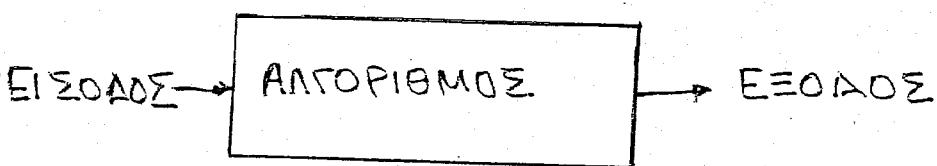


ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

- ΑΝΤΩΝΙΟΣ ΣΥΜΒΩΝΗΣ
ΑΝΑΠΛΗΡΩΤΗΣ ΚΑΘΗΓΗΤΗΣ
3.91
- Δομές Δεδομένων - Ταχινόμητη και Αναζήτηση με Java
ΠΑΝΑΓΙΩΤΗΣ ΗΠΟΖΑΝΗΣ
ΕΚΑ. ΤΖΙΟΛΑ
- ΔΕΥΤΕΡΑ, ΤΡΙΤΗ $8^{45} - 10^{30}$
- Αρθ. - 1
- www.math.ntua.gr/~symvonis

ΕΙΣΑΓΩΓΗ

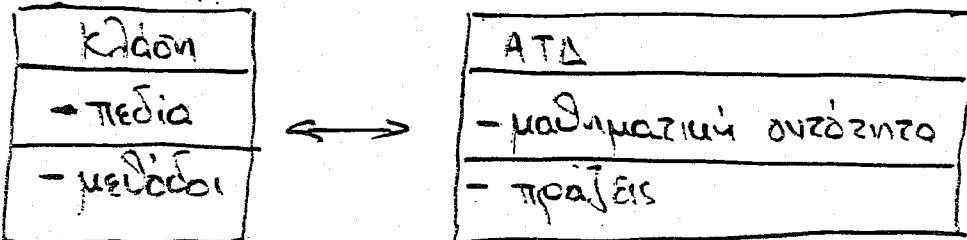
- ΑΛΓΟΡΙΘΜΟΣ: Μια ανοδούσια υπολογιστική δημιουργία που συντάσσεται από μια σειρά ειδοποιήσεων και διενέργειας συναρτήσεων που συνδέονται μεταξύ τους για να λύσει ένα πρόβλημα.



- Νόμιμη εισόδους: Ιανοποιεί τις προδιαγραφές του προβλήματος
- Στιχυμότυπο
- ΑΦΗΡΗΜΕΝΟΣ ΤΥΠΟΣ ΔΕΔΟΜΕΝΟΝ (ΑΤΔ)
[Abstract data type - ADT]

• Μια μαθηματική οντότητα με μια συλλογή πράξεων επί των στοιχείων της

- Αντιστοιχία αλγόριθμου - ΑΤΔ



• ΟΡΘΟΤΗΤΑ ΑΛΓΟΡΙΘΜΩΝ

• Επαγγή [induction]

[Ισχυρή επαγγή - strong induction]

Έστω μια πρόσοση π , η οποία εξαρτάται από ένα φυσικό αριθμό $n \geq n_0$.

Εάν αποδειχθεί ότι π :

(a) ισχύει για $n = n_0$, και

(b) ισχύει για $n = k+1$ εφόσον ισχύει για κάθε $n \leq k$
τότε π ισχύει για καθε $n \geq n_0$

• Αμεράβδητη συνθήκη βρόχου [loop invariant]

Πρόσοση που παραμένει αληθής μετά την ολοκλήρωση
καθε επανάληψης του βρόχου.

Algorithm: Max(A)

Input: Διάνυσμα A A.length αυτορίων

Output: Ο μεγαλύτερος αυτοριών του A

1 max = A[0];

2 for ($i=1$; $i < A.length$; $i++$)

3 if ($max < A[i]$)

4 $max = A[i]$

5 return max

"Κατά το λειτουργεί της i-οσής επανάληψης, η μεγαλύτερη
max αποτελείται από το μεγαλύτερό σε αξίαν την i πράξη
επεξετώντας τα τιμές"

ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

MAX(A)

```

1   max = A[0];
2   for ( i=1; i < A.length; i++ )
3       if ( max < A[i] )
4           max = A[i]
5   return max

```

Γραμμή	Κοστός	# Επελέγεων
1	c_1	1
2	c_2	n
3	c_3	$n-1$
4	c_4	$\leq n-1$
5	c_5	1

Συνολικός χρόνος: $\leq c_1 + c_2n + c_3(n-1) + c_4(n-1) + c_5 \leq$

$$T(n) \leq (c_2 + c_3 + c_4)n + (c_1 - c_3 - c_4 + c_5)$$

Εξαρτάται από το μέγεθος της εισόδου [n]

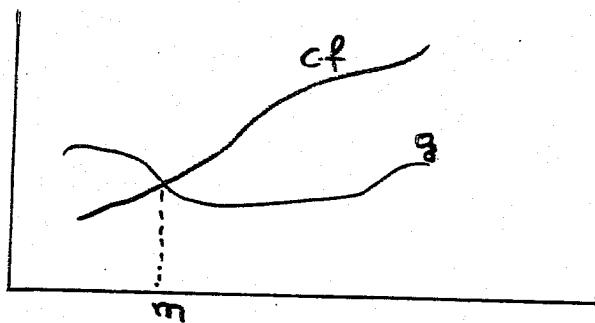
ΑΣΥΜΠΤΟΤΙΚΗ ΑΝΑΛΥΣΗ

- Ραδικός ανάπτυξης / τάξη ανάπτυξης [rate/order of growth]

Εσώ μία συνάρτηση $f: \mathbb{N} \rightarrow \mathbb{N}$

$$O(f) = \left\{ g: \mathbb{N} \rightarrow \mathbb{N} \mid \exists c > 0, m: g(n) \leq c f(n), \forall n \geq m \right\}$$

= όλες οι συνάρτησης που έχουν ασυμπτωτικό όρο
οπιο (asymptotic upper bound) των f .



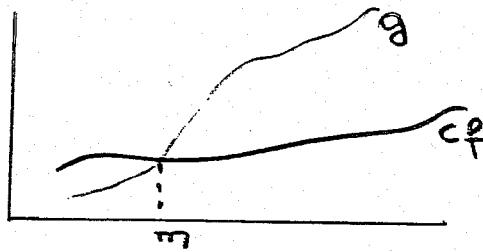
- $5n = O(n)$
- $5n + 1000 = O(n)$
- $n = O(n \log n)$
- $\log n = O(n)$

• $O(f)$

- $f(n) = O(g(n))$ και $h(n) = O(g(n)) \Rightarrow c_1 f + c_2 h = O(g(n))$
- $f(n) = O(g(n))$ και $h(n) = O(d(n)) \Rightarrow f h = O(gd)$
- μεταβατικότητα

$$\Omega(f) = \{g: \mathbb{N} \rightarrow \mathbb{N} \mid \exists c > 0, m : g(n) \geq c f(n), \forall n \geq m\}$$

= όλες οι συναρτήσεις για τις οποίες n f συνιστά ασυμπτωτικό υψηλό όριο (asymptotic lower bound)

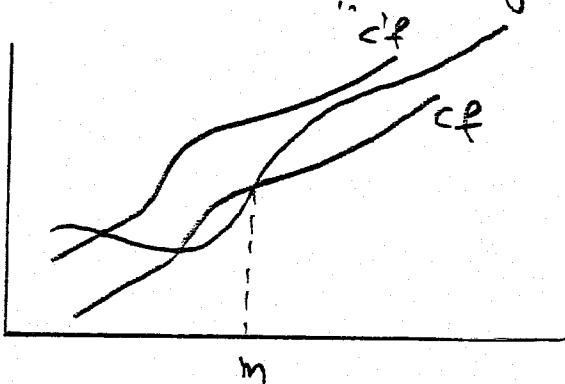


• $\omega(f)$

- $5n = \Omega(n)$
- $n^2 = \Omega(n)$
- $5n - 1000 = \Omega(n)$

$$\Theta(f) = \{g: \mathbb{N} \rightarrow \mathbb{N} \mid \exists c, c' > 0, m : c f(n) \leq g(n) \leq c' f(n), \forall n \geq m\}$$

= όλες οι συναρτήσεις που αποτελούν ασυμπτωτικό "σφιγγό" όριο (asymptotic tight bound) για f .



ΠΟΛΥΠΛΟΚΟΤΗΤΑ ΠΡΟΒΛΗΜΑΤΟΣ

- Άνω όρος : υποδιέξει από του "καλύτερο"
algorίθμο επίγειος του
- κάτω όρος : κάθε algorίθμος που το επιλέγει
(lower bound) έχει τουλάχιστου τέτοια πολυπλοκότητα

Παράδειγμα: Ταξιδόμινο

$$\left. \begin{array}{l} \text{Άνω όρος: } O(n \log n) \rightarrow \text{Merge Sort} \\ \text{κάτω όρος: } \Omega(n \log n) \end{array} \right\} \Rightarrow$$

$$\Theta(n \log n)$$

Πολυπλοκότητα algorίθμων.

- Ενας algorίθμος έχει πολυπλοκότητα $\Theta(f)$, εάν:
 - ζερναίζει μεριά από $O(f)$ χρόνο
 - Υπάρχει ένα σχήμαστο του προβλήματος που αναγνάπτει τον algorίθμο να επεξεργάζεται ότι $\Omega(f)$ χρόνο.

• Ανάδοση χειρότερης περιπτώσεων (worst case)

Algorithm: FIND (A, x)

Input: διάνυσμα A, αριθμός x προς αναζήτηση

Output: Η δεσμή του x στο A, -1 εάν x ∉ A

```

1. location = -1; found = false; l=0;
2. while ( i < A.length AND found == FALSE)
3.       if (A[i] == x)
4.           location = i
5.           found = TRUE
6. return location
    
```

Worst-case : n Bisezta $\Rightarrow \Theta(n)$

• Ανάδοση μέσης περιπτώσεων

• Εάν $x \in A \Rightarrow \frac{n}{2}$ Bisezta $\Rightarrow \Theta(n)$

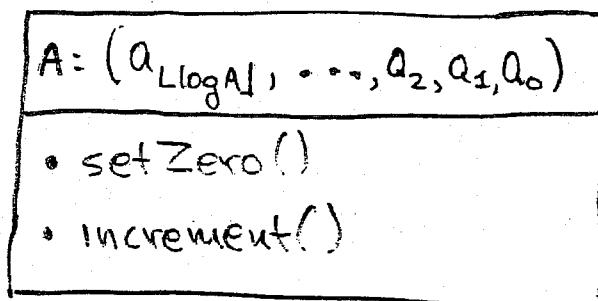
• Εάν $x \notin A \Rightarrow n$ Bisezta $\Rightarrow \Theta(n)$

- Ανάδυον Επιμερισμένης / Κατανεμημένης Περίπτωσης
[amortized case analysis]

- $T(n)$ = μέγιστος χρόνου εκτελέσεων μιας αποιαδήποτε αυθορμίας η πρόβεση επι μιας δορυς

$$\Rightarrow \frac{T(n)}{n} = \text{επιμερισμένος χρόνος για την πράξη}$$

Παράδειγμα: ΑΤΔ: Δυαδικός μεγρυτής



Κίνος: αλλαγή ψηφίου του A

Αυθορμία: setZero(), increment(), $\underbrace{\dots, \dots}_{\#n}$, increment()

Worst case: Αν $A=i \rightarrow \text{increment}() : \lfloor \log i \rfloor + 1 + 1$ αλλαγές ψηφίου

$$\Rightarrow \text{Σύνολος κίνων} \leq \sum_{i=0}^{n-1} (\lfloor \log i \rfloor + 1 + 1) \leq n \log n + 2n$$

- Μέθοδος λογαριασμού τραπεζής (banker account method)

- κάθε πράξη χρεωνεται ενα επιμερισμένο μόσχος
το οποίο ~~τους~~ είναι μικρότερο ή μεγαλύτερο από το
πραγματισμό

- Η διαφορά ανάμεσα στο πραγματισμό και το
επιμερισμένο μόσχο χαρακτηρίζεται ως:
πίσωση (credit) ή δόνειο.

Παράδειγμα: Χρεώνουμε 2 μονάδες για να δίσουμε
ενα φυσικό συνη σημ "1"

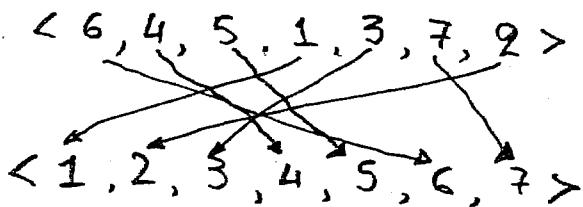
⇒ Επιμερισμένο μόσχος της increment(1) = 2

[το πολ δίνει φυσικό σημερι από "0" → "1"]

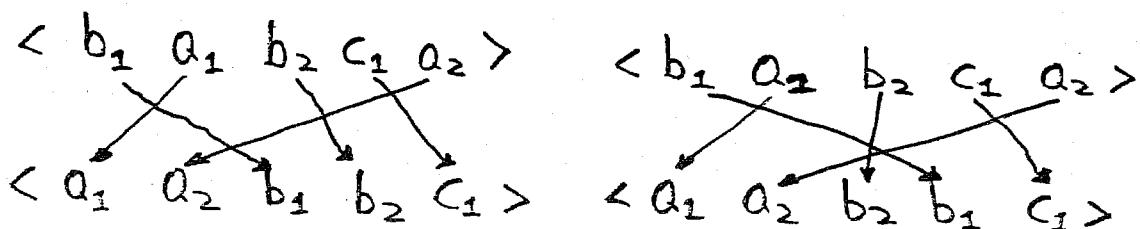
⇒ μόσχος αυθοριδιας $O(n)$

ΤΑΞΙΝΟΜΗΣΗ

- Ταξινόμηση - Σίεραζου (sorting)



- Σταθερή (stable) - Ασταθής (unstable) ταξινόμηση



- Συγκριτικοί (comparison based)

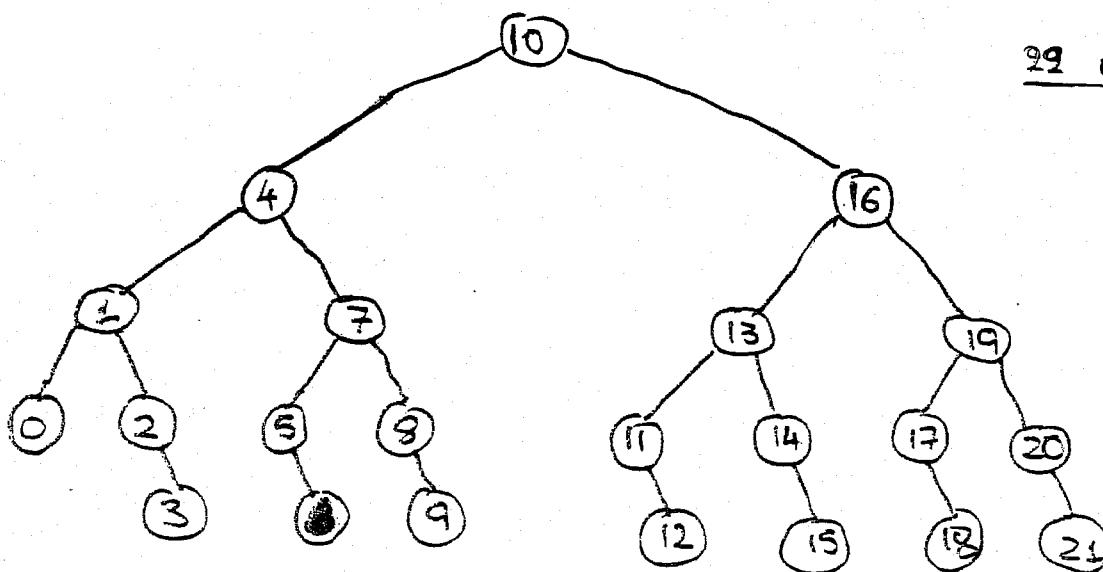
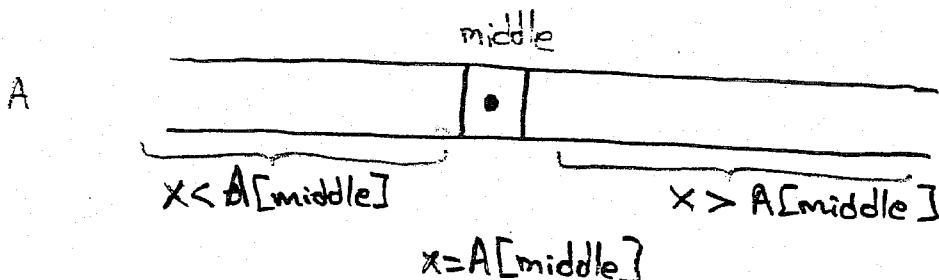
- Διανομής μεταβολής (distribution based)

ΔΥΑΔΙΚΗ ΑΝΑΖΗΤΗΣΗ

- Εισόδος:
- Διατεταγμένους πίνακας αντιτιθένων
 - Αντιτιθένο x προς αναζήτηση

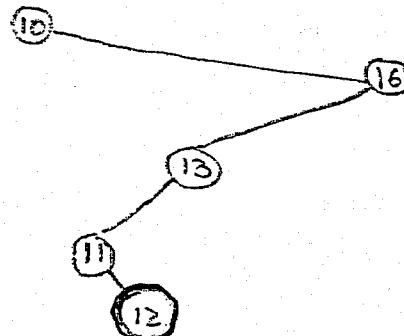
Εξόδος: Η δέση του x στον πίνακα, (-1 εάν το x δεν αντιτίθεται στον πίνακα)

- Μέθοδος: Συγχρίνε το x με το μεσαίο στοιχείο του πίνακα.
- " $=$ " : βρίσκεται.
 - " $x >$ " : ψάχνεται δεξιά την πύρα.
 - " $x <$ " : ψάχνεται αριστερά την πύρα.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
2	4	5	8	9	13	17	41	42	48	50	55	57	59	61	66	67	72	74	75	87	88

x=57



```

int binarySearch ( Object a[], Object x, Comparator c )
{
    int left = 0;
    int right = a.length - 1;
    int middle = 0;
    while ( right >= left ) {
        middle = ( left + right ) / 2 ;
        if ( c.equal ( x, a[middle] ) )
            return middle;
        if ( c.less ( x, a[middle] ) )
            right = middle - 1;
        else
            left = middle + 1;
    }
    return -1;
}
  
```

```
static class Comparator // για Integer
{
    boolean less (Object i, Object j) {
        return (((Integer)i).intValue()) < (((Integer)j).intValue());
    }
}
```

```
boolean equal (Object i, Object j) {
    return (((Integer)i).intValue()) == (((Integer)j).intValue());
}
}
```

- Interface Comparable

int compareTo (Object x) → -1, 0, +1
 $< x = x > x$

int binarySearch (Comparable a[], Comparable x)

