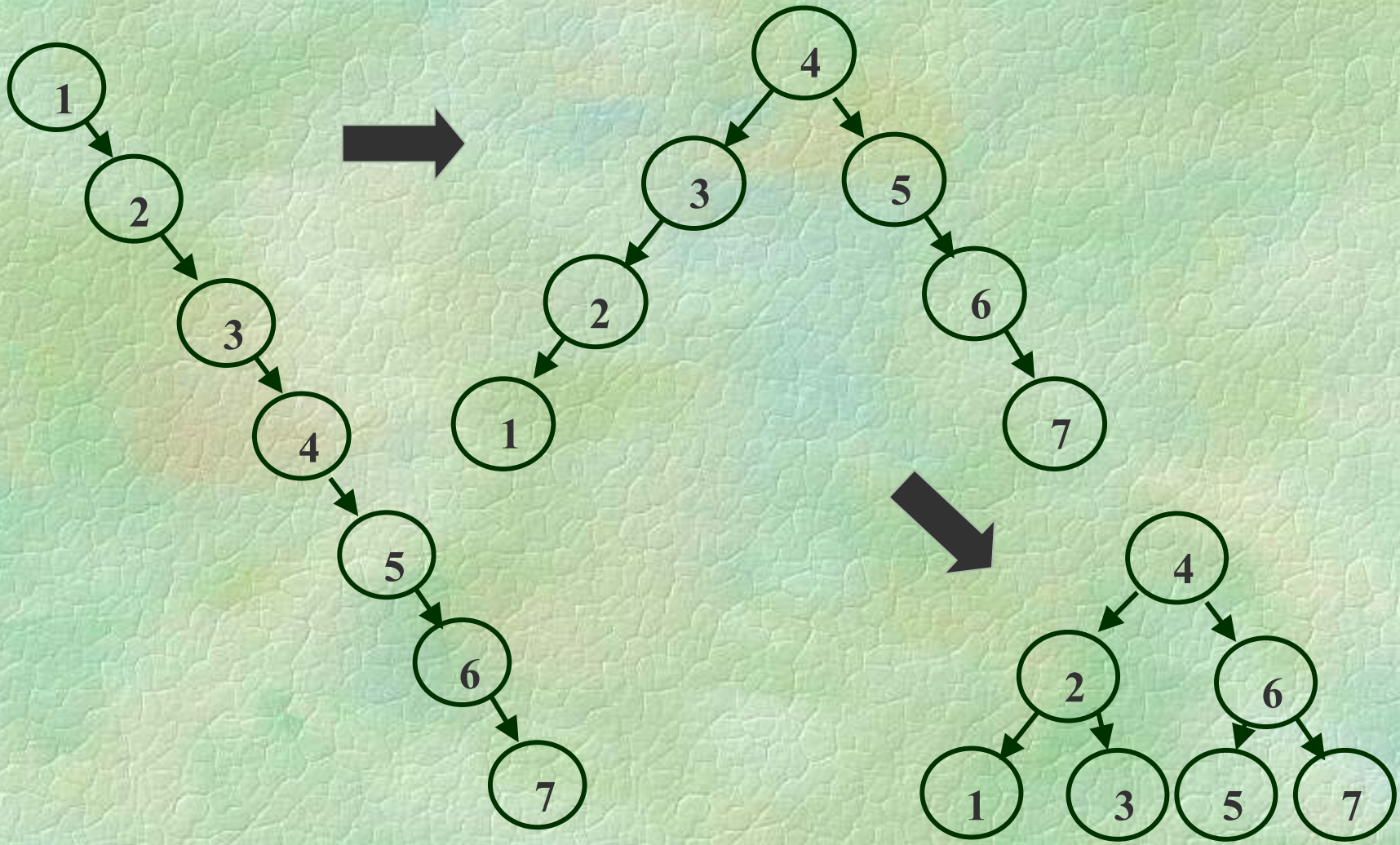
- Αναζήτηση με πολυπλοκότητα
  - Από O(N) [χειρότερη περίπτωση]
  - Μέχρι O(Log_2N) [καλύτερη περίπτωση]



Ζύγισμα δέντρων BST

- Προκειμένου μια αναζήτηση να διατρέχει όσο το δυνατόν μικρότερο μέρος του δέντρου, αυτό πρέπει να είναι «ζυγισμένο»
  - «Ζυγισμένο»: το βάθος του αριστερού υποδέντρου δεν διαφέρει περισσότερο από 1 από αυτό του δεξιού (ιδανικά: είναι ίσα)
- Αν κατά την προσθήκη νέων στοιχείων στο δέντρο η ζυγαριά «γείρει» δεξιά ή αριστερά, απαιτείται διόρθωση του δέντρου

Παραδείγματα ζυγισμένων δέντρων



Ζυγισμένα δέντρα

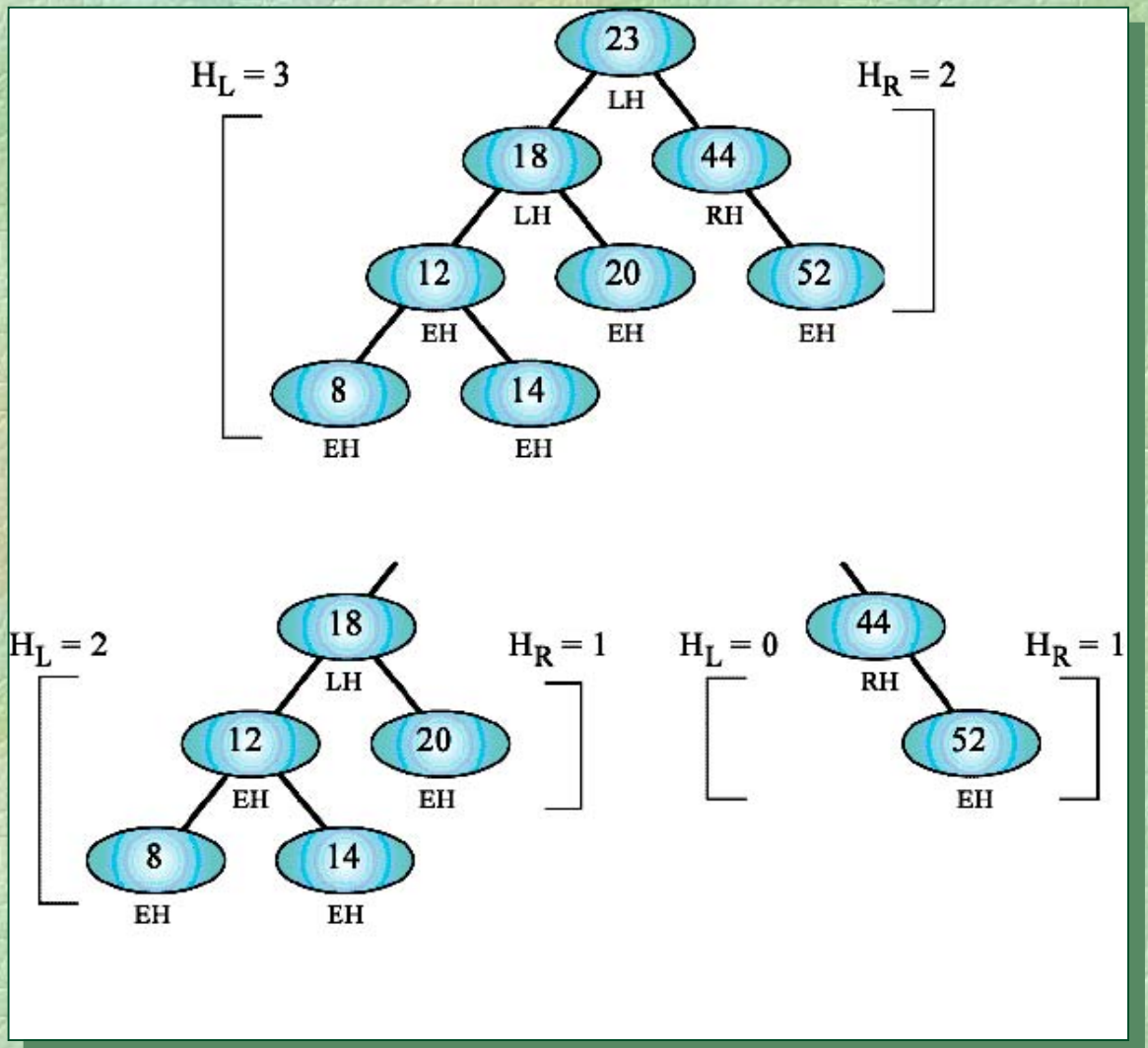
- Η περίπτωση όπου το ύψος του αριστερού υποδέντρου είναι ***ακριβώς ίσο*** με το ύψος του δεξιού, δεν επιτυγχάνεται εύκολα
- Αν από την παραπάνω περίπτωση υπήρχαν **«μικρές» αποκλίσεις**, οι επιδόσεις του δέντρου στην αναζήτηση δεν θα επηρεάζονταν ιδιαίτερα