ΑΞΗΣΗ: Να γραψει αναδρομικη υλοποίηση
 των μεθόδων binarySearch

ΘΕΩΡΗΜΑ: Η χρονική πολυπλοκότητα, σε πλήρες ουγγρίσεων, της μέθοδου binarySearch περιγράφεται από την ανέφοδη σχέση

$$T_b(n) \leq T_b\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1 , \quad T_b(1) = 1$$

η οποία έχει ως θέμα

$$T_b(n) \leq \lfloor \log n \rfloor + 1 = O(\log n)$$

ΑΠΟΔΕΙΚΗ [το n είναι δυνάμη του 2]

$$\begin{aligned} T_b(n) &\leq T_b\left(\frac{n}{2}\right) + 1 \\ T_b\left(\frac{n}{2}\right) &\leq T_b\left(\frac{n}{4}\right) + 1 \\ &\vdots \\ T_b\left(\frac{n}{2^{\log n - 1}}\right) &\leq T_b\left(\frac{n}{2^{\log n}}\right) + 1 \end{aligned} \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} \text{τελευταίων}$$

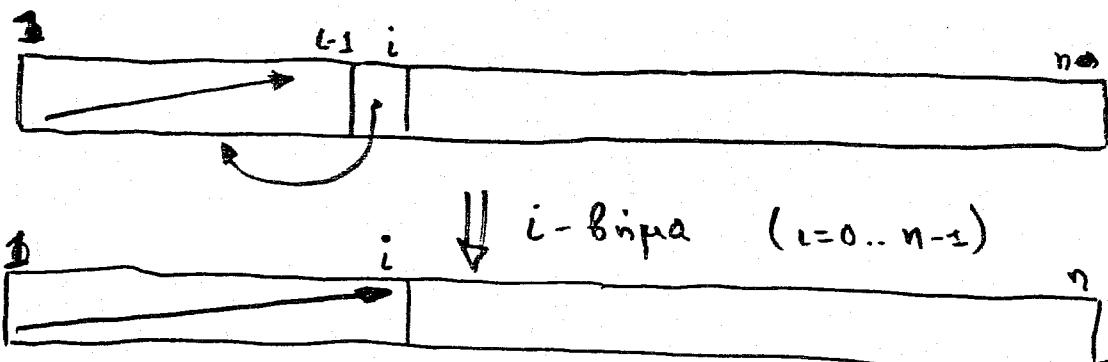
$$T_b(n) \leq 1 + \log n$$

Υπόσχεση για γενέτειν: σελ 67-73

$$\begin{aligned} \text{Άριθμος} \quad 3.2 - 3.4 \\ (3.5 - 3.7)^x \end{aligned}$$

ΣΥΓΚΡΙΤΙΚΟΙ ΑΛΓΟΡΙΘΜΟΙ ΤΑΞΙΝΟΜΗΣΗΣ

ΤΑΞΙΝΟΜΗΣΗ ΕΙΣΑΓΩΓΗΣ (Insertion Sort)



```

static void insertionSort (Object a[], int left, int right,
                          comparator c)
{
    for (int i = left+1; i <= right; i++)
    {
        Object temp = a[i];
        int j = i; // j: υποψήφια θέση για temp
        while ((j > left) && (c.less(temp, a[j-1])))
        {
            a[j] = a[j-1];
            j--;
        }
        a[j] = temp;
    }
}
  
```

To $a[j]$ είναι πάντα "άδειο"

\Downarrow

Ανάλυση πολυπολύτων

Χειρότερη περίπτωση: Το διάνομα είναι τελικοπρόγονο κατέ φύουσα διάταξη

Kατά την εξίσωση του i-thου συντελεστή της διαδικασίας μήδε ολη η σειρά του προσθένται \Leftrightarrow $i-1$ διαγραφές.

$$\text{Συντελεστή: } 1 + 2 + 3 + \dots + n-1 = \frac{n(n-1)}{2} \text{ συγχρίσις}$$

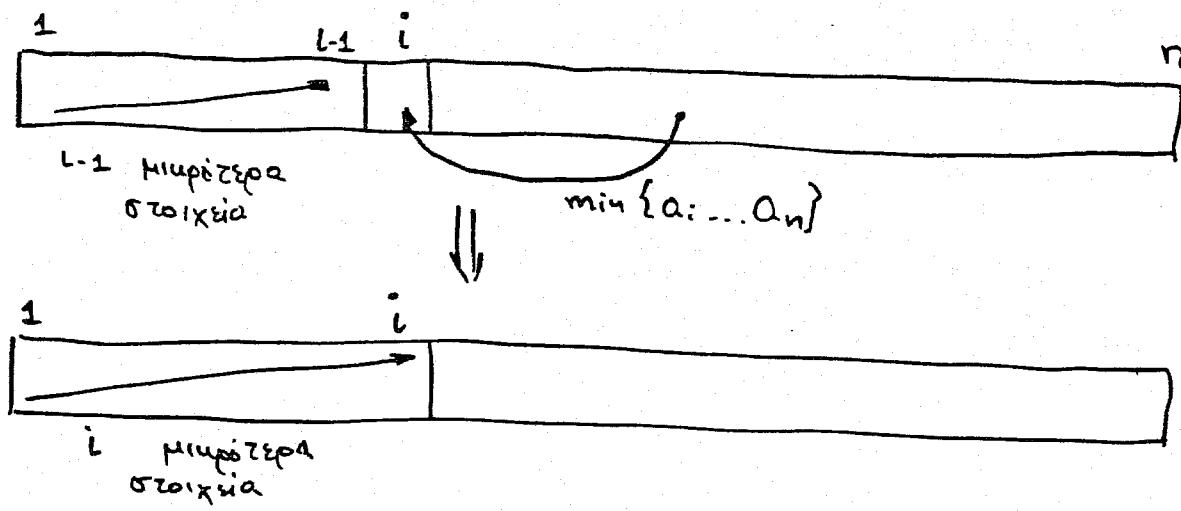
$$= O(n^2)$$

Μέλος Ηερίπτωσης: Αναπτύσσεται οτι το i -οτού στοιχείο είναι μεγαλύτερο από τα μηδέ στοιχεία του ήδη διατάξιμου τμήματος

$$\Rightarrow \frac{1}{2} + \frac{2}{2} + \frac{3}{2} + \dots + \frac{(n-1)}{2} = \frac{n(n-1)}{4} = O(n^2)$$

Απ: Insertion sort : $O(n^2)$
 Ε παραδίκηρα πως
 απαιτεί: $\Omega(n^2)$ $\Rightarrow \Theta(n^2)$

ΤΑΞΙΝΟΜΗΣΗ Επιλογής (selection Sort)



```

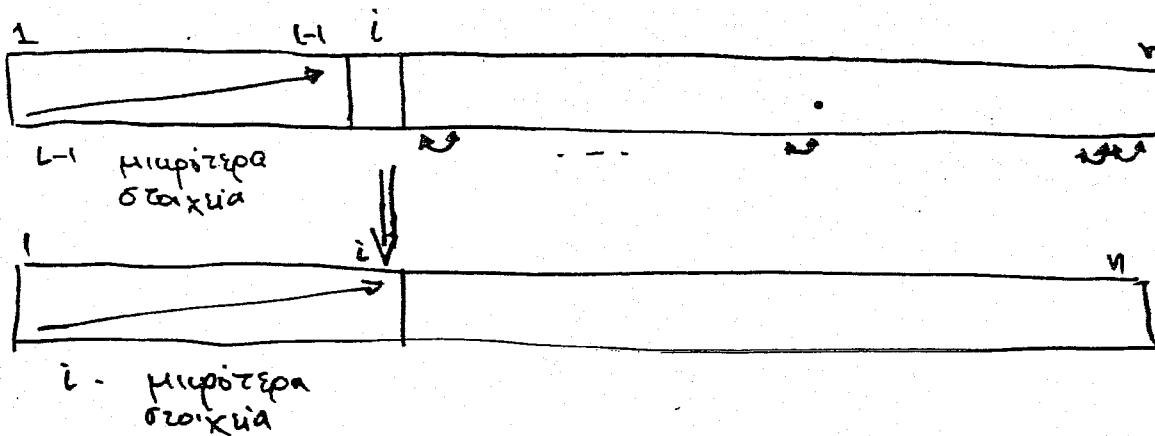
static void selectionSort (Object a[], int left, int right,
                          Comparator c)
{
    for (int i = left; i < right; i++)
    {
        int min = i;
        for (int j = i+1; j <= right; j++)
            if (c.less(a[j], a[min]))
                min = j;
        Object temp = a[i];
        a[i] = a[min];
        a[min] = temp;
    }
}

```

Άναλυση: $(n-1) + (n-2) + \dots + 2 + 1 = \frac{n(n-1)}{2}$ συγκλίσεις

$\Rightarrow O(n^2)$
 Ταξινόμηση των
 n^2 τελετών $O(n^2)$

ΤΑΞΙΝΟΜΗΣΗ ΤΥΕΛΛΙΔΑΣ (bubble sort)



```
static void bubbleSort (Object a[], int left, int right,
                      comparator c)
```

```
{
    Object temp;
    for (int i = left; i < right; i++)
        for (int j = right; j > i; j--)
            if (c.less(a[j], a[j-1]))
                {
                    temp = a[j-1];
                    a[j-1] = a[j];
                    a[j] = temp;
                }
}
```

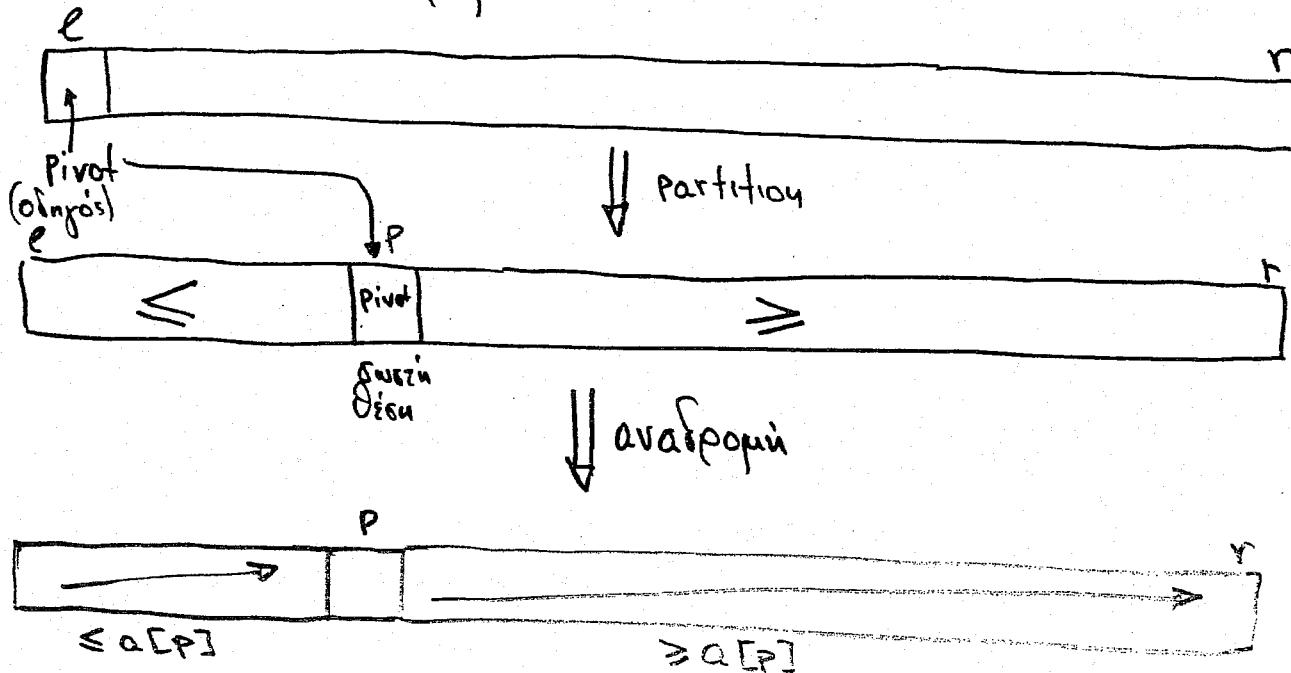
Ανάλυση: Ομοια με selection Sort \Rightarrow

$$\Rightarrow \Theta(n^2)$$

"ΤΑΞΕΙΑ" ΤΑΞΙΔΙΩΝ ΗΕΗ (Quicksort)

Bipa-1 : PARTITION (διαχωρίσιος)

Bipa-2 : αναφορι.



```
static void quickSort (Object a[], int left, int right,
                      Comparator c)
{
```

```
    if (left < right)
    {
```

```
        int p = partition (a, left, right, c);
        quickSort (a, left, p-1, c);
        quickSort (a, p+1, right, c);
    }
```

}

PARTITION:

- Τοποθετεί το πρώτο στοιχείο του διαυγήσας στη σωστή του θέση, έτσι ώστε p
- Όλα τα στοιχεία στις δύο επαναλαμβανόμενες περιοχές να είναι $\leq \text{pivot}$
- Όλα τα στοιχεία στις δύο επαναλαμβανόμενες περιοχές να είναι $\geq \text{pivot}$

```
static int partition (Object a[], int left, int right,
                     comparator c)
```

```
{
    Object pivot = a[left];
    int i = left + 1;
    int j = right;
    while (i <= j)
    {
        while ((i <= right) && c.less(a[i], pivot))
            i++;
        while (c.less(pivot, a[j]))
            j--;
        if (i < j)
        {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
            i++;
            j--;
        }
    }
    a[left] = a[j];
    a[j] = pivot;
    return j;
}
```

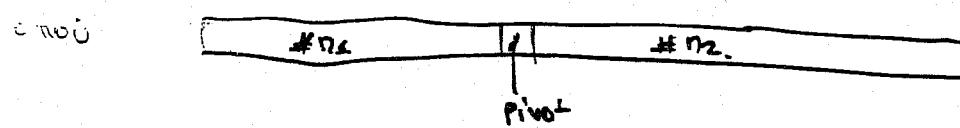
Ανάλυση της partition: O(right-left)

O αρχορίδιον περιλαμβάνει οταν οι διατάξεις $i \geq j$ *συναρτήσουν*. Άλλα με το ποι θα συναρτήσουν, διαχίθου απόστασης right-left ή αν θα μεταβιβάσει μία συγκεκριμένη.

Ανάλυση της QuickSort

O αριθμός των συγκεκριμένων διατάξεων από:

$$Q(n) = \begin{cases} Q(n_1) + Q(n_2) + n & n \geq 2, n_1 + n_2 = \underline{\underline{n-1}} \\ 0 & n=1 \end{cases}$$



Χρήσιμη περίπτωση: $n_1 = 0$ (Πάντας. Συμβαίνει οταν το διόνυσμα είναι ίδιο το γενικό μήκος)

$$\begin{aligned} Q(n) &= Q(0) + Q(n-1) + n = \\ &= 0 + Q(0) + Q(n-2) + n-1 + n = \\ &= 0 + 0 + Q(0) + Q(n-3) + (n-2) + (n-1) + n = \dots \\ &= \dots = 1 + 2 + \dots + (n-1) + n = \frac{(n+1)n}{2} = \\ &= O(n^2) \end{aligned}$$

Μίαν Μεταποίηση.

Υποθέτουμε ότι το πρόβλημα διαιρέται σε δύο μεγαλύτερα υπόπρωτα πρόβληματα α, β, οπόια εκάλυπτε την ίδια πλήθηση και αποτελούν το συνολικό διαχωρισμό.

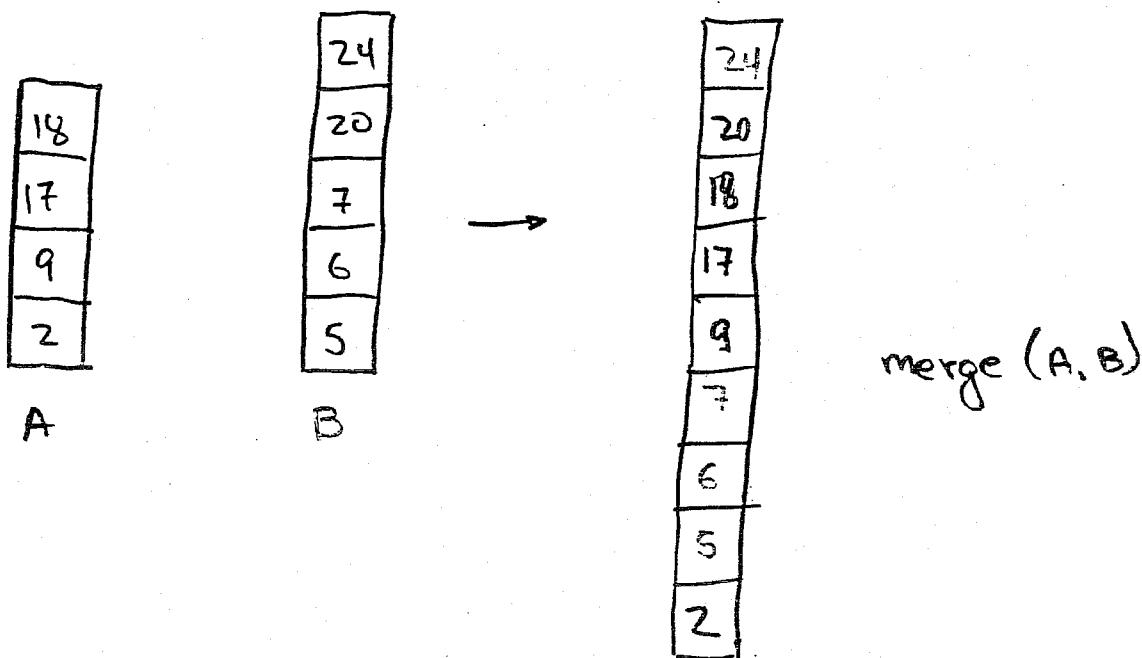
$$\Rightarrow \overline{Q(n)} = O(n \log n)$$

Ti da γίνεται τώρα το πιοτερό νέο πάντα το μεριδιανό στοιχείο;

$$\begin{aligned}
 Q(n) &\leq Q\left(\frac{n}{2}\right) + Q\left(\frac{n}{2}\right) + n = \\
 &= 2Q\left(\frac{n}{2}\right) + n \leq \\
 &\leq 2\left(2Q\left(\frac{n}{2}\right) + \frac{n}{2}\right) + n = 2^2 Q\left(\frac{n}{2^2}\right) + n + n \\
 &\leq 2^2 \left(2Q\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right) + n + n = \\
 &= 2^3 Q\left(\frac{n}{2^3}\right) + n + n + n = \dots \\
 &= \dots = \\
 &= 2^{\log n} Q\left(\frac{n}{2^{\log n}}\right) + \underbrace{n + \dots + n}_{\# \log n} = \\
 &= n Q(1) + n \log n = n(\log n + 1) = \\
 &= O(n \log n)
 \end{aligned}$$

ΤΑΞΙΝΟΜΗΣΗ ΣΥΓΧΩΝΕΥΣΗΣ (merge Sort)