Δέντρα AVL

- Ενα δέντρο AVL (Adelson-Velskii and Landis) είναι ένα δέντρο BST του οποίου *το ύψος του αριστερού υποδέντρου διαφέρει από αυτό του δεξιού το πολύ κατά 1*
- Το δεξί και αριστερό υποδέντρο ενός δέντρου AVL είναι επίσης δέντρα AVL
- Το κενό δέντρο είναι δέντρο AVL

Οι ιδιότητες αυτές διατηρούνται με παρεμβάσεις (αναδιάταξη) καθώς νέα στοιχεία προστίθενται στο δέντρο

Δέντρα AVL


$H_L = 3$
23
LH
$H_R = 2$
18
LH
44
RH
12
EH
20
EH
52
EH
8
EH
14
EH
$H_L = 2$
18
LH
$H_R = 1$
12
EH
20
EH
8
EH
14
EH
$H_L = 0$
44
RH
$H_R = 1$
52
EH

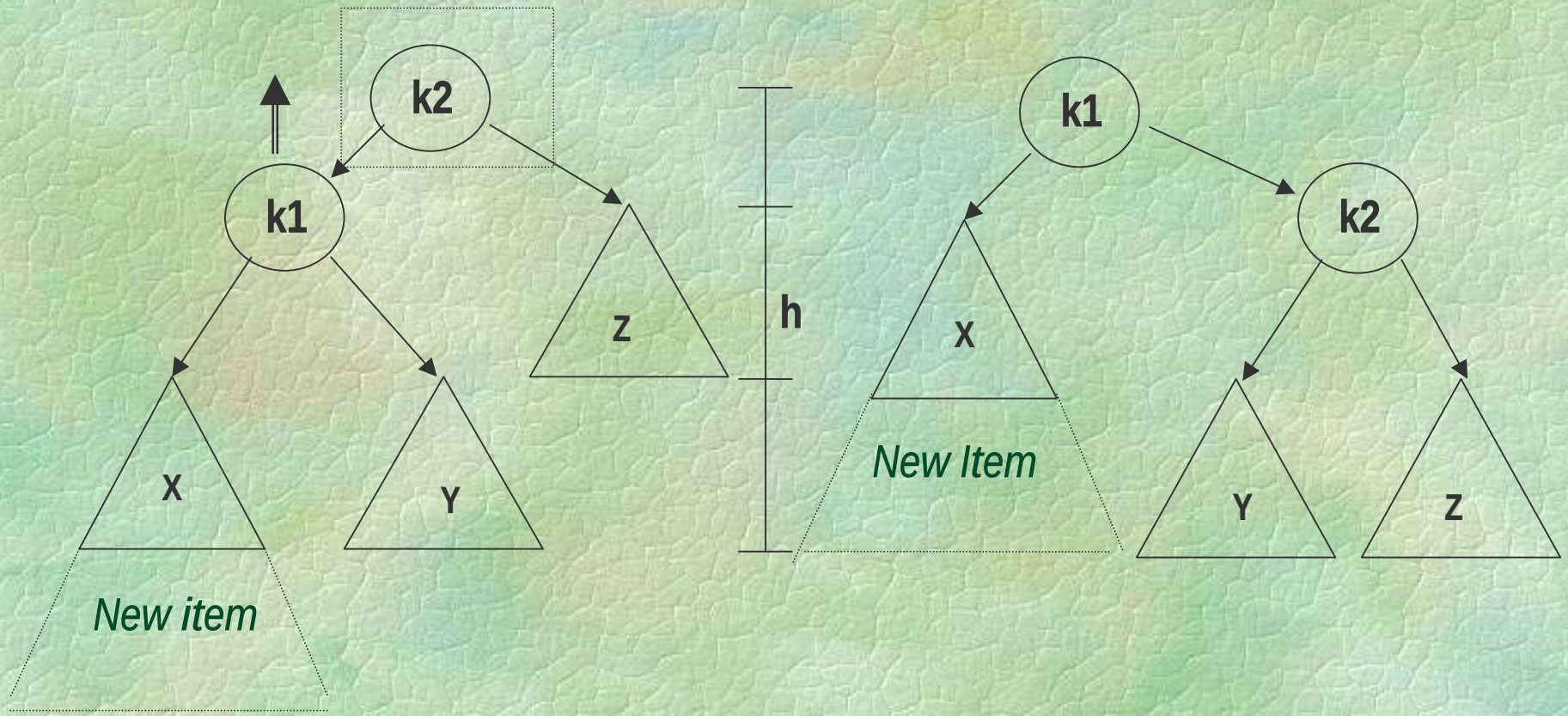
Δέντρα AVL

- Ενα δέντρο AVL λέγεται «ψηλό από αριστερά» όταν το ύψος του αριστερού υποδέντρου είναι μεγαλύτερο από αυτό του δεξιού (κατά πόσο;;;)
- Αντίστοιχα για το «ψηλό από δεξιά»
- Υπάρχουν αρκετοί τρόποι με τους οποίους η προσθήκη ή η διαγραφή στοιχείων σε ένα δέντρο AVL παραβιάζει τη συνθήκη χαρακτηρισμού του

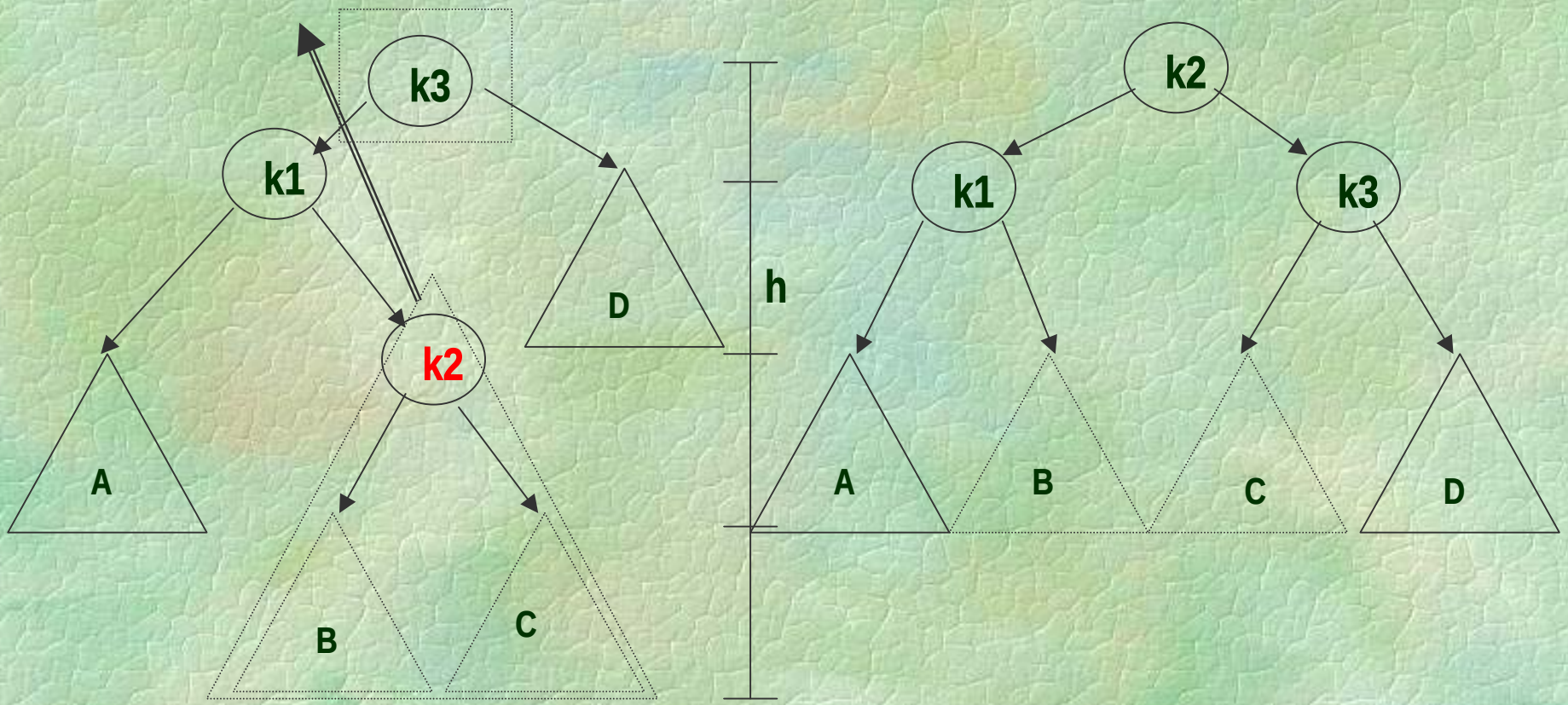
Αναδιάταξη δέντρων AVL

- Η χαρακτηριστική ιδιότητα ενός δέντρου AVL μπορεί να παύει να ισχύει με την προσθήκη νέων στοιχείων
- Η επαναφορά της ιδιότητας σε ισχύ, γίνεται με ***περιστροφή*** του δέντρου, ανάλογα με την περίπτωση αναδιάταξης που συντρέχει