- Συγχώνευση διατάξεων διανομής



$$\text{Εφώ } |A| = n_1$$

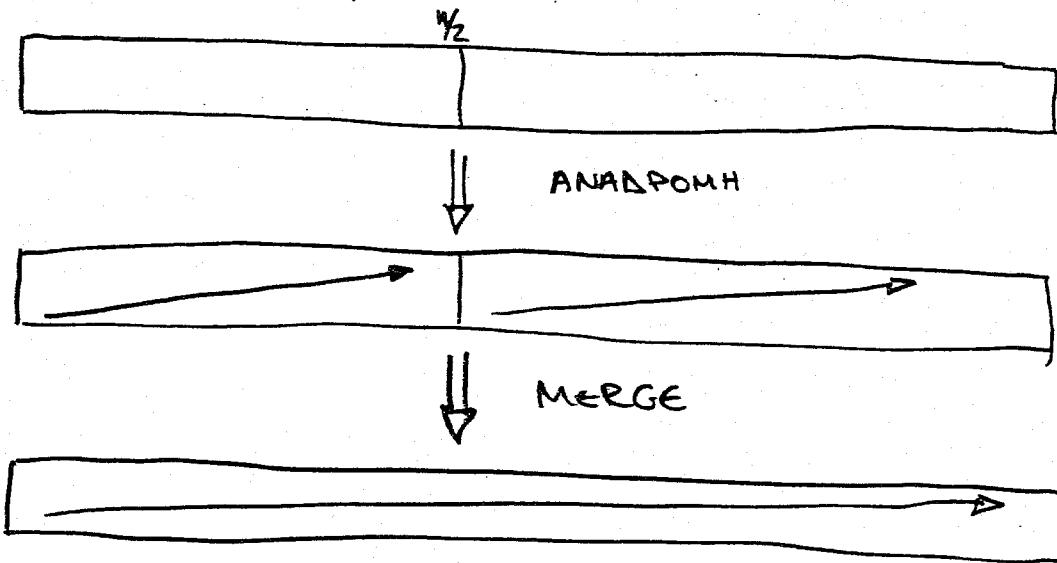
$$|B| = n_2 \quad \text{και} \quad n_1 + n_2 = n$$

Η συγχώνευση μπορεί να γίνεται με $n-1$ συγχώνευσης.

$$n_1 + n_2 - 1 = n - 1 \text{ συγχώνευσης.}$$

$$\Rightarrow = O(n)$$

- Ταξινόμηση μέσω συγχινώσεως



Έργο σε πεδίο

```
static void merge(Object a[], int left,
                  int middle,
                  int right,
                  Comparator c)
```

```
a [left] [middle] [right]
```

```
static void mergeSort(Object a[], int left, int right,
                      Comparator c)
{
    if (right - left >= 1) // τουλάχιστον 2 στοιχεία
        int middle = left + (right - left) / 2;
        mergeSort(a, left, middle, c);
        mergeSort(a, middle + 1, right, c);
        merge(a, left, middle, right, c)
}
```

Ανάδοση: Χαρότυπη περίπτωση

$$T_m(n) = \begin{cases} 2T_m\left(\frac{n}{2}\right) + n-1 & n \geq 2 \\ 0 & n=1 \end{cases}$$

$$\Rightarrow T_m(n) = \dots = n \log n - n + 1 = O(n \log n)$$

ΣΥΝΟΨΗ:

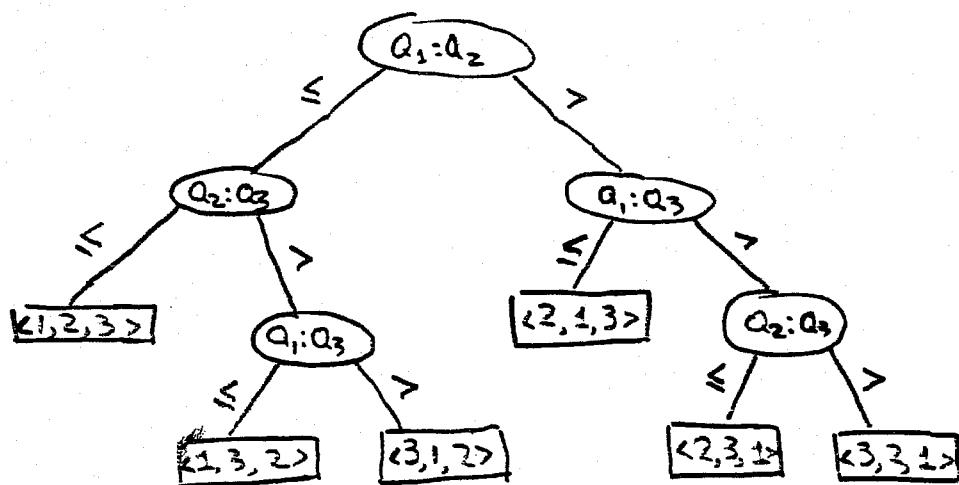
	Worst Case	Average Case
Insertion Sort	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n^2)$	$O(n^2)$
Quick Sort	$O(n^2)$	$O(n \log n)$
MergeSort	$O(n \log n)$	$O(n \log n)$

ΚΑΤΩ ΟΡΟ ΓΙΑ ΣΥΓΚΡΙΤΙΚΟΥΣ ΑΛΓΟΡΙΘΜΟΥΣ ΤΑΞΙΝΟΜΗΣΗΣ

Θεώρημα: Κάθε αλγόριθμος διατάξεως των βασιζεται στις συγκρίσεις χριστεται $\Omega(n \log n)$ χρόνο για να διατάξει η σειρά σειρά χρήσην και μέση περιπτώσει.

η διακριτική σειρά a_1, a_2, \dots, a_n

Διάφορο αποφάσισμα (decision tree)



- Εδιμέρινοι ανόρθωτοι → συγκρίσεις
- Ετετραμερινοί ανόρθωτοι → συγκρίσεις (τέτοιας σειράς)
⇒ ΤΟΥΛΑΧΙΣΤΟΝ $n!$ φύλλα
- Όμοιος διάφορος → αριθμός συγκρίσεων →
→ πολυπλοκότητα
- Διαδικτικός διάφορος

Ευρίσκειν αλγόριθμου ταξινόμησης =

Εύρειν μονοτονίου από pifa ή fulla ουδέ χιούζιν
διεύρυνσης αποφάσις. \Rightarrow

Μεγαλύτερος αριθμός } δύνεται με μεγαλύτερη μήκος μονοτονίας
συχνίσεων }

Το κάτιν άριστο ουδέ υψης του διεύρυνσης συχνίσεων είναι
και κατώ αριστού για όλους την αλγόριθμου διεύρυνση

\therefore Καθε διαδικασία διεύρυνσης έχει το τελευταίο φύλλο

Ενα διεύρυνσης αποφάσις έχει ταυτόχρονη $n!$ φύλλα \Rightarrow

$$n! \leq 2^h \Rightarrow \log(n!) \leq h \quad \text{(*)} \quad \textcircled{*} \quad n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + O(n^{-1})\right)$$

$$\log \left(\frac{n}{e}\right)^n \leq h \quad \Rightarrow$$

$$n \log n - n \log e \leq h \quad \Rightarrow$$

$$h = \Omega(n \log n)$$

Τύποι για μέτρη:

- insertion sort
selection sort
bubble sort } οξ 75-80
- QuickSort οξ 82-89 Η αναζήτηση σε κάθε παραίτηση δημιουργεί ένα νέο σύνολο
- MergeSort οξ 97-101
- Κάτια ωρίο οξ 102-105 (μόνη περιπτώση στην Ελλάς)

Ασυντάξις: 4.1 - 4.3, 4.5

ΑΛΓΟΡΙΘΜΟΙ ΚΑΤΑΝΟΜΗΣ

- Τα λέξεις συνομίζονται "λέξεις"
- Οι λέξεις αρχικοποιούνται από τη ψηφία ευός "αλφαβήτου"
- Το δύνατον από το οποίο προέρχονται οι πρώτες λέξεις εχει περιορισμένο μεγέθος

ΠΧ Τεχναρχιφες λέξεις με κεφαλαια ελληνικά γράμματα:

$$\text{Αλφαριθμός } \Sigma = \{ \text{Α, Β, ..., Ψ, Ω} \}$$

$$|\Sigma| = 24$$

Σύρπται σεγραφήψιμων λέξεων : μεγέθος 24^4

ΤΑΞΙΝΟΜΗΣΗ ΚΑΔΩΝ (bucket Sort)

->- Δοχείον (bin Sort)

- O. Διέξις αποτελείται από ένα μονο γράμμα του αλφαριτου

```

class alphabet {
    protected String sigma;
    protected int radix;

    public alphabet() {
        sigma = "0123456789";
        radix = sigma.length();
    }

    public alphabet(String s) {
        sigma = s;
        radix = sigma.length();
    }

    // επιστρέφει την "τάξη" του < στο sigma
    public int getDigit(char c) {
        return sigma.indexOf(c);
    }
}

```

// επιστρέφει την ταξή σειρά σήμα του χαρακτή
// στη θέση position του String s

```
public int getDigit (String s, int position) {  
    return sigma.indexOf(s.charAt(position));  
}
```

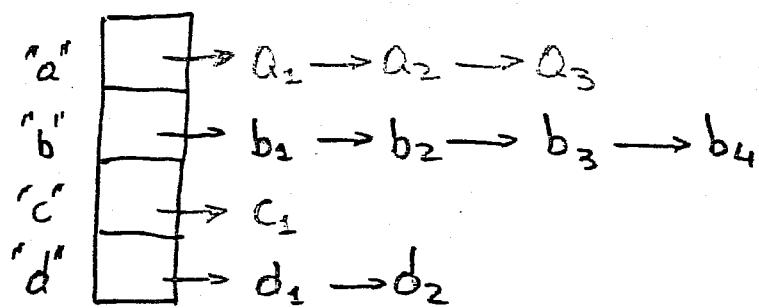
```
public char getLetter (int i) {  
    return sigma.charAt(i);  
}
```

```
public int getRadix () {  
    return radix;  
}
```

3. Alphabet

ΤΑΞΙΝΟΜΗΣΗ ΚΑΔΟΥ (Bucket Sort)

$S \rightarrow b_1 \rightarrow d_1 \rightarrow a_1 \rightarrow b_2 \rightarrow b_3 \rightarrow c_1 \rightarrow a_2 \rightarrow b_4 \rightarrow d_2 \rightarrow a_3$



$\Rightarrow S \rightarrow a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow b_1 \rightarrow b_2 \rightarrow b_3 \rightarrow b_4 \rightarrow c_1 \rightarrow d_1 \rightarrow d_2$

- Η μέθοδος είναι σταθερή (stable)

Άνθρωποι: $O(n + |\Sigma|)$

ΤΑΞΙΝΟΜΗΣΗ ΒΑΣΕΩΣ (RADIX SORT)

- ΛΙΓΟΤΕΡΟΥ ΣΗΜΑΝΤΙΚΟΥ ΨΗΦΙΟΥ

Εισόδος • n θέσεις

• l γράμματα / θέση

• Η ίδια S από n θέσεις $w_1 w_2 \dots w_n$

Έσοδοι θέσεων: $x_0 x_{e-1} \dots x_2 x_1$

ΑΛΓΟΡΙΘΜΟΣ

For $i = 1 \dots l$ do

- Ταξινομηθεί τη θέση S με εναν σύντομό
alg. με βαση το i -ος το γιγάντερο δικαντίου
συνοχών.

Ανάδυση: $O(l \cdot (n + |\Sigma|))$

είναι χρησιμοποιείται και bucket Sort.

Αναδύση σφύζωντας: Με επαργία μετά το μήνα
των θέσεων

Βαση: $i=1$ (μονοψήφιες θέσεις)

Προβαστή πότε του οριζεται bucket sort
ταξινομηση σώστα (και σταθερά)

Επομένων γιαδρου : Εσών οι σ αλγορίθμοι ταξινομεί
σωρά (ναι σταθερά) όταν τις γίνεται
μήνυσης $\ell \pm$ γραμμάτων

Επομένων βίβε

\Rightarrow Ταξινομεί σωρά (ναι σταθερά) τις γίνεται
μη ℓ γραμμάτων

Εσών 2 ωχαις γέζεις συνη ℓ -οσιν επανάληψη

$a_0 a_{e_1} \dots a_1 \quad b_0 b_{e_1} \dots b_1$

• $a_\ell = b_\ell$

\Rightarrow Η σταθερότητα του αλγ. εξαρτάται οι
δωδεκα αδιάστατη και με τον ίδιον τρόπο
τους.

• $a_\ell \neq b_\ell \Rightarrow$ Η σταθερότητα του αλγ. εξαρτάται μόνο
οι τρόποι που έχει γράψει.

□

ΒΑΣΙΚΕΣ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

* ΠΛΙΝΑΚΕΣ (ARRAYS)

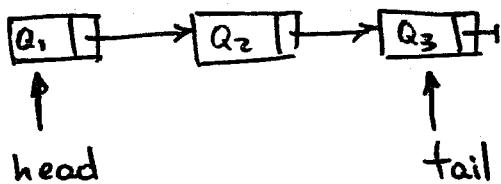
- Συλλογή αντικεμένων "ίδιου" τύπου
- μεγάλος αριθμός συστημάτων
- `Java.util.ArrayList`

* ΓΡΑΜΜΙΚΕΣ ΛΙΣΤΕΣ

* αναλογίες που μπορούν να γροποιούνται
οποιαδήποτε θέση

- | | |
|----------------------------------|--|
| • <code>first()</code> | Επιστρέφει δίνη τρόis zou πρώτο κόμβο |
| • <code>head()</code> | |
| • <code>last()</code> | Επιστρέφει δίνη πρόis του τελευταίου κόμβου |
| • <code>tail()</code> | |
| • <code>insertAfter(p,e)</code> | Ενθέτει <u>νέο</u> κόμβο με στοιχείο <u>e</u> ακ τρόis του κόμβου <u>p</u> |
| • <code>insertBefore(p,e)</code> | |
| • <code>remove(p)</code> | Αφαιρεί εναν κόμβο οι σχέση με του κόμβου <u>p</u> |

And's Nices (singly linked list)



```

class SNode {
    private SNode next;
    private Object element;

    SNode(SNode nodeNext, Object nodeElement) {
        next = nodeNext;
        element = nodeElement;
    }

    public Object getObject() { return element; }
    public SNode getNext() { return next; }

    public void setElement(Object newElement) {
        element = newElement;
    }

    public void setNext(SNode newNext) {
        next = newNext;
    }
}

```

```

public class SinglyNodeList
{
    protected int noOfElements;
    protected SNode head, tail;

    SinglyNodeList()
    {
        noOfElements = 0;
        head = null;
        tail = null;
    }

    public int size() { // O(1)
        return noOfElements;
    }

    public boolean isEmpty() { // O(1)
        return (noOfElements == 0);
    }

    public boolean isFirst(SNode v) { // O(1)
        return (v == head);
    }

    public boolean isLast(SNode v) { // O(1)
        return (v == tail);
    }
}

```

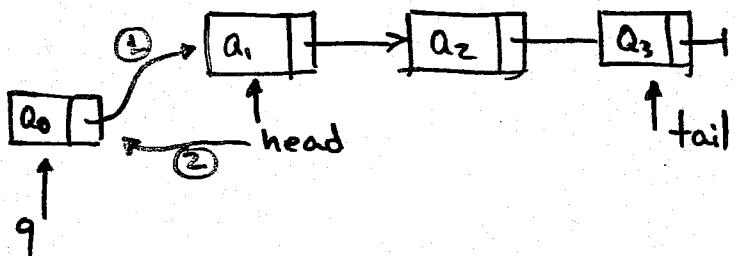
~~public SNode first() {
 if (isEmpty())
 System.out.println ("Hárra sval áðra...");
 return head;
}~~

public SNode first() { return head; } //O(1)

~~public SNode last() {
 if (isEmpty())
 System.out.println ("Hárra sval áðra...");
 return tail;
}~~

public SNode last() { return tail; } //O(1)

- Εύθετη σεναρίο όχι των διόρας



```

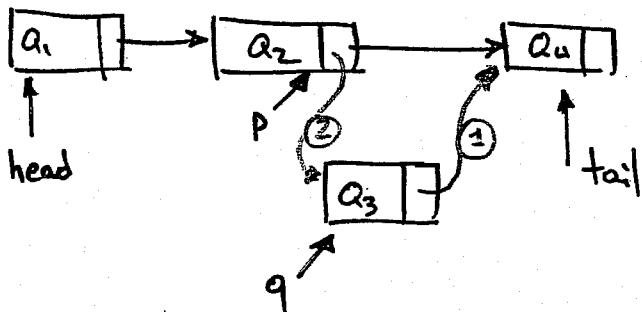
public SNode insertFirst (Object element) {
    SNode q = new SNode (head, element);
    head = q;
    if (tail == null)
        tail = q;
    noOfElements++;
}
  
```

Ανάλυση: $O(1)$

- Εύθετη σε τις διόρας των διόρας

... με όμοιο τρόπο

Ένδειν μέτρα από την συσκέψιμο. Συλλεκτικός και δημόσιος
και, P



```
public SNode insertAfter(SNode p, Object element)
{
```

~~If (!isEmpty()) {~~

~~System.out.println ("Δια στηθεται η εισαγωγή στην
λίστα");~~

~~? . return null;~~

noOfElements++;

SNode q = new SNode (p.getNext(), element);

if (p.getNext() == null)

tail = q;

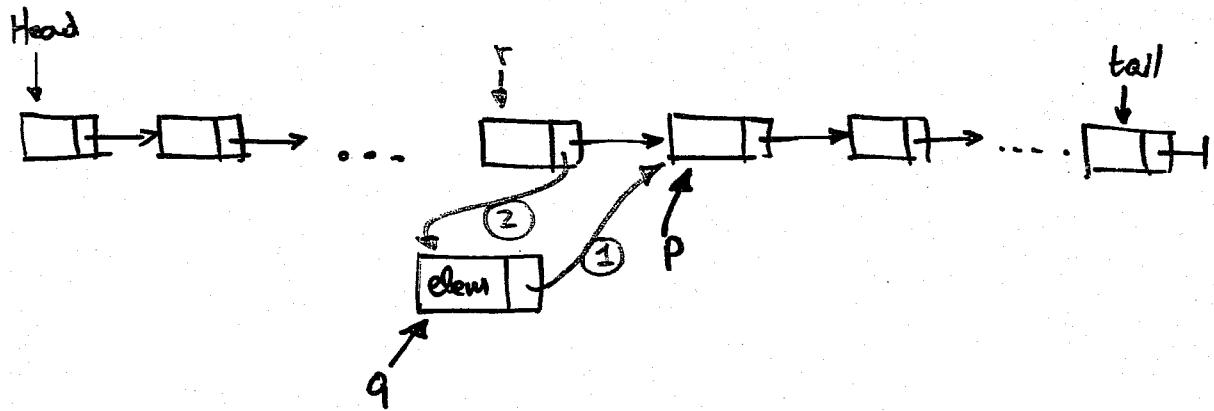
p.setNext(q);

return q;

}

Ανάλυση: O(1)

Ένδρον πριν από το στοιχείο P



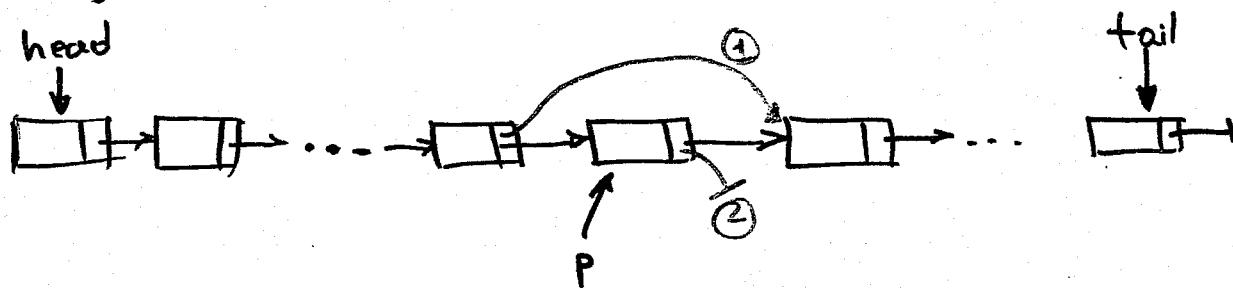
Πρόβλημα: Τις βρίσκου το στοιχείο πριν το P είναι μως να δώσω την ομαλήν την επόμενη πέμπτη next?
[Επιτροφή ②].