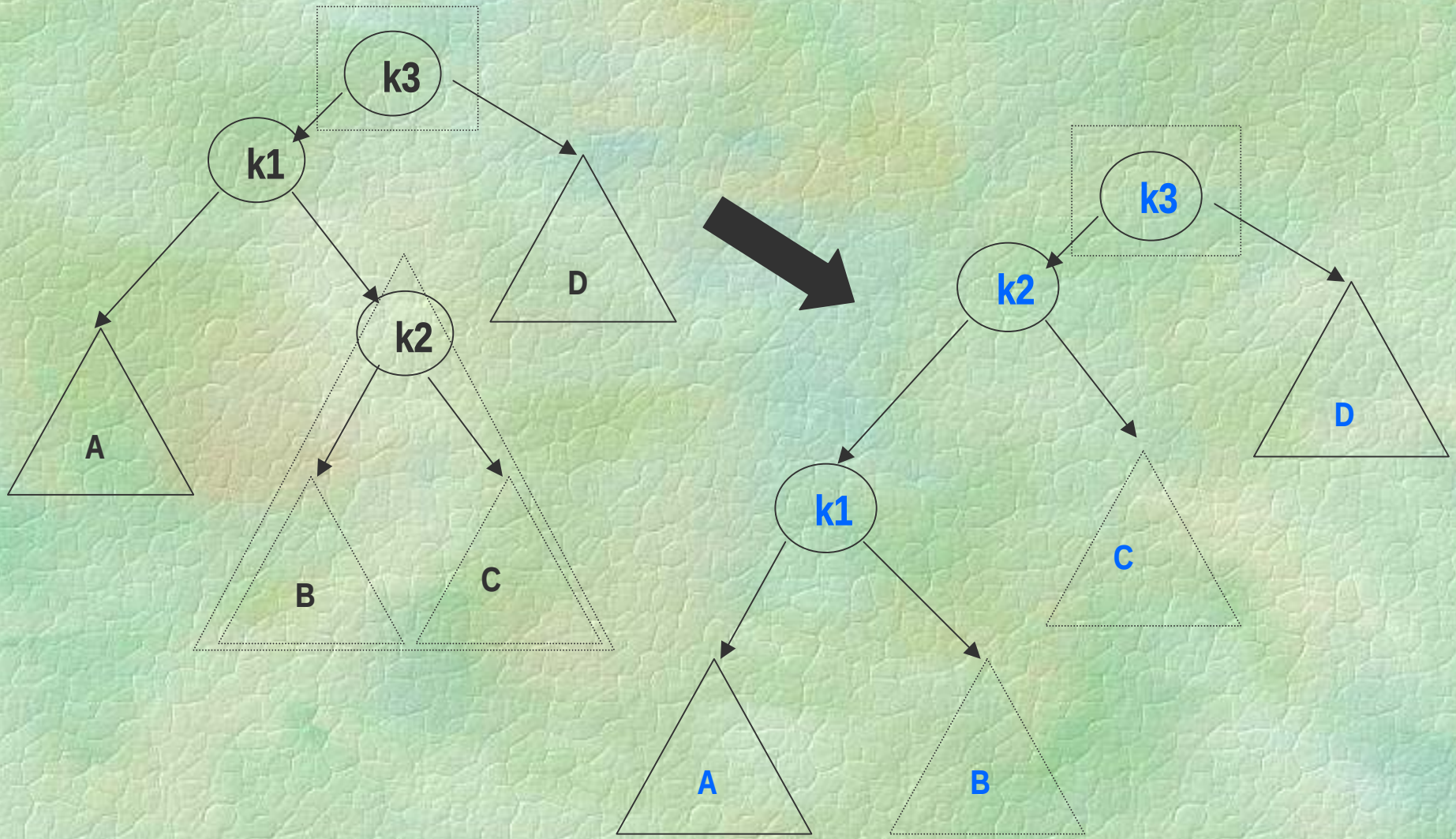
Απλή περιστροφή



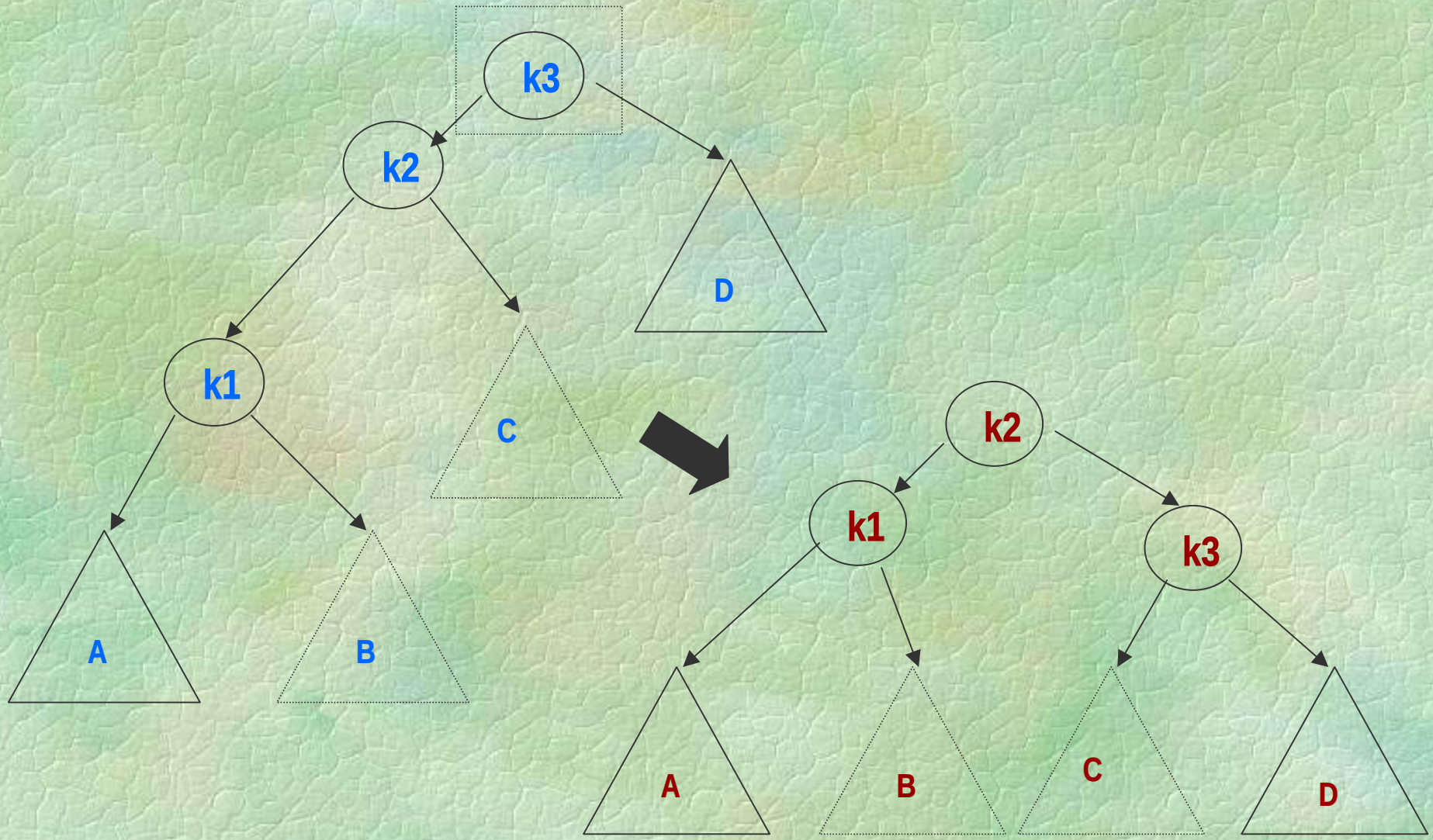
Διπλή περιστροφή (α)



Διπλή περιστροφή (β)



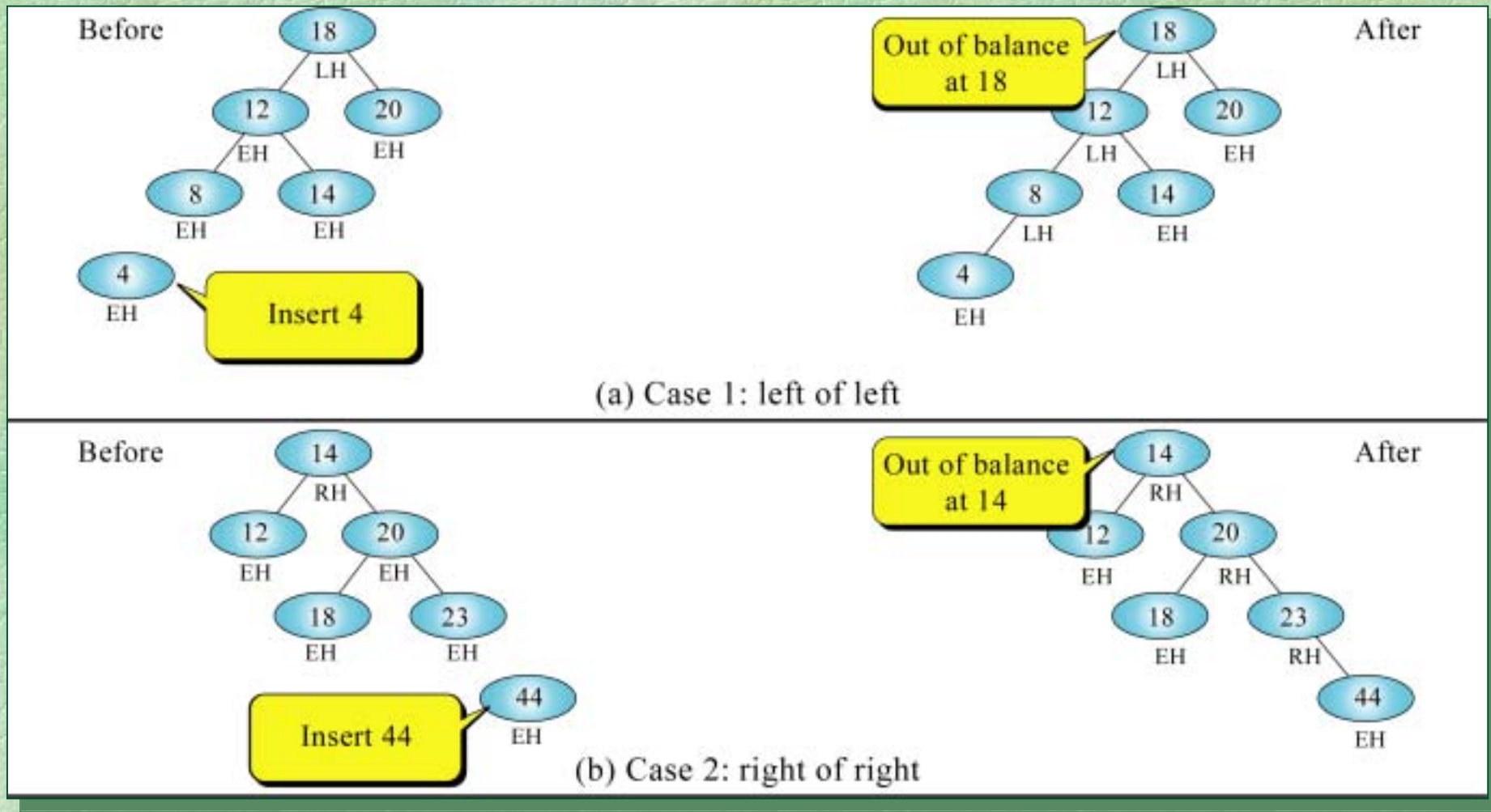
Διπλή περιστροφή (γ)



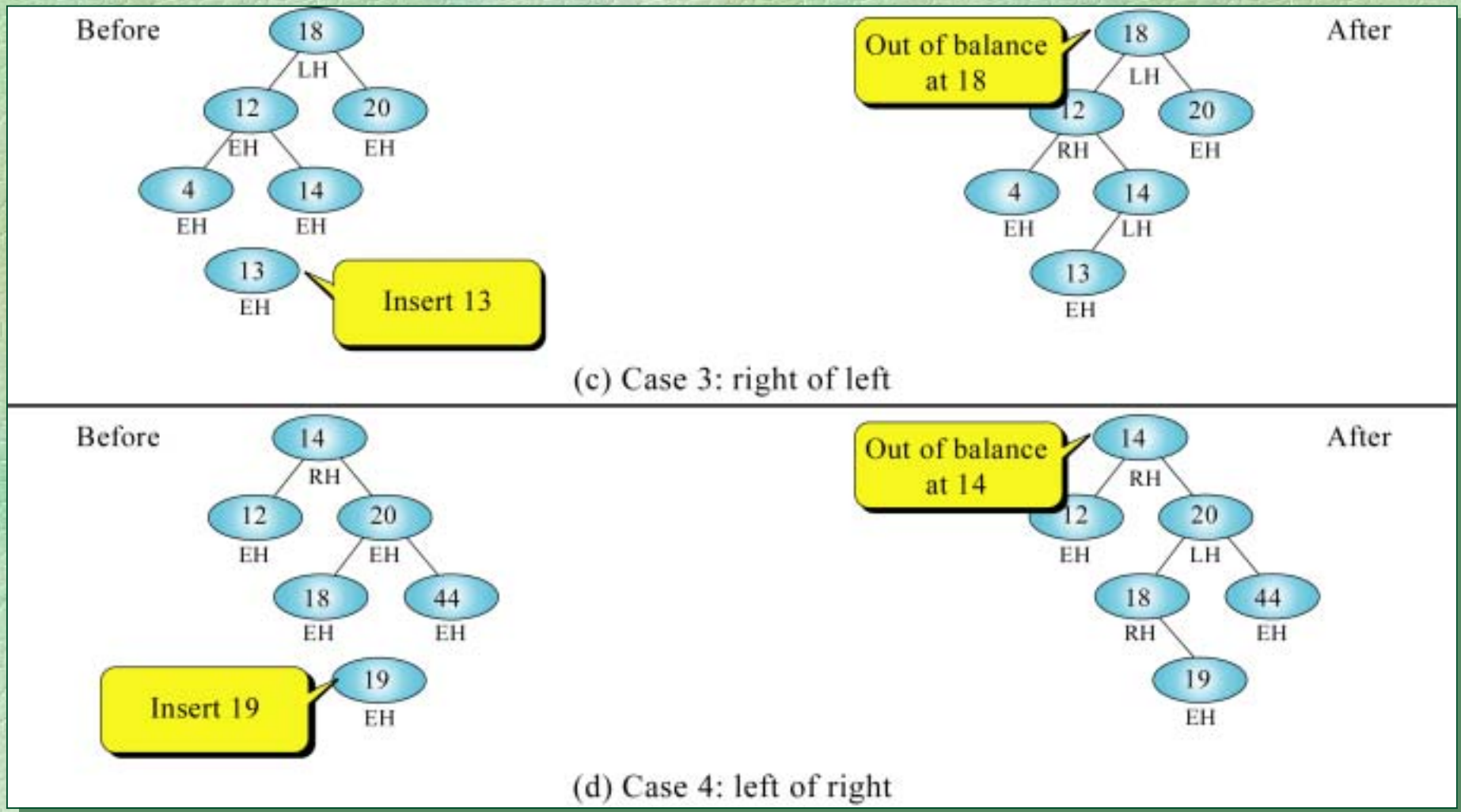
Αναδιάταξη δέντρων AVL

- Περιπτώσεις αναδιάταξης
  - ΑΑ: ένα αριστερό υποδέντρο ενός δέντρου AVL που είναι ψηλό από αριστερά, γίνεται επίσης ψηλό από αριστερά (**left of left**)
  - ΔΔ: τα αντίστοιχα για το δεξί υποδέντρο (**right of right**)
  - ΔΑ: ένα υποδέντρο ενός δέντρου AVL ψηλού από αριστερά, γίνεται ψηλό από δεξιά (**right of left**)
  - ΑΔ: ένα υποδέντρο ενός δέντρου AVL ψηλού από δεξιά, γίνεται ψηλό από αριστερά (**left of right**)