Απάντηση: Διασχίζω τη λίστα!

→ Άναλυση: $O(n)$!

⇒ Για $O(1)$ υλοποίηση χρησιμοποιήσε διπλα-συνδεδμένη λίστα

- Διαγραφή στοιχείων (P)



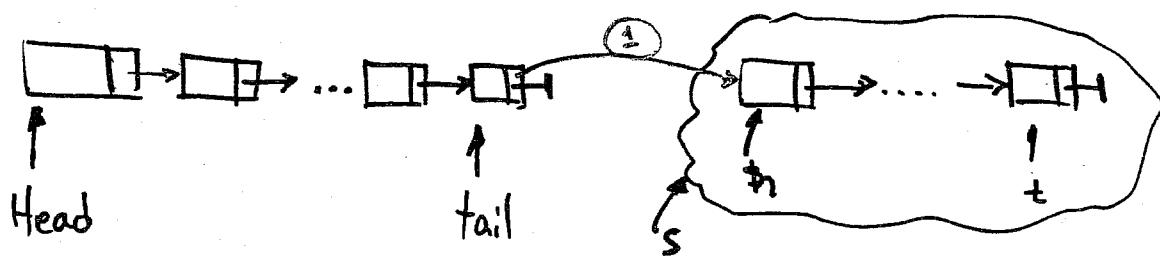
Πρόβλημα: Ο ευποιησμός του στοιχείου που προηγείται του P απαιτεί σάρωση των λιστών
 $\Rightarrow O(n)$

Λύση: ... Διπλή συνδεσμένη λίστα.

Προβλήματα με την υλοποίηση:

- Οι μέθοδοι `setNext()` ΔΕΝ πρέπει να είναι `public`
- Δίνει την δυνατότητα να σπάσει η λίστα
- Η κλάση `SNode` πρέπει να δηλωθεί `public`
 [οπου υλοποίηση του βιβλίου]

- Συνένωση με άλλη λίστα



Προσοχή!: πρέπει να "αδνάσουμε" την s

Fazit ;

```
public void setEmpty()
```

```
    head = null;
```

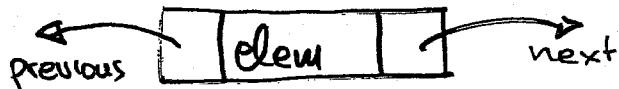
```
    tail = null;
```

```
    noOfElements = 0;
```

```
}
```

```
} // Singly NodeList
```

Doubly Linked List



```
class DNode {
```

```
    private DNode next, previous;
```

```
    private Object element;
```

```
DNode(DNode nodePrev, DNode nodeNext, Object nodeElement)
```

```
{   previous = nodePrev;
```

```
    next = nodeNext;
```

```
    element = nodeElement;
```

```
}
```

```
public Object getElement() { return element; }
```

~~public DNode getPrevious() { return previous; }~~

~~public DNode getNext() { return next; }~~

```
public void setElement(Object newElement) {
```

```
    element = newElement;
```

```
}
```

~~X public void setPrev(DNode newPrev) {~~
~~previous = newPrevious;~~
~~}~~

~~X public void setNext(DNode newNext) {~~
~~next = newNext;~~
~~}~~

~~}~~ //DNode

public class DoublyNodeList
{

 protected int noOfElements;

 protected DNode head, tail;

 NodeList() {

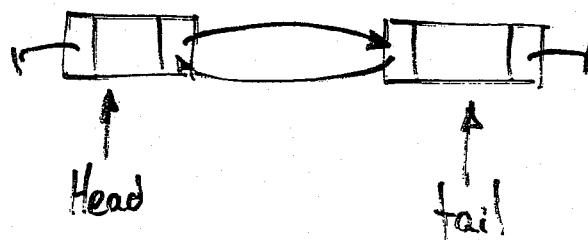
 noOfElements = 0;

 head = new DNode(null, null, null);

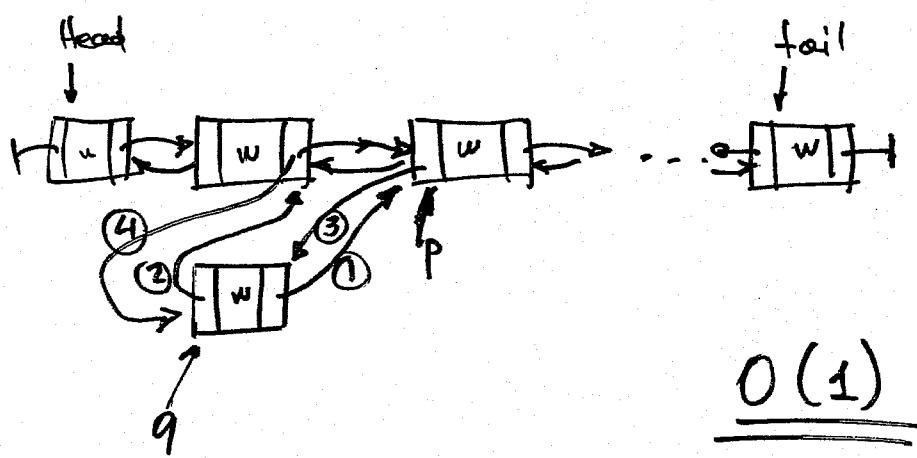
 tail = new DNode(head, null, null);

 head.setNext(tail);

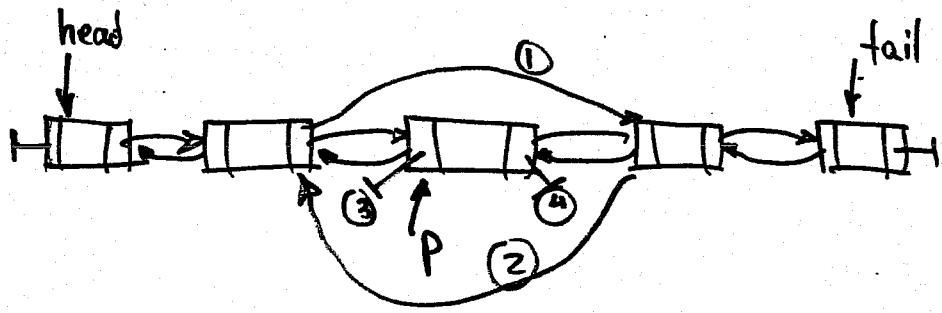
}



- Ένδειν πίριν από στοιχείο P



- Διαγραφή στοιχείου P



Δομής Δεδομένων:

- Λιστά
 - ταξινομημένο] διότι γράφουμε
 - μη ταξινομημένο] διότι γράφουμε
- Λίστες
 - Ταξινομημένες] διότι γράφουμε
 - μη ταξινομημένες] διότι γράφουμε
- απλά συνδεδεμένες διπλά συνδεδεμένες] ΔΟΜΗ

ΑΦΗΡΗΜΕΝΟΣ ΤΥΠΟΣ ΔΕΔΟΜΕΝΩΝ

- ΛΙΣΤΑ
 - new
 - insert / insertFront / insertBack
 - delete / deleteBefore / deleteAfter
 - isEmpty
 - size
 - ...

Να είχασει και κάποια ArrayList από την Java.

ΟΥΡΕΣ (Queues)

- ΣΤΟΙΒΑ [Stack]

→ LIFO [Last-In-First-Out]

AAT: STACK

- is Empty()
- push(x)
- pop()
- top()

- Η προσβαση είναι δυνατή μόνο στο κορυφαίο στοιχείο της στοίβας

- Τα στοιχεία εμφανίζονται σε δορυφ με σερά αριστερά από τη σερά εισαγωγής

- Εφαρμογές - Παραδείγματα

- + Κέρμασις - στοίβα πίστων

- * Τίτλου "Back" του browser

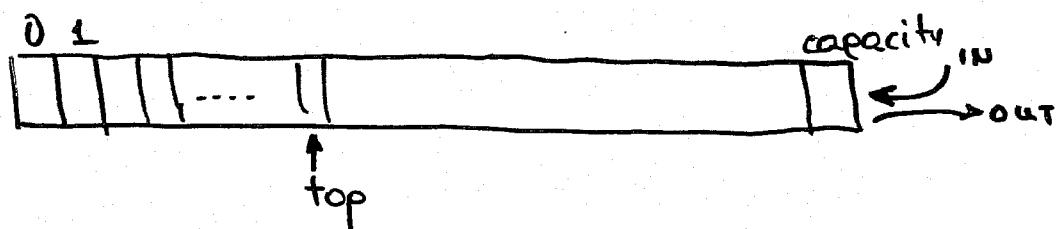
- * Αντιγραφη σεριαλισμού βορών

- * Κλίκοι μεσάνια - Αναζήτηση (activation records)

* Βασικόν για Τίτλων

* Εμπλέκου προδος : isFull()

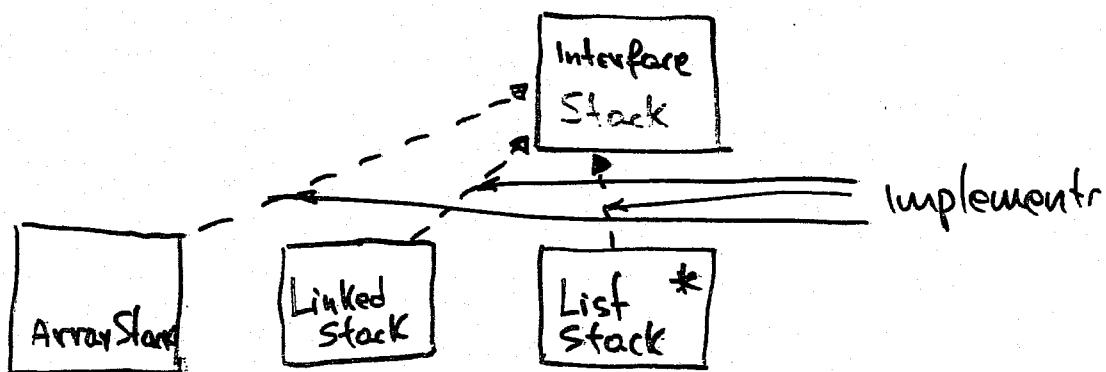
* Μεγάλη χρησιμότητα



interface Stack

```
{
    boolean isEmpty();
    void push (Object x);
    Object pop ();
    Object top ();
    boolean isFull();
    int size();
}
```

}



```
public class ArrayStack implements Stack
{ public static final int defaultCapacity = 1000;
  private int capacity;
  private Object stack[];
  private int top = -1;
```

```
public ArrayStack(int cap)
{ capacity = cap;
  stack = new Object[cap];
}
```

```
public ArrayStack()
{
  this(defaultCapacity);
}
```

```
public int size()
{
  return top + 1;
}
```

```
public boolean isEmpty()
{
  return (top < 0);
}
```

...ArrayStack conclusion

```

public void push ( Object obj )
{
    if ( size () == capacity )
    {
        System.out.println (" Υπερχειδικη στοκα ");
        return ;
    }
    top++;
    stack [ top ] = obj;
}

```

```

public Object top ()
{
    if ( isEmpty () )
    {
        System.out.println (" Αδυνα στοκα ");
        return ;
    }
    return stack [ top ];
}

```

... Array Stack ouivixio

```
public Object pop()
```

```
{  
    Object elem;  
    if (isEmpty())
```

```
{  
    System.out.println("Adua oziba");  
    return;
```

```
elem = stack[top]
```

```
stack[top] = null;
```

```
return elem;
```

```
}
```

Dostupnosti za

garbage collection

```
public boolean isFull()
```

```
{
```

```
    return (top == capacity);
```

```
}
```

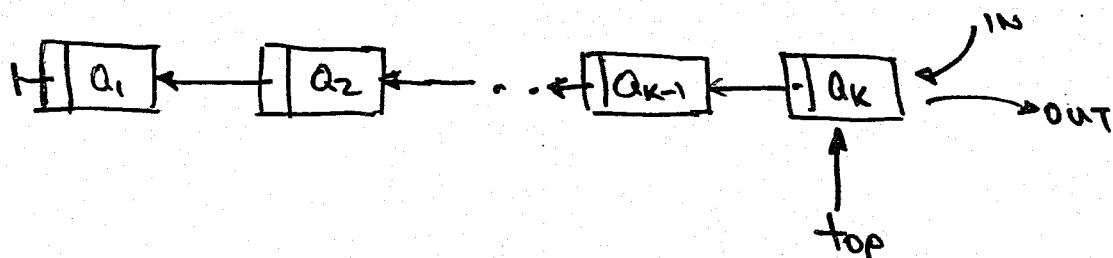
```
} // ArrayStack
```

ANALYΣΗ:

<ul style="list-style-type: none"> • size() • isEmpty • isFull • push 	$O(1)$	<ul style="list-style-type: none"> • pop() • top()
---	--------	--

$ArrayStack(cap) = O(cap)$

* Ηλεκτρική γραμμή πίστας



- χρησιμοποιούμε την μέθοδο SNode.

public class LinkedStack implements Stack

```
{
    private SNode top;
    private int size;
```

```
public LinkedStack()
{
```

```
    top = null;
    size = 0;
}
```

```
public int size()
```

```
{
    return size;
}
```

... LinkedStack συνέχεια

```
public boolean isEmpty()
{
    return (size == 0);
}
```

```
public boolean isFull()
{
    return false;
}
```

```
public void push(Object elem)
{
    SNode x = new SNode(top, elem);
    top = x;
    size++;
}
```

... Linked Stack swixya

```

public Object pop()
{
    SNode = oldTop;
    if (isEmpty())
    {
        System.out.println("n oraiba sivu abua....");
        return null;
    }
    oldTop = top;
    top = top.getNext();
    oldTop.setNext(null);
    size--;
    return oldTop.getElement();
}

```

```

public Object pop()
{
    if (isEmpty())
    {
        System.out.println("n oraiba sivu abua");
        return null;
    }
    return top.getElement();
}

```

} // LinkedStack

Avdouon: O(1) για κάθε λειτουργία.

* ListStack [Δεν είναι στο βιβλίο !!!]

LIST ADT:

first()	Σημειώνεται ότι ο πρώτος ιδιότητας είναι η πρώτη στοιχείο
last()	>> >> απλωτισμός >
insertAfter(p, e)	εισαγάγεται το e μετά τον p
remove(p)	διαγράφεται το p
size()	μήκος
isEmpty()	

public class ListStack implements Stack

{

private List stack;

public ListStack()

{

stack = new DoublyNodeList();

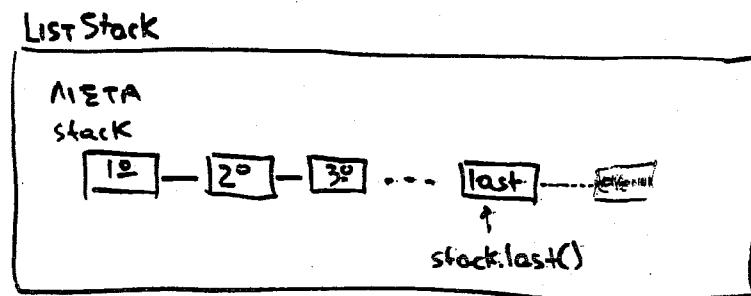
}

```
public int size()
{
    return stack.size();
}
```

```
public boolean isEmpty()
{
    return stack.isEmpty();
}
```

```
public boolean isFull()
{
    return false;
}
```

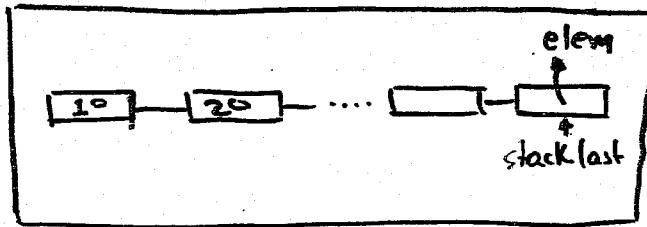
```
public void push(Object elem)
{
    DNode q = insertAfter(stack.last(), elem);
}
```



```
public Object pop()
{
```

DNode q;

Object elem;



q = stack.last();

stack.remove(q);

elem = q.getElement();

return elem;

}

```
public Object top()
{
```

DNode q;

q = stack.last();

return q.getElement();

}

} // List Stack

Avaliação: O(1) para cada operação.

- * Não "apaga" ainda as ArrayStack, LinkedStack

- * Não "enquadradas" ainda

Διπλούπα [double ended Queue] ~~dequeue~~

ADT:

interface ~~Dequeue~~

{

void inject (Object x)
 Object eject();
 void push (Object x);
 Object pop();
 Object top();
 Object last();

}

Evaluation's operations

interface Double Ended Queue

{

void insertFront (Object x)
 void insertBack (Object x)
 Object removeFront();
 Object removeBack();
 Object front();
 Object back();
 boolean isEmpty();
 boolean isFull();
 int size();

}

Υπόπτωμα:

Άριθμος: Διπλά - συνδεδεμένη Λίστα

Έργον: List (Συλ. διπλα - συνδεδεμένη Λίστα...)

ΑΝΑΛΥΣΗ: $O(1)$ χρόνο για κάθε λειτουργία.