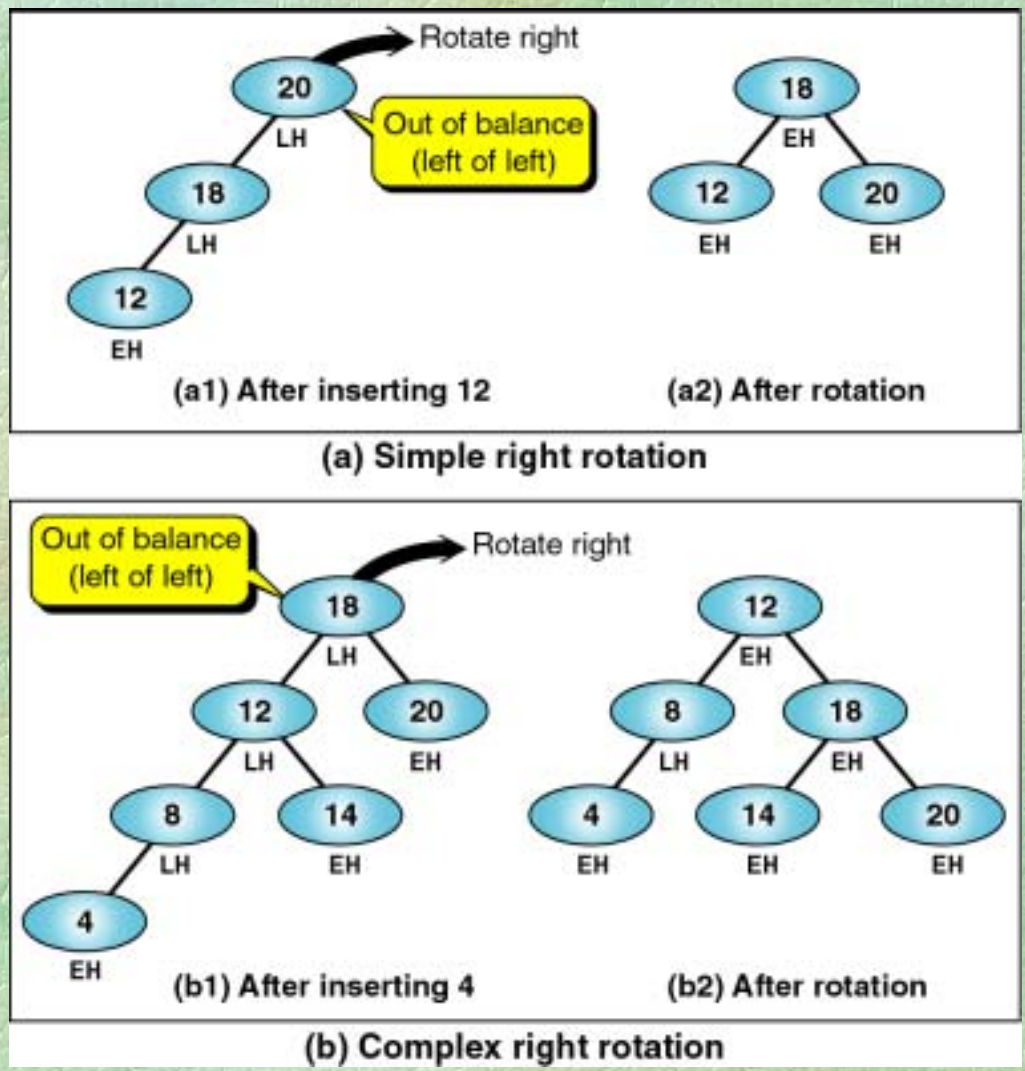
Περιπτώσεις αναδιάταξης (α)



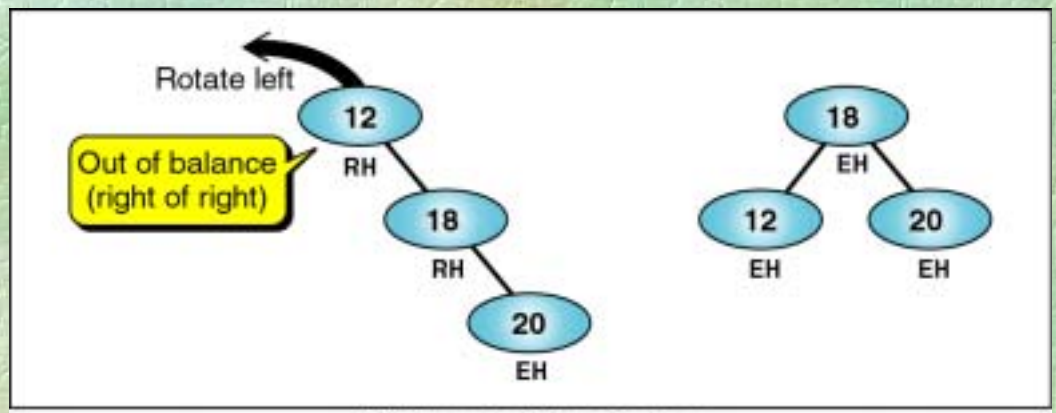
Περιπτώσεις αναδιάταξης (β)