ΑΞΙΗΣΗ: Να ωλεται σε Doubly Ended Queue
με βασι την List

ΣΤΑΤΙΚΑ ΔΕΝΔΡΑ

ΑΤΔ: Tree:

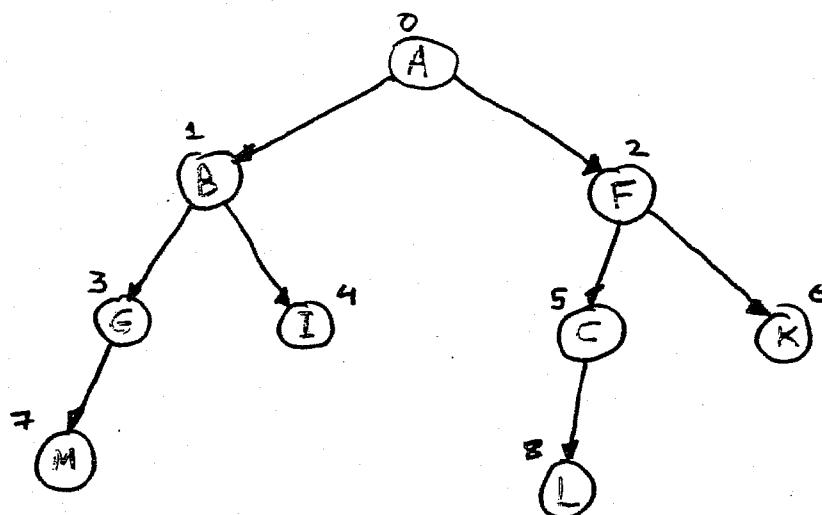
- element(v)
 - father(v)
 - children(v)
- } v: κόρης του δένδρου
- Επιτρέπουν επισυστήματα του δένδρου

- setElement(v,e)
 - setSon(v,p,i)
 - setFather(v,p)
- } → Αλλαγή σε μορφή του δένδρου
- Θέτει τὸν υòρθο p ως τὸν i-οντο πατέρα του
- _____ p — πατέρα του v

Στατικό δένδρο: Ένα επιχρεπτεί την αρχή της
"μορφής" του δένδρου



ΥΛΟΠΗΝΗ ΜΕ ΤΙΝΑΚΕΣ
(για διάδικτη διέφο)



ΟΓΓΗ	ΣΤΟΙΧΕΙΑ	ΑΡΙΣΤΕΡΟΣ ΥΙΟΣ	ΔΕΞΙΟΣ ΥΙΟΣ	ΠΑΤΕΡΑΣ
0	A	1	2	null
1	B	3	4	0
2	F	5	6	0
3	G	7	null	1
4	I	null	null	1
5	C	8	null	2
6	K	null	null	2
7	M	null	null	3
8	L	null	null	5

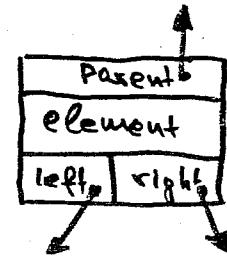
σε External and Internal links

ΥΛΟΠΟΙΗΣΗ ΜΕ ΔΙΚΤΕΣ (διαδικτική διεύθυνση)

```
public class BTNode
```

```
    private Object element;
```

```
    private BTNode left, right, parent;
```



```
public BTNode() {}
```

```
public BTNode(Object o, BTNode p, BTNode l, BTNode r)
```

```
{ element = o;
```

```
    left = l;
```

```
    right = r;
```

```
    parent = p;
```

```
}
```

```
public Object element() { return element; }
```

```
public BTNode getLeft() { return left; }
```

```
public BTNode getRight() { return right; }
```

```
public BTNode getParent() { return parent; }
```

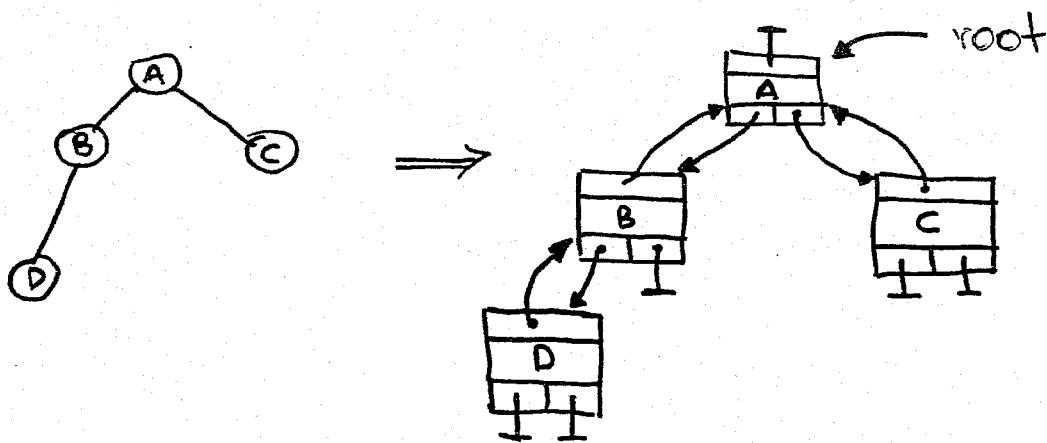
```
public void setElement(Object o) { element = o; }
```

```
public void setLeft(BTNode l) { left = l; }
```

```
public void setRight(BTNode r) { right = r; }
```

```
public void setParent(BTNode p) { parent = p; }
```

? BTNode



```
public class LinkedBinaryTree
```

```
{
    private BTNode root;
    private int size;
```

```
public LinkedBinaryTree()
```

```
{
    root = null;
    size = 0;
}
```

~~public LinkedBinaryTree() (BTNode el)~~

~~root = el~~
size = 1;
???

Elvar 1?

```

public LinkedBinaryTree(Object ob)
{
    root = new BTNode(ob, null, null, null);
    size = 1;
}

```

```
public int size() { return size; }
```

X ~~public void setSize(int s) { size = s; }~~

```
public BTNode root() { return root; }
```

X ~~public void setRoot(BTNode v) { root = v; }~~

```
public boolean isRoot(BTNode v) { return (v == root); }
```

~~public boolean isLeft(BTNode v)~~

~~if (v == root)~~
~~{ System.out.println("Eror! ozu pida...");~~
~~return false;~~
~~}~~

~~return (v == v.getParent().getLeft());~~

1

```
public boolean isRight(BTNode v)
{
    return (v == v.getParent().getRight());
}
```

```
public BTNode getSibling(BTNode v)
{
    if ((v == null) || isRoot(v))
    {
        System.out.println("Δεν υπάρχει αδέλφος");
        return null;
    }
    if (isRight(v))
        return v.getParent().getLeft();
    else
        return v.getParent().getRight();
}
```

3.6 Linked Binary Trees

• K-adjacé déjpo

```
public class TNode
```

```
private Object element;
private TNode parent;
private TNode sous[];
private int noOfSous;
```

```
public TNode() {}
```

```
public TNode(Object obj, TNode p, int sousNumber)
```

```
{ element = obj;
```

```
parent = p;
```

```
sous = new TNode[sousNumber];
```

```
noOfSous = sousNumber;
```

```
}
```

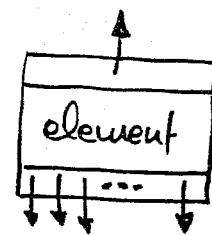
```
public Object element() { return element; }
```

```
public void setElement(Object obj)
```

```
{
```

```
element = obj;
```

```
}
```



```

public TNode getSon(int i)
{
    if (i > noOfSous - 1)
        System.out.println("Avu uttafel rridos x'ios....");
    return null;
}
return sous[i];
}

```

Ambi Tásser!!

```

public void setSon (int i, TNode v)
{
    sous[i] = v;
}

```

```

public TNode getParent () { return parent; }

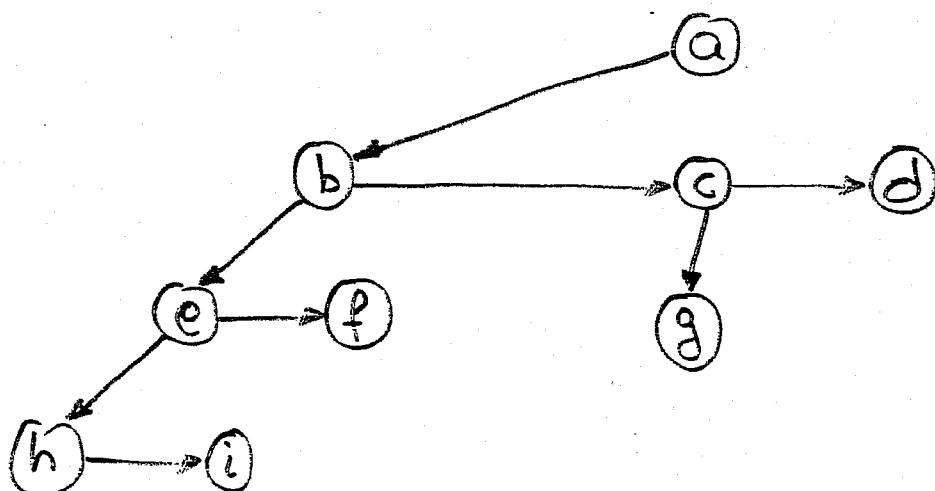
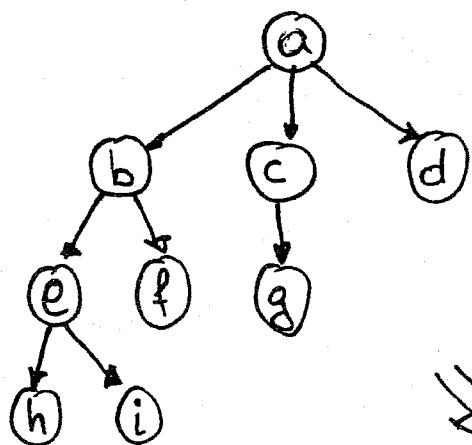
public void setParent (TNode v)
{
    parent = v;
}

```

1.1.1.1.1.1.

Αυτοπαράσταση πρώτων παιδίων - Δίξιο σελίφου

[First child - left sibling]



- Η υλοποίηση μπορεί να γίνεται με κανονικότητα του καθές BTNode
- Είναι δύσκολη δράση
 - Αναλογική "εργασία"

ΔΙΕΛΕΥΣΕΙΣ ΔΕΝΔΡΩΝ

- Διέλευση διαπεργού
tree traversal
- } \cong μια διαδικασία επίσκεψης/
επιζήφυσης όλων των κοριτών
ενός δένδρου με ανατυπωτικό
χρόνο

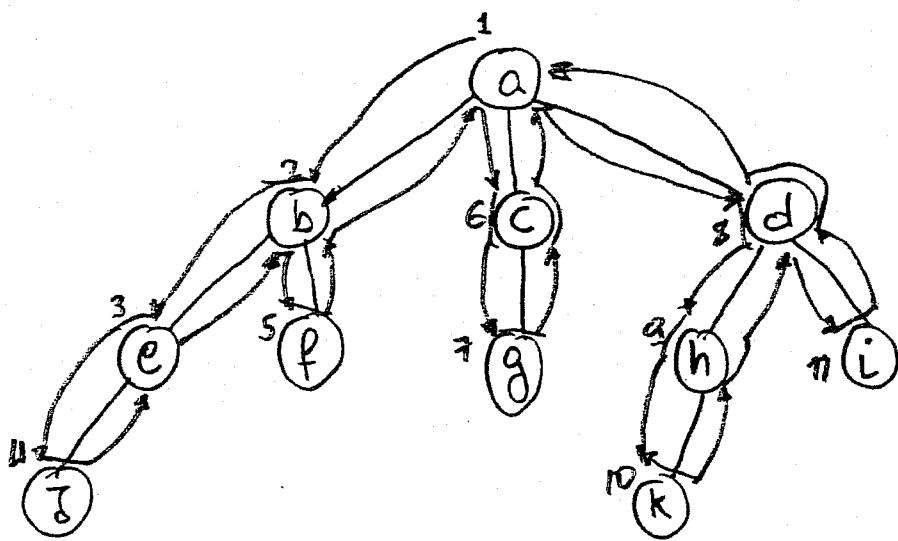
Pre-order (Προδιάταξη)

- Ενας κορμός διεμφέρεται πριν γίνεται τα παιδιά του

Algorithm preOrder(TNode v)

1. if ($v == \text{null}$) return;
2. visit(v) // επιζήφυση του v
3. for each παιδί x του κοριτού
 preOrder(x) // αναδρομή

Αυτή: $f(n)$



PreOrder aufzählen: a - b - e - j - f - c - g - d - h - k - i

- postOrder - (Ηετα-διάδοση)

- Κάθι αρμόσεις επίτρεψε σήμερα την παρίστανση του

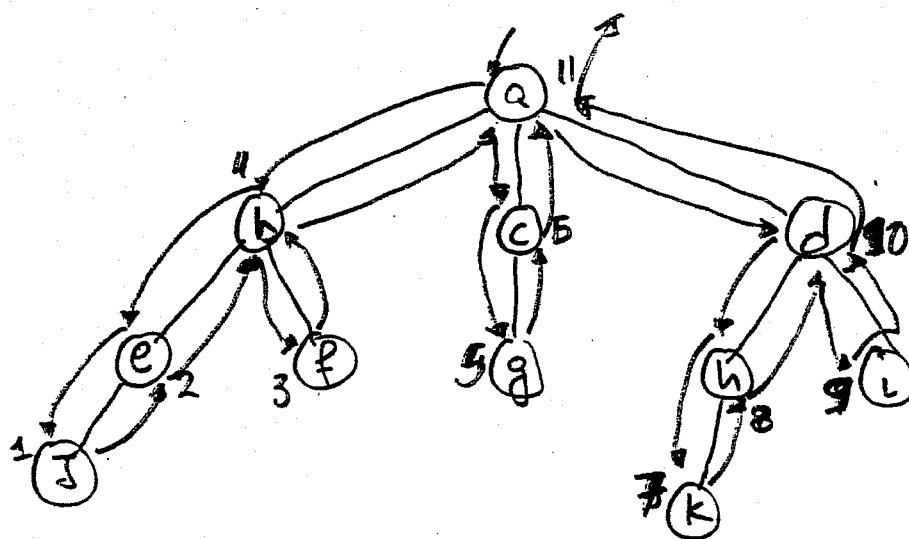
Algorithm postOrder (TNode v)

```

1 if (v == null) return
2 foreach παιδί x του v do
3   postOrder(x)           // ανεργοποίηση
4   visit(v);             // επεξεργασία του v

```

Anάλυση: $O(n)$



post-order αναζήτηση: J-E-F-B-G-C-K-H-I-D-Q

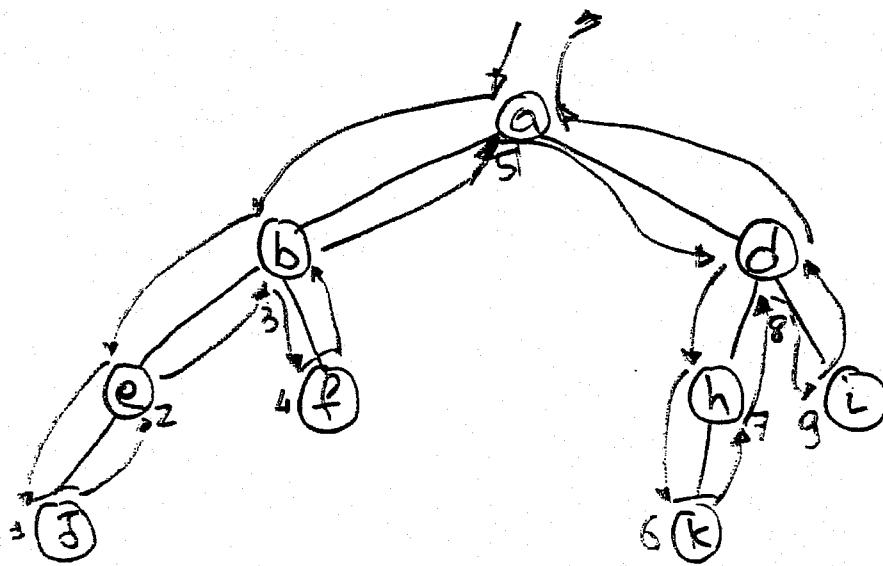
• In-order (εσωτερική) διάταξη.

- Εφαρμόζεται μόνο σε δυαδικά δένδρα
- Ενώ ποτέ δεν επεξεργάζεται του αριστερού παιδιού και προηγείται του δεξιού

Algorithm inOrder (TNode v):

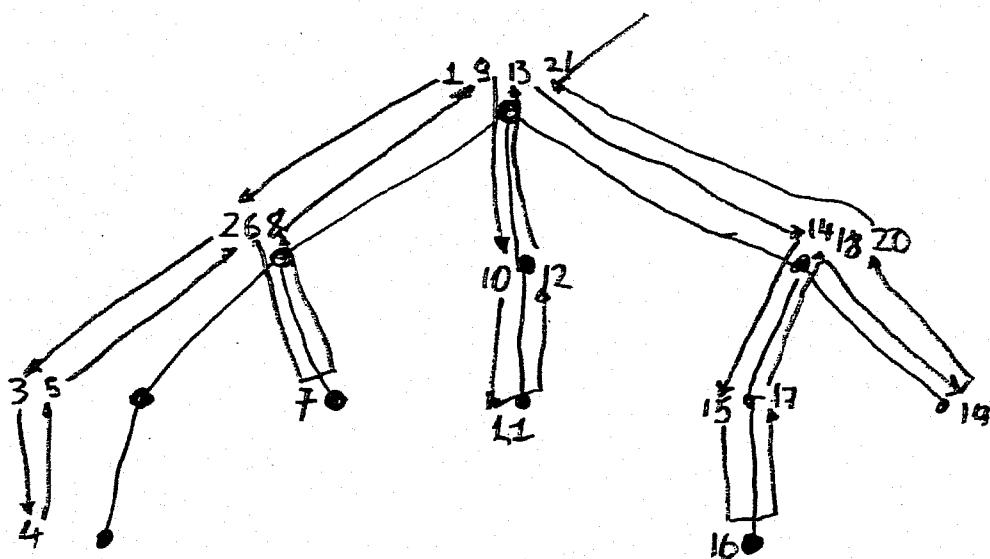
- 1 if ($v == \text{null}$) return
- 2 Else If $v.l$ και $v.r$ το αριστερό και δεξιό παιδί, αυτίστριψτε
- 3 inOrder($v.l$)
- 4 visit(v)
- 5 inOrder($v.r$)

Τιμήσουν: $O(n)$

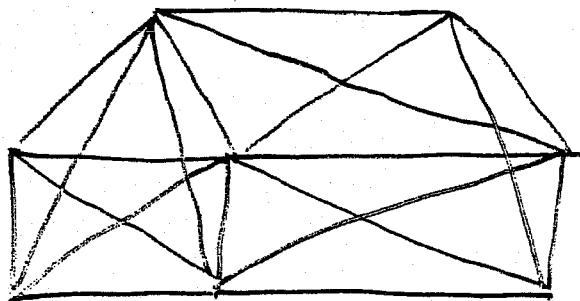


In-order traversal: $j - e - b - f - c - g - h - k - l - i$

Διάταξη ωρία Euler:



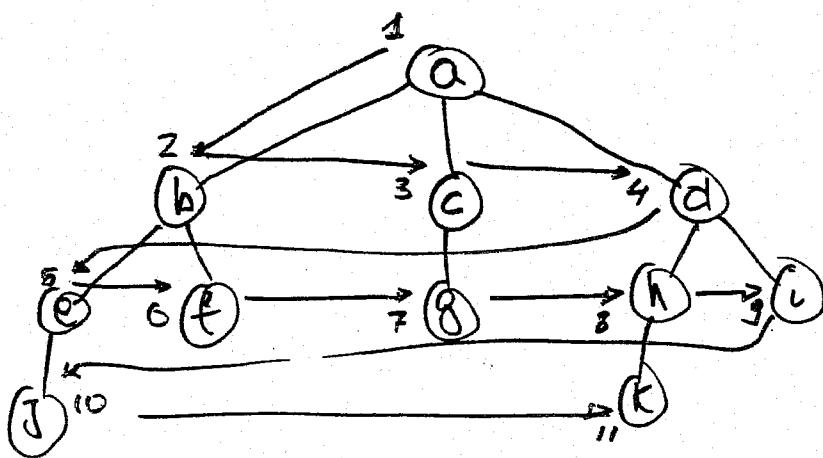
- Μονοπότοι - κύκλος euler



- Μπορεί να συμβαστεί χωρίς να απαισιγορεί το μονοπότο;

Διατίρση ή πάτα επίπεδο

- Επισυντέτω τους κόμβους ενώ δημιουργούν
ανα στάγδα, και από αριστερά της
διέτα.