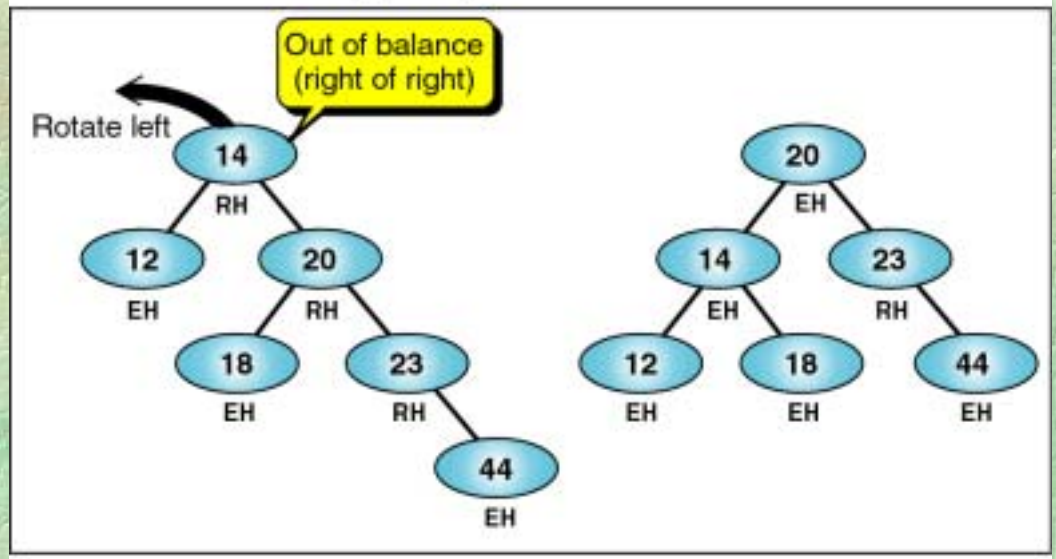
Αναδιάταξη σε περίπτωση AA



Αναδιάταξη σε περίπτωση ΔΔ

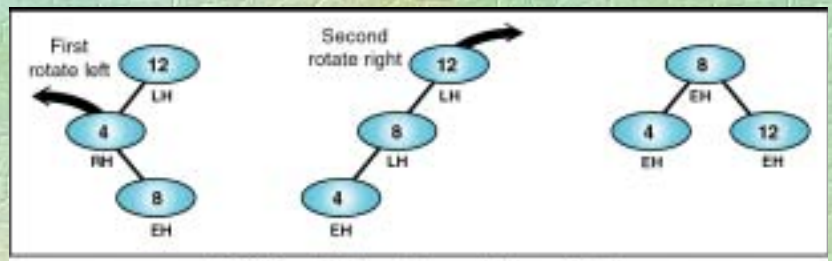


(a) Simple left rotation

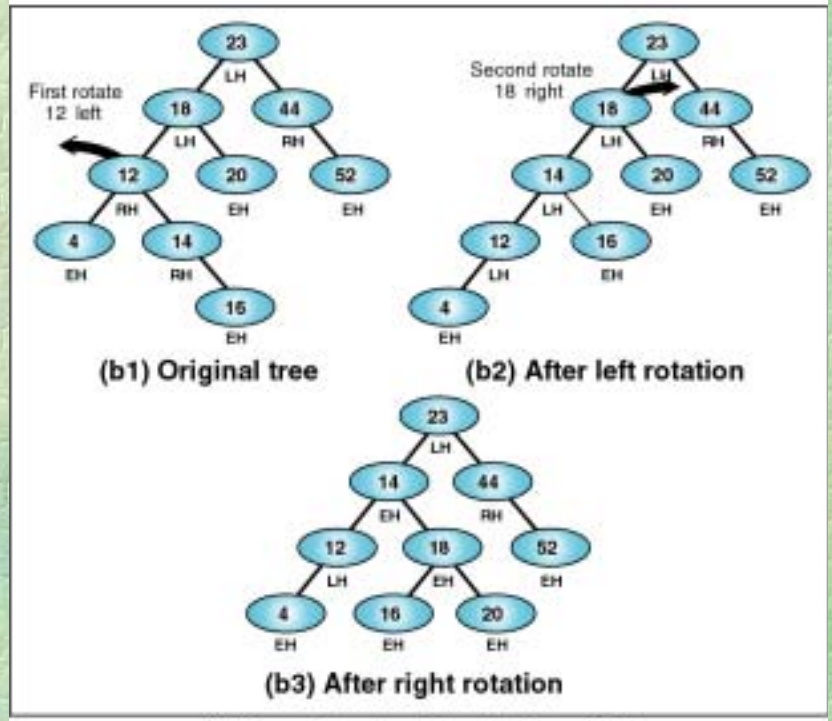


(b) Complex left rotation

Αναδιάταξη σε περίπτωση ΔΑ



(a) Simple double rotation right



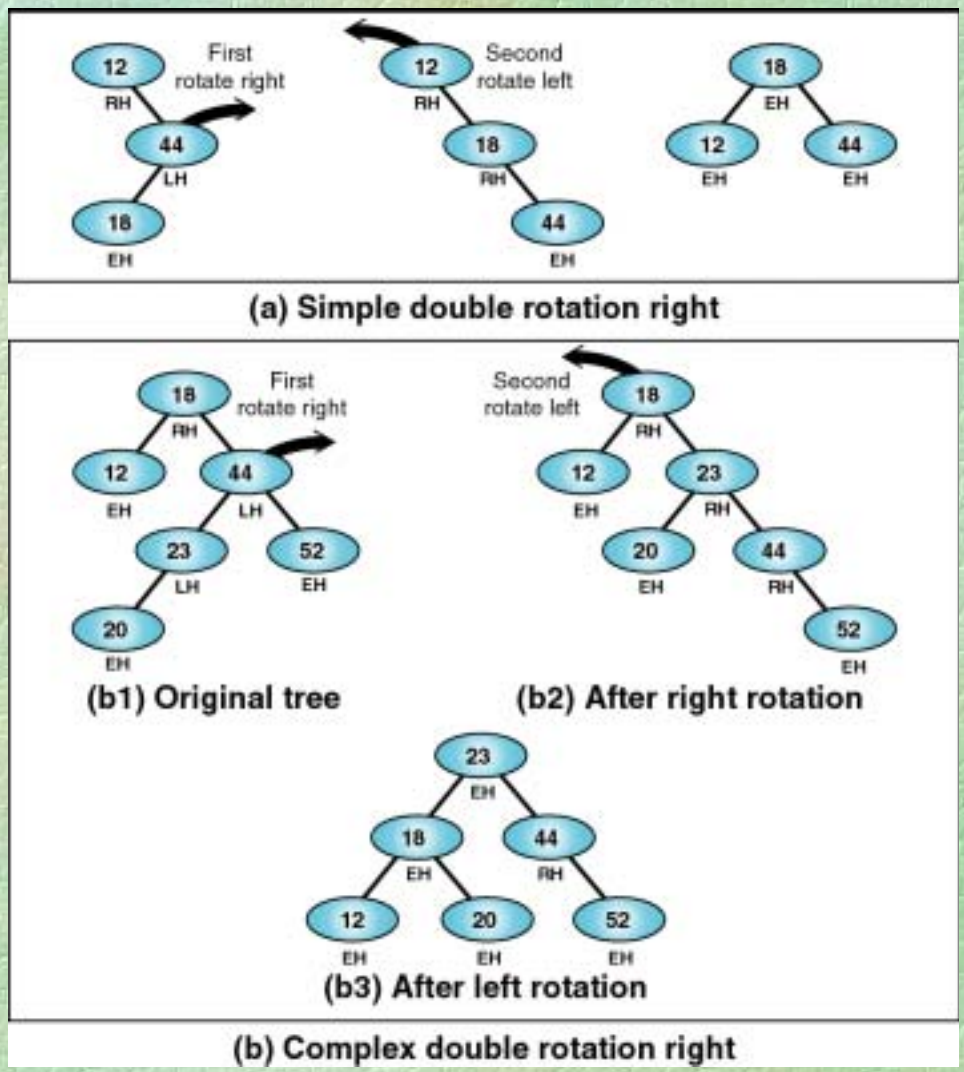
(b1) Original tree

(b2) After left rotation

(b3) After right rotation

(b) Complex double rotation right

Αναδιάταξη σε περίπτωση ΑΔ



Υλοποίηση δέντρων AVL στη C

- Μια δομή δεδομένων

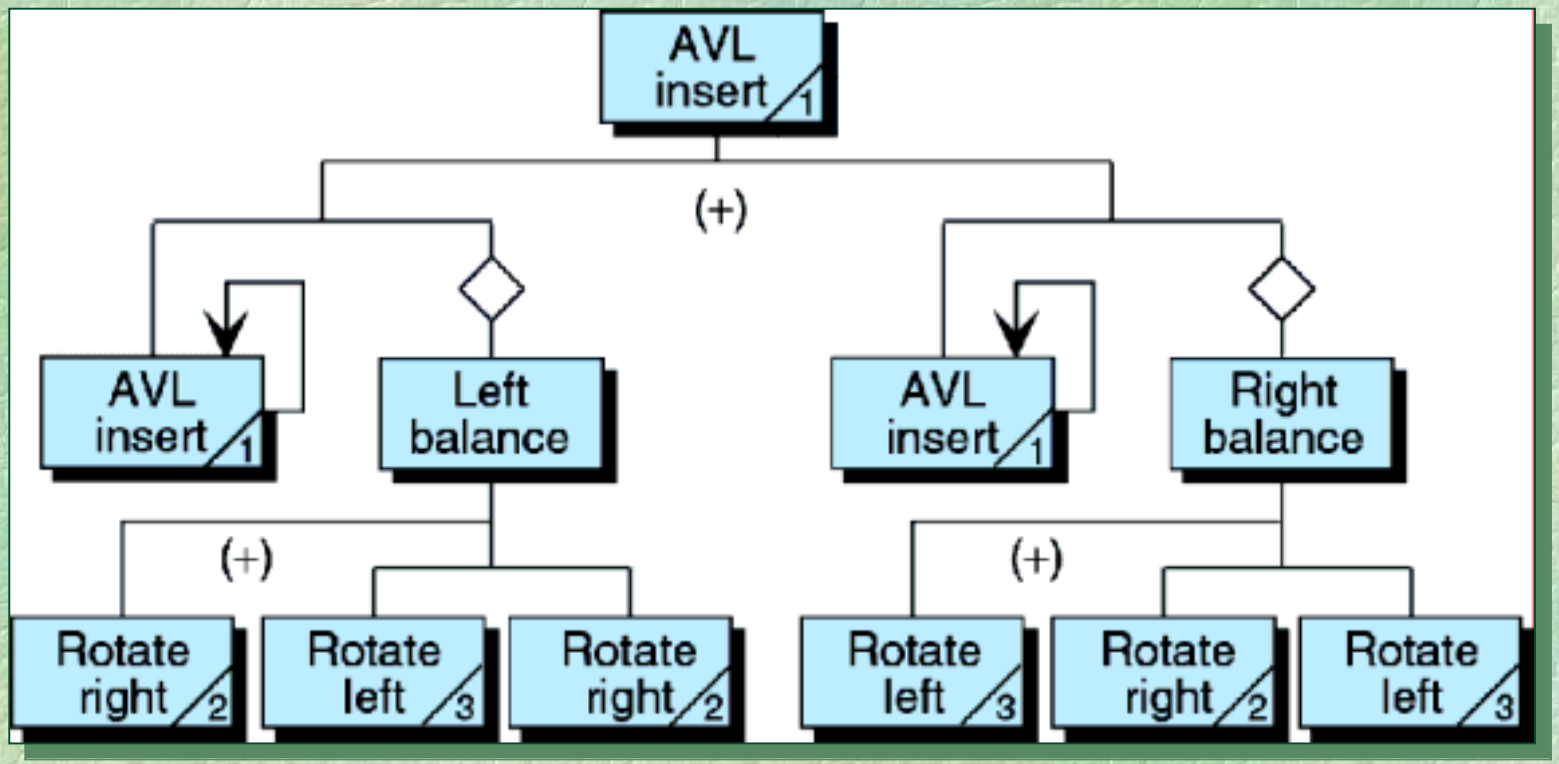
```
Node
        key         <keyType>
        data        <dataType>
        left        <pointer to Node>
        right       <pointer to Node>
        bal         <LeftH, EvenH, RightH>
End Node
```

Εισαγωγή σε δέντρα AVL

- Η εισαγωγή νέων στοιχείων γίνεται στα φύλλα, όπως στα δέντρα BST
- Εντοπίζεται το φύλλο στο οποίο θα γίνει η εισαγωγή και δημιουργείται νέος κόμβος
- Γίνεται οπισθοδρόμηση μέχρι την κορυφή του δέντρου, με έλεγχο της ισχύος της συνθήκης ισορροπίας AVL σε κάθε βήμα προς τα πίσω, και επαναφορά σε ισορροπία, όπου απαιτείται

Εισαγωγή σε δέντρα AVL

- Διάγραμμα κλήσης μιας αναδρομικής συνάρτησης εισαγωγής



ΑΛΓΟΡΙΘΜΟΣ AVLInsert(ref root <tree pointer>, val newPtr <tree pointer>, ref taller <Boolean>)

```
1  if (root null)
   1  root = newPtr
   2  taller = true
   3  return root
2  end if
3  if (newPtr->key < root->key)
   1  root->left = AVLInsert (root->left, newPtr, taller)
   2  if (taller)                              // Left subtree is taller
      1  if (root left-high)
         1  root = leftBalance (root, taller)
      2  elseif (root even-high)
         1  root->bal = left-high
      3  else                                  //Was right high -- now even high
         1  root->bal = even-high
         2  taller = false
      4  end if
4  else                                        // New data >= root data