Algorithm LevelOrder (TNode v)

```

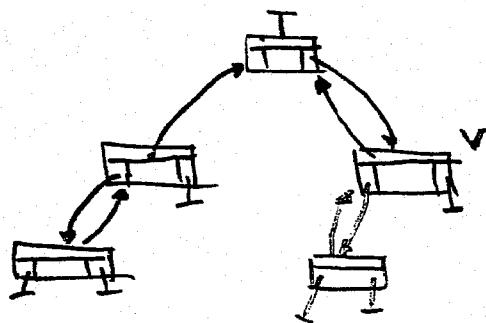
1. FIFO f = new FIFO();
2. TNode w;
3. f.enqueue(v);
4. while (!f.isEmpty()) do
5.   w = (TNode) f.dequeue();
6.   visit(w);
7.   For each παιδί x του w
8.     f.enqueue(x);
  
```

DYNAMIKA DENARA

- addLeaf (v , kindOfSon)

$\begin{matrix} L & & R \end{matrix}$

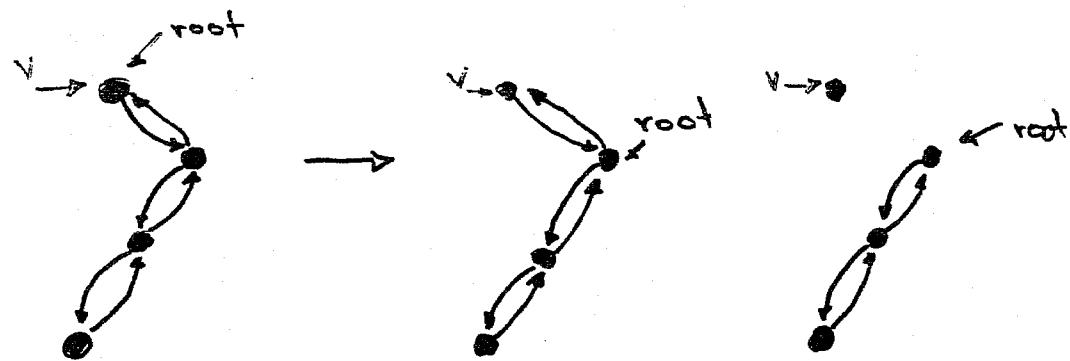
- deleteNode (v)



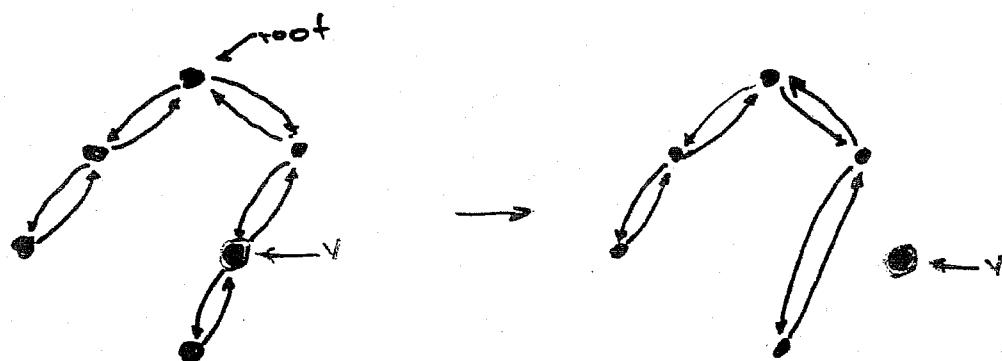
Требование остаточного условия

```
public void addLeaf (BTNode v, boolean kindOfSon) // Left = true  
{ // Right = false  
    if (kindOfSon == Left)  
        v.setLeft (new BTNode (null, v, null, null));  
    else  
        v.setRight (new BTNode (null, v, null, null));  
    size++;  
}
```

Dioecia *uvaria*



Dioecia *pifas*



Dioecia *conspicua* *uvaria* и
аэробиц 1 пайді

```

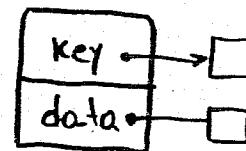
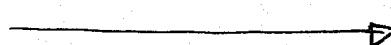
public void deleteNode (BTNode v)
{
    if (isRoot(v))
    {
        BTNode son = (v.getLeft() != null ? v.getLeft() : v.getRight());
        root = son;
        if (root != null)
            root.setParent (null);
    }
    else
    {
        BTNode parent = v.getParent();
        BTNode son = (v.getLeft() != null ? v.getLeft() : v.getRight());
        if (isLeft(v))
            parent.setLeft (son);
        else
            parent.setRight (son);
        if (son != null)
            son.setParent (parent);
    }
    size--;
    v.setLeft (null);
    v.setRight (null);
    v.setParent (null);
    lastDeletedElement = v.getElement();
}

```

ΟΥΠΕΣ ΤΙΠΟΤΕΡΑΙΟΤΗΤΑΣ

- ΑΤΔ Priority Queue

```
class PQEntry
```



```
private Comparable key;
```

```
private Object data;
```

```
PQEntry(Comparable k, Object d)
```

```
{
```

```
    key = k;
```

```
    data = d;
```

```
}
```

```
public Comparable getKey() { return key; }
```

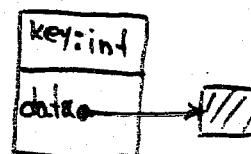
```
public Object getData() { return data; }
```

```
public void setData(Object d) { data = d; }
```

```
public void setKey(Comparable k) { key = k; }
```

```
}
```

Typecast: Comparable \leftrightarrow int



interface Priority Queue

{

```

public void insert ( PQEntry entry )
public Comparable findMin(); // findMax
public PQEntry deleteMin(); // deleteMax
public boolean isEmpty();

```

```
public int size();
```

}

- Υποτοινόν με ταξινομητική διαδικασία σε διάρκεια

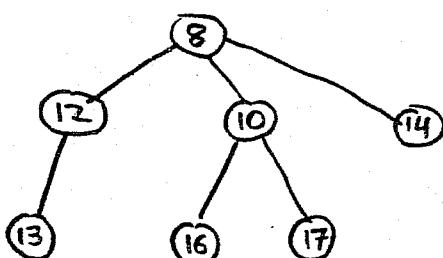
insert	$O(n)$
findMin	$O(1)$
delete Min	$O(1)$
isEmpty	$O(1)$
size	$O(1)$

- χρήσερη τεριτωρίου
- ο αριθμός των στοιχείων στην ουρά προγραμματίζεται n

ΔΕΝΑΡΙΑ ΜΕ ΤΑΞΙΝΟΜΗΣΗ ΣΕΡΟΥ

- Κάθε κόρβας έχει ένα ή περισσότερα παιδιά
- Δεν παραδίδονται τους εξωτερικούς κόρβους
- Το μέγεθος κόρβου είναι \geq από αυτό του κόρβου-πατέρα του

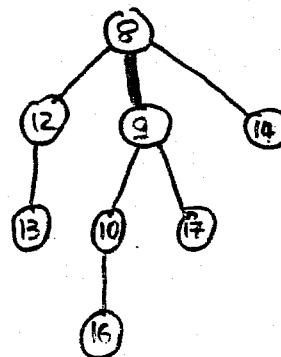
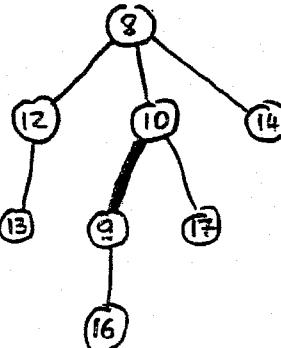
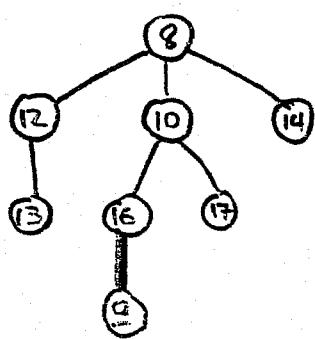
ΠΑΡΑΔΕΙΓΜΑ



- ((x)) συμβολίζει την αριθμό παιδιών του κόρβου x

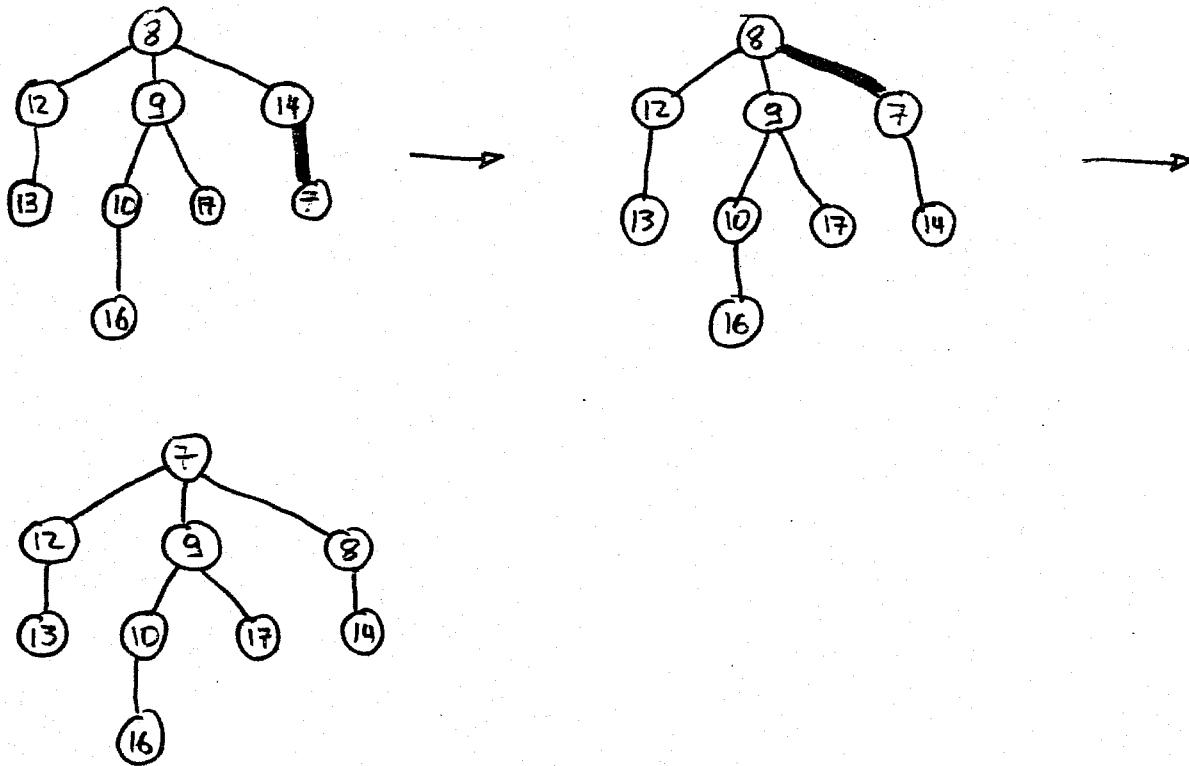
addLeaf:

- Προσθίζει ένα νέο παιδί σε εναν κόρβο
(ή "χοντρό" γερμανικό διάλογο οτι η διάταξη-σερού δεν ισχύει)



- Το νέο μέγεθος είναι τέλοτε μικρότερο από το μέγεθος των αντικατούσαν \Rightarrow μπορεί να είναι ευρός διάταξης μόνο μη το πατέρα του (και όχι μη τη πατέρα του)

ΠΑΡΑΔΕΙΓΜΑ (addLeaf)



- ΑΝΑΛΥΣΗ ΤΗΣ ΔΙΑΔΙΚΑΣΙΑΣ addLeaf (χειρός περίπτωση)

• Στη χειρότερη περίπτωση, το μέτρο ναίς νέου καρφού.
× συγχρίνεται με όλους τους "προγόνους" του, με
συνολικό αριθμό $d(x)$, οπου $d(x)$ συμβολίζει το
βάρος (defth) του × στο δίδυμο.

- Το υιοίμε ενός σωρού Τ μέσω addLeaf διαδικασιών
κατεξί:

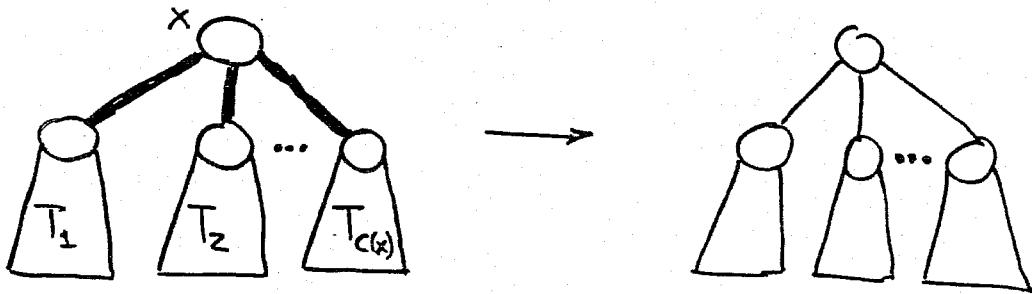
$$W(n) = \sum_{x \in I(T)} d(x) = i(T) -$$

→ internal path length

—————
→ πλήρης "παντοποιημένη" καρφωμένη του δέντρου.

addRoot:

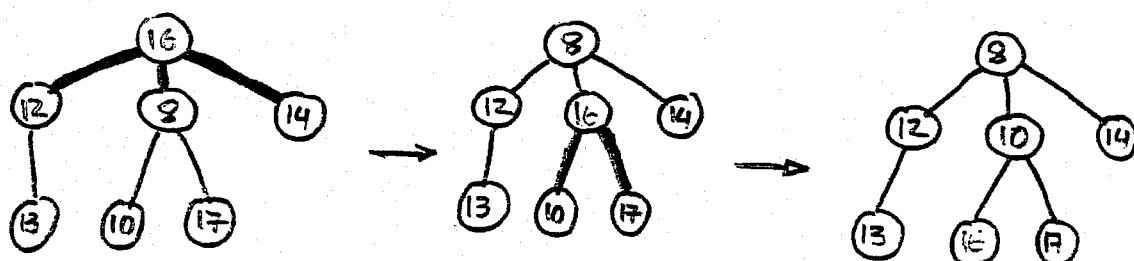
- "Προσθίζει" μια νέα ρίζα σε ένα δένδρο.

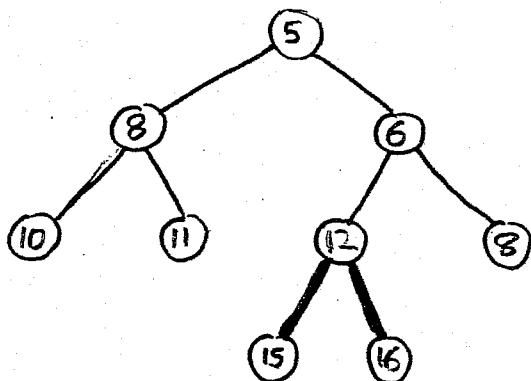
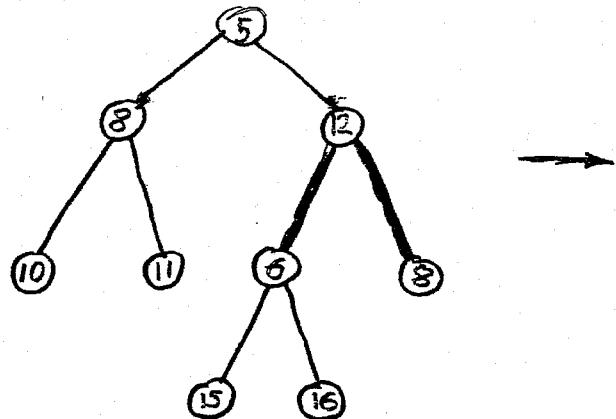
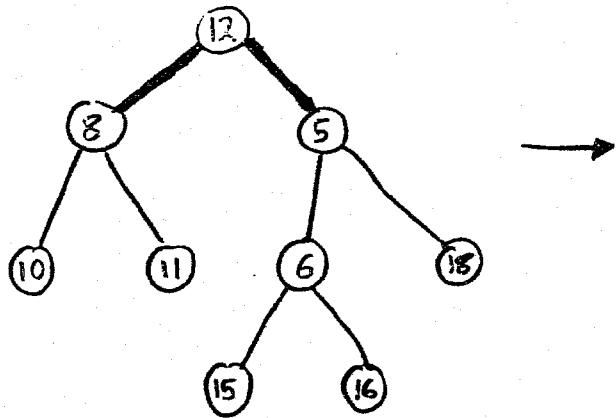


- Εάν το x δεν έχει παιδία, οποιαδήποτε αγωγή που είναι γένος του x θα μπορεί να γίνει παιδί του x .
- Αλλιώς, μεταβιβάζεται το παιδί του y στο x με τη μεταρρύθμιση $y \leftarrow addRoot(y)$

If $\text{keyOf}(y) < \text{keyOf}(x)$

- ανταλλάσσεται το x με το y
- εφαρμόζεται $\text{addRoot}(x)$ αναδρομικά

ΠΑΡΑΔΕΙΓΜΑ 1

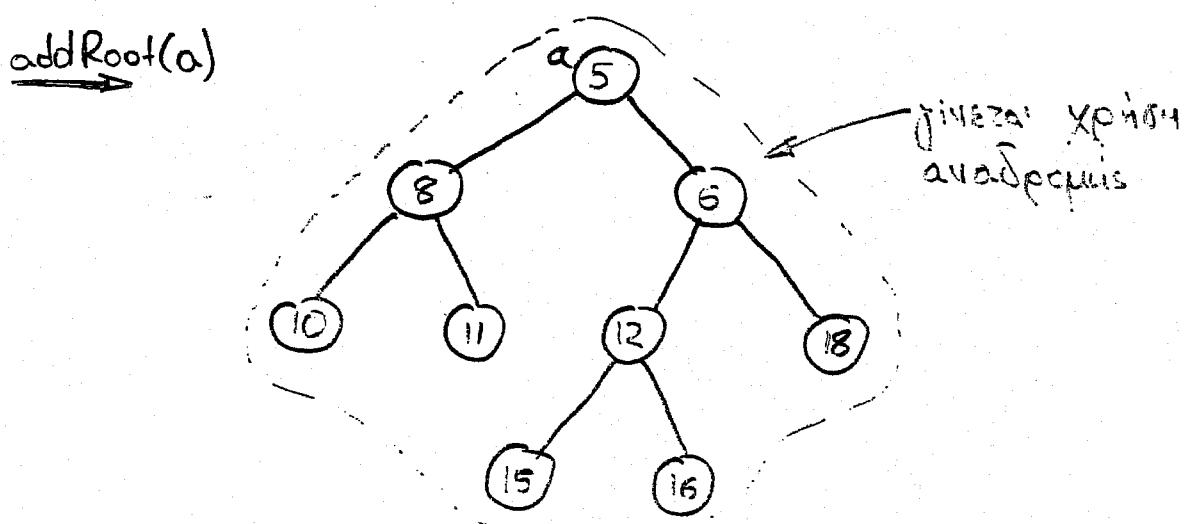
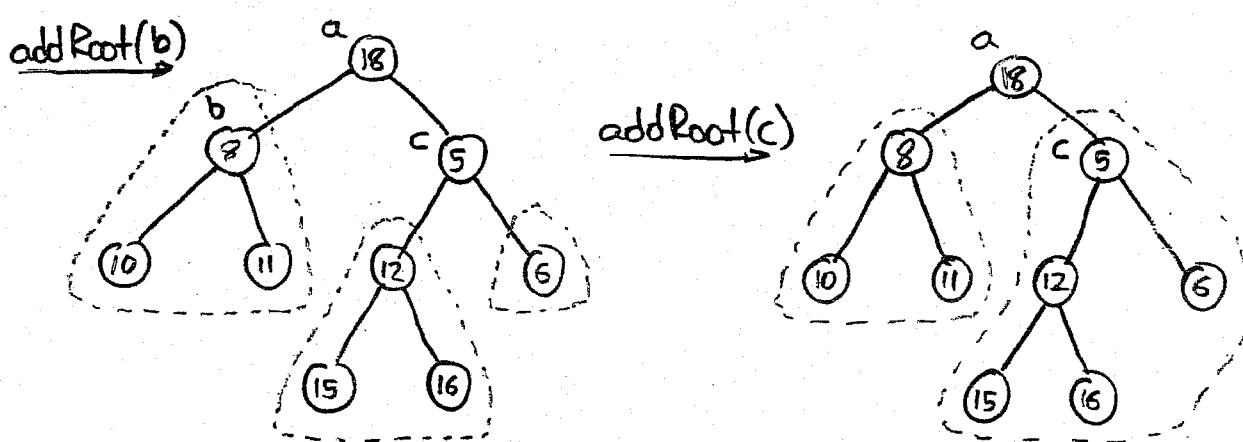
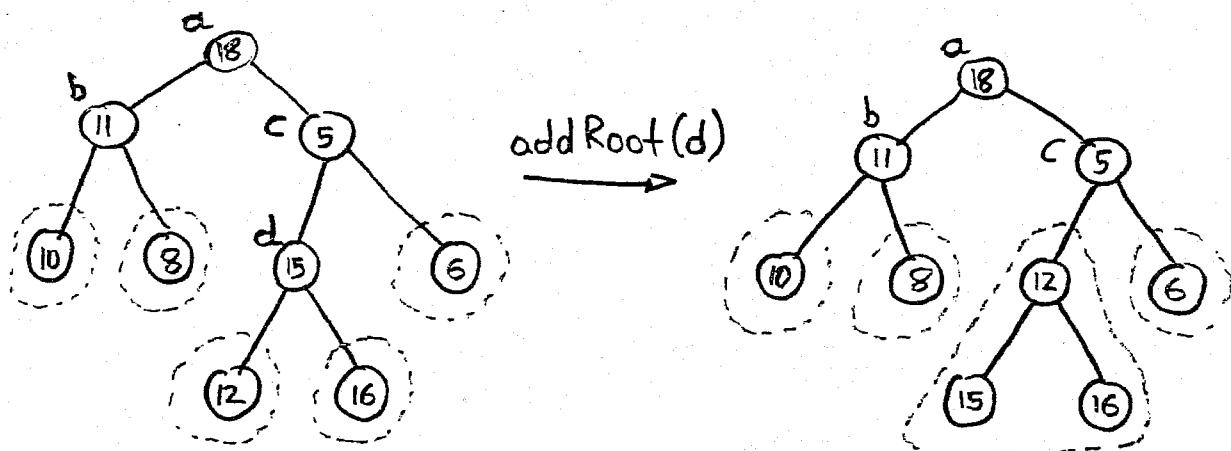
ΠΑΡΑΔΙΓΜΑ - 2

- Ανάλογη τις διαδικασίες addRoot (χειρότερη περίπτωση)
 - Εάν το x έχει $c(x)$ παιδιά, χρειάζονται $c(x)-1$ συγχρίσεις για την εύρεση του παιδιού με ελάχιστο μήκος (y) και 1 επιπλέον συγχρίση μεταξύ x και y για να βοηθεί έτσι χρειάζεται να αναζητηθούν.
 - Στην χειρότερη περίπτωση, το x ανιχνεύεται μεταξύ μονοπάτι, και οι $c(x)$ συγχρίσεις γίνονται $c(x)-1$ φορές. [$c(x)$ είναι το "υψός" του δένδρου με ρίζα το x]

$$W(x) \leq c \cdot (h(x) - 1)$$

Οπου c είναι έτσι άνω φάσμα για το $c(x)$.

ΔΗΜΙΟΥΡΓΙΑ ΣΕΓΡΟΥ ΜΕΣΩ addRoot ΔΙΕΡΓΑΣΙΑΣ

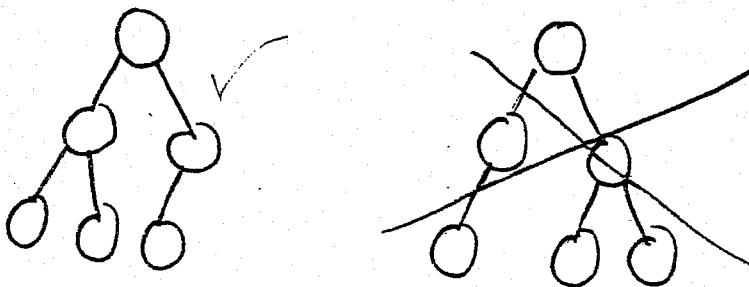


Ανάδυνη δυμούργιας "σωρού" μέσω addRoot

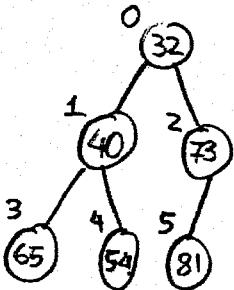
- Το μόνος χιονιάς πώλης κόμβος x είναι το πολύ
 - c. $(h(x)-1) \rightarrow$
$$N(n) \leq c \sum_{x \in I(T)} (h(x)-1)$$

"ΣΩΡΟΣ"

- Δυαδικό δεύτερο με διάφορη σωρού το οποίο είναι πλήρης και όλοι οι κόμβοι του τελευταίου επιπέδου είναι αριστερά - στριχωμένοι
 - Όλοι οι κόμβοι που δεν έχουν παιδιά βρίσκονται στα 2 τελευταία "επιπέδα"



- Αποδίνουμε ότι δένδρο σε διάνυσμα ως εξής:



0	1	2	3	4	5	
32	40	73	65	54	81	...

ΣΟΡΟΣ: heap-ordered, left justified,
complete binary tree,
αποδινούμενο σε διάνυσμα.

Πατ η κατανόηση των βρικών στη διάνυσμα position(x)
τοχυτή:

$$\text{position}(\text{parent}(x)) = \left\lceil \frac{\text{position}(x)}{2} \right\rceil - 1$$

$$\text{position}(\text{left child}(x)) = 2 * \text{position}(x) + 1$$

$$\text{position}(\text{right child}(x)) = 2 * \text{position}(x) + 2$$

Υλοποίηση της insert

- Η διαμερίσια insert υλοποιείται σαν addLeaf ή με θέση size.

insert (PQEntry entry)

- A [size] = entry
- addleaf (size) // ← είναι αυτόφορη
- Size = size + 1

- Ανάδειξη χειρότερων περιπτώσεων για τη διαμερίσια insert

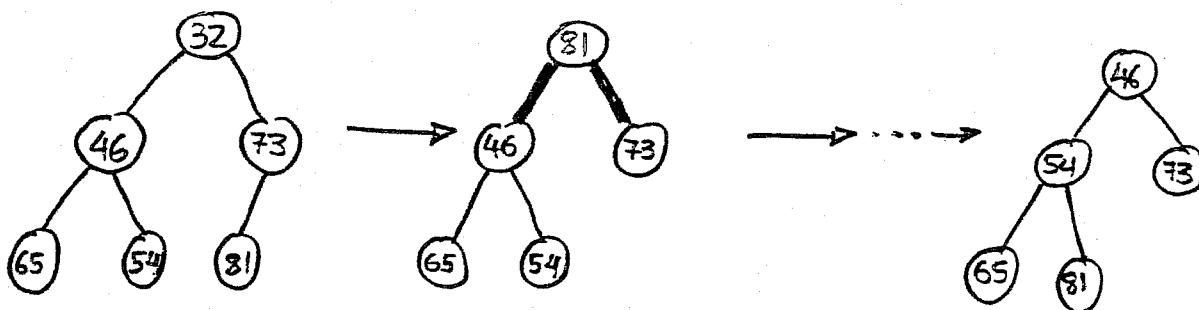
-Εάν ο αριθμός περιέχει η συστοιχία, τότε το addLeaf γίνεται σε θέση n. Το νέο συστοιχό βρίσκεται σε περισσότερο δύνατο βάθος.

$$\begin{aligned}
 W(n) &= \text{το βάθος του νέου συστοιχού} \\
 &= h(T) - 1 \\
 &= \lceil \log_2(n+1) \rceil - 1
 \end{aligned}$$

ΥΠΟΛΟΙΨΗ ΤΗΣ deleteMin

- Υπολογίζεται μεταφέρεται το πλευρά στην αρχή ορθώς
στην πίσια και υπόκειται στο addRoot(0)

↑ πρώτη



PQEntry deleteMin()

- ```

 |-----|
 | PQEntry x
 | x = A[0]
 | size = size - 1
 | if size > 0
 | A[0] = A[size]
 | addRoot(0) //← αναδρομή
 |
 | return x

```

• Ανάλυση χρόνες τροπής και  $\text{deleteMin}$

• Εάν  $n=1$ , τότε  $W(n) = O(1)$

• Εάν  $n > 1$ , τότε

$$W(n) = \text{το } \text{χρόνο } \text{της } \text{addRoot}(\emptyset) \\ \leq 2(h(\emptyset) - 1)$$

Λύθηκε

Λέτο την ανάλυση  $\text{addRoot}$ .

$$= 2(\lceil \log_2(n+1) \rceil - 1)$$

## Υδοτοινον Priority Queue με ΣΟΡΤΟ

|            | SCRTED<br>LINKED LIST | HEAP        |
|------------|-----------------------|-------------|
| insert     | $O(n)$                | $O(\log n)$ |
| find Min   | $O(1)$                | $O(1)$      |
| delete Min | $O(1)$                | $O(\log n)$ |
| IsEmpty    | $O(1)$                | $O(1)$      |
| size       | $O(1)$                | $O(1)$      |

- ΤΑΞΙΝΟΜΗΣΗ ΣΕΡΟΥ ΉΕΑΡ ΣΟΡΤ

Ταξινόμηση n στοιχείων ( $a_1, \dots, a_n$ )

1. Priority Queue  $q$

2. For  $i=1..n$

    insert  $\rightarrow a_i$  στην  $q$

3. For  $i=1..n$

    deleteMin από την  $q$

• Τα στοιχεία εξόργωνται σε αυτούσια σερά.

### Ανάλυση ΗΕΑΡ ΣΟΡΤ

- Eπειδή

$$\begin{aligned} n \text{ insert} &\rightarrow n \cdot O(\log n) \\ n \text{ deleteMin} &\rightarrow n \cdot O(\log n) \end{aligned} \quad \left. \begin{array}{l} \rightarrow \\ \rightarrow \end{array} \right\} = O(n \log n)$$

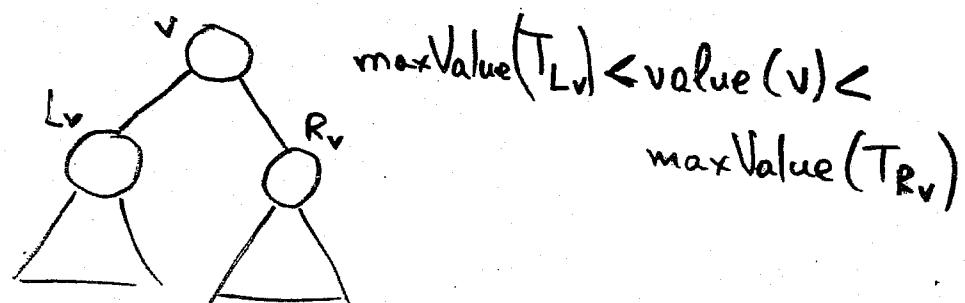
# ΔΕΝΔΡΑ ΑΝΑΖΗΤΗΣΗΣ (Search Trees)

## ΑΣΥΓΙΕΤΑ ΔΥΑΔΙΚΑ

- Κάθι ούρβος έχει μία "ζιρή" (υλιδή)

- Για νέας ούρβον ριχτεί:

"Οι ζιρές των κόμβων σαν αριστερό υπόδενδρο του  $v$  είναι μηκούστερες από τις ζιρές του  $v$  και οι ζιρές των ούρβων του δεξιού υπόδενδρου του  $v$  είναι μεγαλούστερες από τις ζιρές του  $v$ .

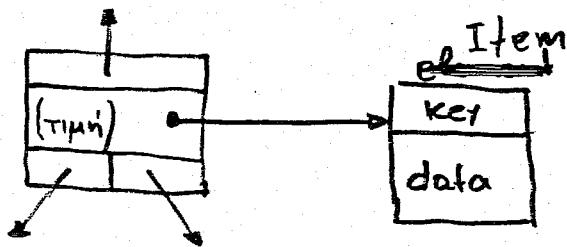


- Στην περίπτωση που επιρρέπεται αριχτά μία ίδια ζιρή:

$$\maxValue(T_{Lv}) \leq \text{value}(v) < \maxValue(T_{Rv})$$

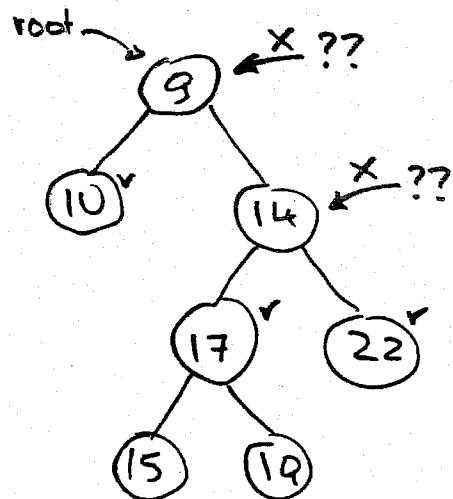
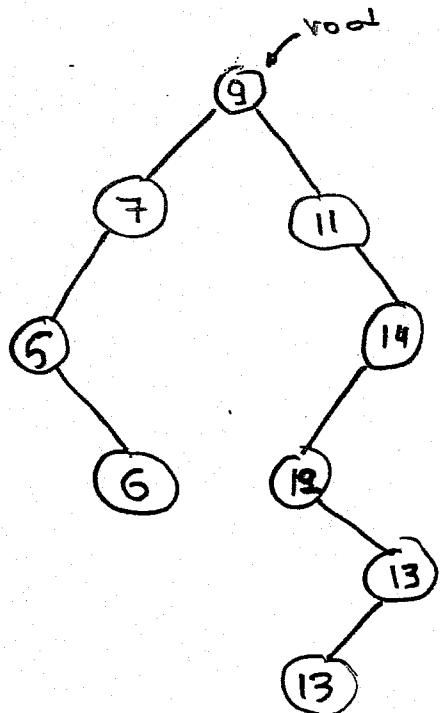
- node-oriented binary search trees  
(κομβοποσαρτολογικά δυαδικά δένδρα αναζήτησης)

- Η δομή ενός νόμερου



Η στρuktur τροποποίησης από το περιεχόμενο του 'item'

Μπορεί να είναι  
αυτοσχέδιο το key.