```

(Συνέχεια) **AVLInsert(ref root <tree pointer>, val newPtr <tree pointer>, ref taller <Boolean>)**

```
...
4  else                                    // New data >= root data
   1  root-> right = AVLInsert (root->right , newPtr, taller)
   2  if (taller)                          // Right subtree is taller
      1  if (root left-high)
         1  root->bal = even-high
         2  taller = false
      2  elseif (root even-high)           // Was balanced -- now right high
         1  root->bal = right-high
      3  else
         1  root = rightBalance (root, taller)
      4  end if
   3  end if
5  end if
6  return root

end AVLInsert
```

ΑΛΓΟΡΙΘΜΟΣ

leftBalance (ref root <tree pointer>, ref taller <Boolean>)

```
1  leftTree = root-> left
2  if (leftTree left-high)
     // Case 1: Left of left.  Single rotation right.
   1  rotateRight (root)
   2  root->bal = even-high
   3  leftTree->bal = even-high
   4  taller = false
3  else
     // Case 2: Right of left.  Double rotation required.
   1  rightTree = leftTree->right
     // adjust balance factors
   2  if (rightTree->bal left-high)
      1  root->bal = right-high
      2  leftTree->bal = even-high
   3  elseif (rightTree->bal = even-high)
      1  leftTree->bal = even-high
   4  else
```

(Συνέχεια)
leftBalance (ref root <tree pointer>, ref taller <Boolean>)

```
 4  else
        // rightTree->bal is right-high
        1  root->bal = even-high
        2  leftTree->bal = left-high
    5  end if
    6  rightTree->bal = even-high
    7  root->left = rotateLeft (leftTree)
    8  root = rotateRight (root)
    9  taller = false
 4  end if
 5  return root

end leftBalance
```

Περισσότερα για δέντρα AVL

- Διαγραφή στοιχείου
  - Διαβάστε αντίστοιχο download από τη σελίδα του μαθήματος
- Βιβλιοθήκη συναρτήσεων
  - Μπορείτε να κατεβάσετε ένα σύνολο έτοιμων συναρτήσεων C για το χειρισμό δέντρων AVL από τη σελίδα web του μαθήματος