Λανθασμένο  
"δευτερο ουράγιο"

Συρτάδες διάμερα αναλύμανσης

• ΑΝΑΖΗΤΗΣΗ ΣΤΟΙΧΕΙΟΥ

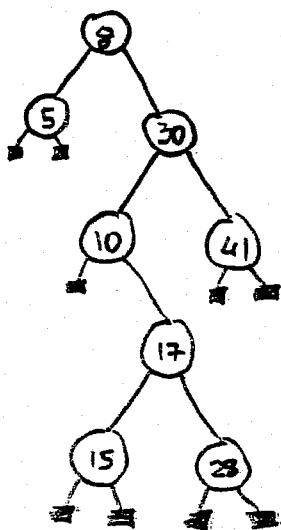
Άρχοντας: findNode(BTNode v, Object key)

Ελεύθερος: Ο κόμβος του Tree οπού θα εμφανίσει την αναζήτηση  
στοιχείο  $\Rightarrow$  με  $v = \text{key}$

```

1. if (key < key(v))
2 if (o v δεν έχει αριστερό παιδί) return v
3 else
4 return findNode(αριστερό παιδί του v, key)
5 else if (key == key(v))
6 return v;
7 else
8 if (o v δεν έχει δεξιό παιδί) return v
9 else
10 return findNode(δεξιό παιδί του v, key)

```



• Εάν έρει στο key

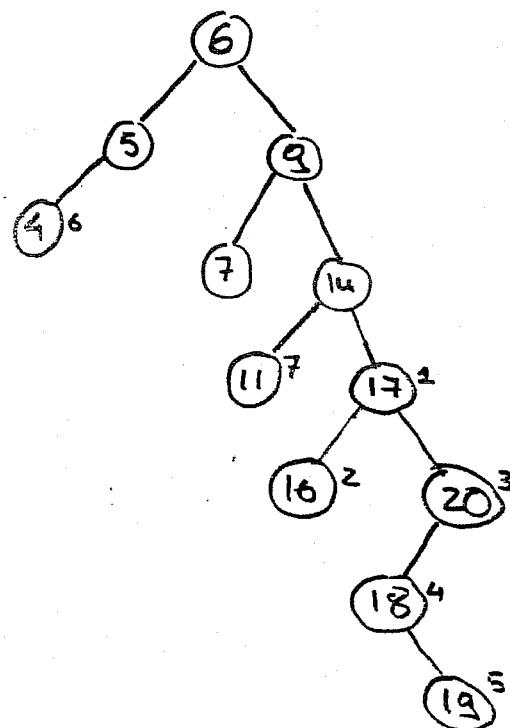
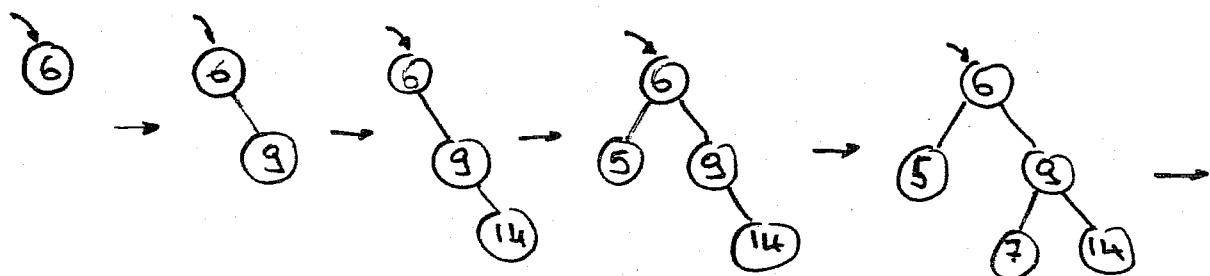
επιστρέψει τον κόμβο που έχει το key

• Εάν δεν έρει στο key

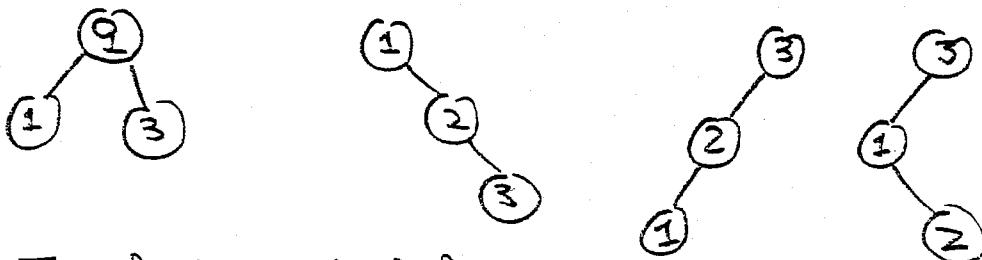
επιστρέψει τον κόμβο που θα  
μπορεί να έχει το key ως  
τίτλο των παιδιών του (εάν είχετε)

ΣΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ

Παράδειγμα: 6 - 9 - 14 - 5 - 7 - 17 - 16 - 20 - 18 - 19 - 4 - 11



- Διαφορετικά δίδυμα για τη ίδια συνάρτηση



Τοπα διαφορετικά δίδυμα;

Algoritmhos insertItem ( Item i )

Efodos: o nuphos tou da replixxi zo i

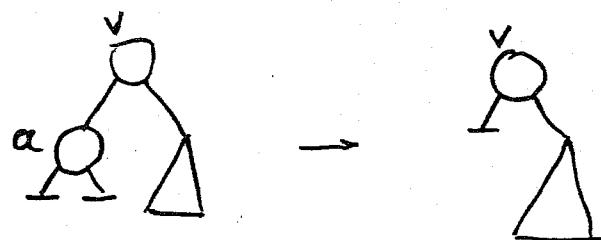
```

1. x = key(i) //to nuphi zou i
2. insNode = findNode (root, x)
3. if (key (insNode) == x)
 return null //nupexei ozo dimos
4. else if (key (insNode) > x)
5. { dimisouregkis vio nupho w ws apiesqo
6. parhi zou insNode kai zatolitikos zo i
7. }
8. else
9. { dimisouregkis vio nupho w ws desgi
10. parhi zou insNode kai zatolitikos zo i
11. }
12. }
13. return w;

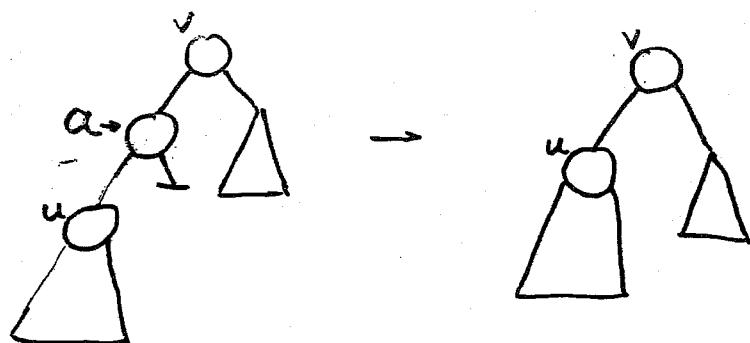
```

## ΔΙΑΓΡΑΦΗ ΣΤΟΙΧΕΙΟΥ

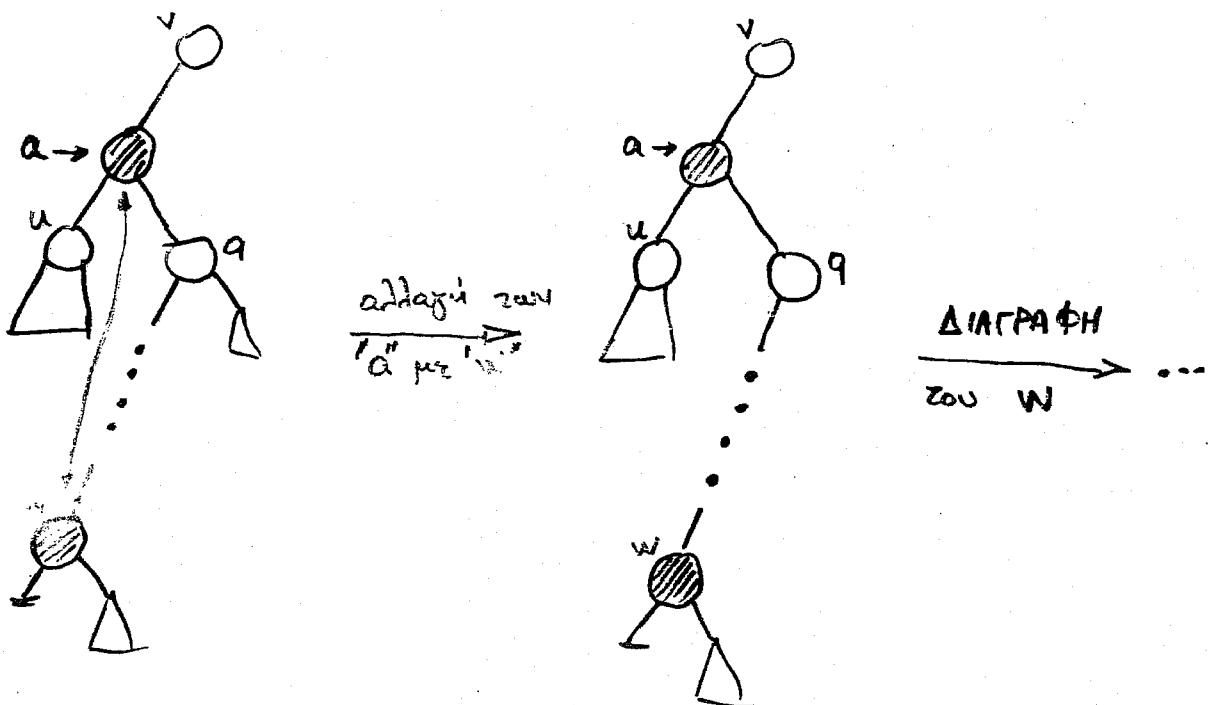
ΠΕΡΙΠΤΩΣΗ - 1 : ο μόρφωσα που θα διαγραφεί  
δεν έχει παιδί



ΠΕΡΙΠΤΩΣΗ - 2 : ο μόρφωσ α που θα διαγραφεί  
έχει 1 παιδί



ΠΕΡΙΠΤΩΣΗ - 3: ο αύριος η που θα διαχραφεί  
εξα 2 παιδιά



→ ΠΕΡΙΠΤΩΣΗ 1 ή 2.

### ΑΣΚΗΣΕΙΣ:

- Βρές το μεγαλύτερο σε αχέιο
- Βρές το μικρότερο τοικύο
- Βρές το επόμενο σε αχέιο

## ΑΝΑΛΥΣΗ

- βάθος μορφου  $v$ : depth( $v$ ): η απόσταση από την ρίζα
- ύψος δένδρου  $T$ : height( $T$ ): το μεγαλύτερο από τα βάθη των μορφών του

## ΑΝΑΖΗΤΗΣΗ / ΕΙΣΑΓΩΓΗ

- έξω  $v$  είναι αποτέλεσμα
- $\approx$  depth( $v$ ) βήματα

worst case:  $O(n)$

average case:  $O(\log n)$

## ΔΙΑΓΡΑΦΗ του $v$

- height( $T_v$ )
- worst case  $O(n)$

Δυαδικά δίνδρα αναζήτησης με μη καθαδικά κλειδιά

- Εισαγωγή ?
- Διαγραφή ?
- Ανάτυπη ?
- Ταξινόμηση ?

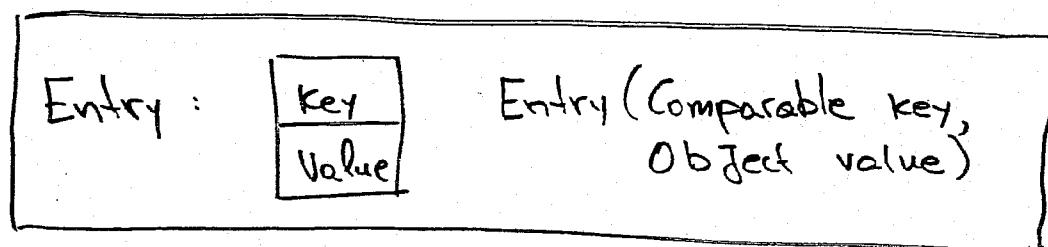
Leaf-oriented Binary search trees [Arg. 7.2]

- (a) ολα τα οροιχεια στα φύλλα  
από δεξιά προς αριστερά, γεγονομένα ↗
- (b) οι ενωσερινοί ψήφοι αποτελούνται μεν  
κλειδιά που είναι ωστε όλα τα τέσσερα νέα  
νικητές:
- $$\text{maxValue}(T_{L_1}) \leq \text{value}(r) < \text{maxValue}(T_{R_1})$$

? Πόσους νέους ψήφους έχει?

ΤΙΝΑΚΕΣ ΣΥΜΒΟΛΩΝ - ΔΟΜΗ ΛΕΞΙΚΟΥ  
 [Symbol Tables - Dictionary]

ATA Symbol Table



- public void insert (Entry x)
- public Entry retrieve (Comparable key)
- public void delete (Entry x)

ATA Ordered Symbol Table

- public Entry retrieveFirst()
- public Entry retrieveFrom (Comparable key)
- public Entry retrieveNext (Entry x)
- public Entry retrieveLast()
- public Entry retrieveUpTo (Comparable key)
- public Entry retrievePrev (Entry x)

ΥΠΟΠΟΙΗΣΗ ΜΕ ΤΑΞΙΝΟΜΗΜΕΝΗ ΛΙΣΤΑ  
ΠΙΝΑΚΑ ΣΥΜΒΟΛΩΝ

- ΔΕΝΔΡΟ ΑΝΑΖΗΤΗΣΗΣ

|               | ΤΑΞΙΝΟΜΗΜΕΝΗ ΛΙΣΤΑ | ΔΕΝΔΡΟ<br>Worst Case | ΑΝΑΖΗΤΗΣΗΣ<br>AVERAGE CASE |
|---------------|--------------------|----------------------|----------------------------|
| Insert        | $O(n)$             | $O(n)$               | $O(\log n)$                |
| retrieve      | $O(n)$             | $O(n)$               | $O(\log n)$                |
| delete        | $O(1)$             | $O(n)$               | $O(\log n)$                |
| retrieveFirst | $O(1)$             | $O(n)$               | $O(\log n)$                |
| retrieveFrom  | $O(n)$             | $O(n)$               | $O(\log n)$                |
| retrieveNext  | $O(1)$             | $O(n)$               | $O(\log n)$                |

Ερώτηση: Μπορούμε να πετύχουμε  $O(\log n)$  απόδοση στην χειρότερη περί πάση;

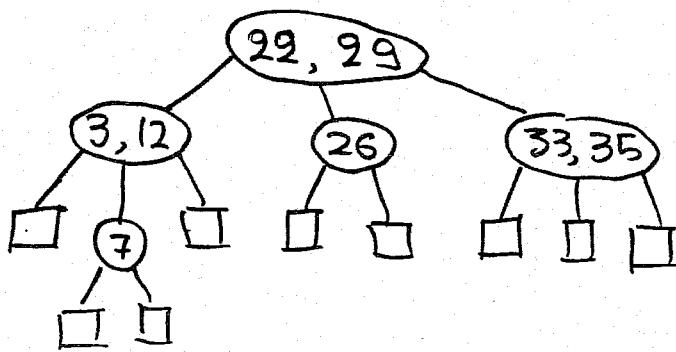
Εξαρνήσεις Πίνακα Συμβόλων

- Ο πίνακας συμβόλων είναι Compiler
- Βάσεις δεδομένων

## B - ΑΕΝΔΡΑ

## B-Trees [Bayer, McCreight, 1972]

- K-οδικά δένδρα αναζήτησης [είναι πολύτιμη αναζήτηση]
  - Δένδρα αναζήτησης με  $\geq 1$  υπεδία σε κάθε κόρυφο

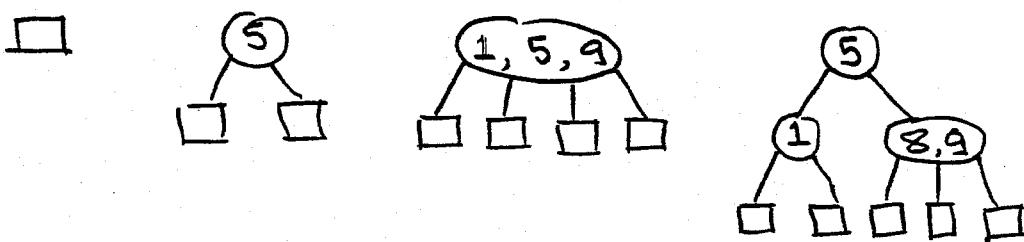


- Τα υπεδία σε κάθε κόρυφο είναι ταξινομημένα
- Η αναζήτηση είναι γενικώς της αναζήτησης σε δυαδικό δένδρο αναζήτησης
- Η διαπέραση είναι γενικώς της inorder διαπέρασης για δυαδικό δένδρο αναζήτησης  
 $\Rightarrow$  Ταξινόμηση των στοιχείων.

- Είναι B-δένδρο τάξεως m μανούσει:

- (1) Είναι ένα δένδρο πολλατής αναζήτησης
- (2) Η ρίζα είναι ένα εξωτερικός κόμβος ή  
εχει από 2 έως m παιδιά
- (3) Κάθε εσωτερικός κόμβος (η ρίζα μπορεί να αποτελεί  
εξαιρέσιμη) εχει από  $\lceil \frac{m}{2} \rceil$  έως m παιδιά
- (4) Όταν οι εξωτερικοί κόμβοι έχουν τα ίδια  
παιδιά

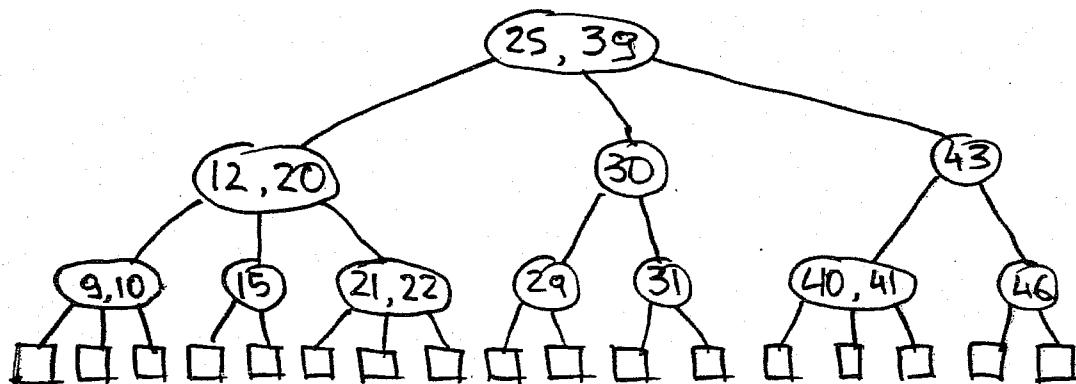
ΠΑΡΑΔΕΙΓΜΑ: B-δένδρο τάξεως 4.



- Οι δυνατότητες (2) και (3) επιτρέπουν τη χρήση  
εσωτερικού κόμβου μίκην σε κάθε κόμβο.  
[m αναφορές, m-1 ιδεώδια, 1 μερικές αριθμούς αλεντίσια]
- Η τουλάχιστη (4) εγκυάται 1 ποσούλισμαν δένδρο

B-δένδρα σάγκος 3 (2-3 δένδρο)

- έχουν  $2 \times 3$  παριά / κόρυφα
- ιδανικά για υποτομική πίνακα συρθώσης ή εσωτερική μνήμη (internal memory)



B-δένδρα σάγκος  $m$ , για μεγάλο  $m$  ( $m \geq 256$ )

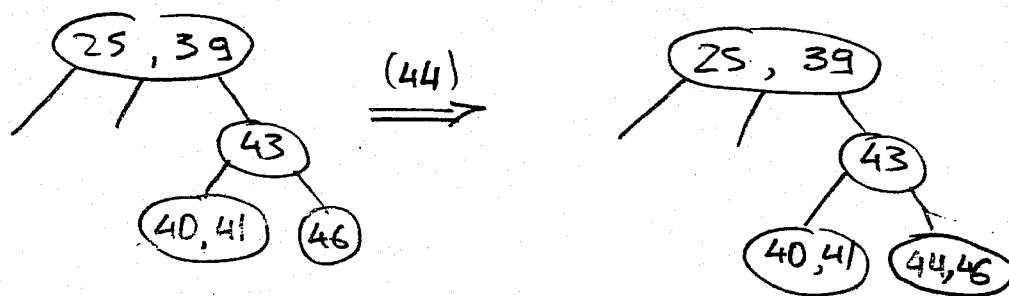
- μικρό βάρος
- εύκολη διαχείριση
- αποδινεύσιμη συσχίσιως στο δίσκο/εξωτερική μνήμη
- κερβούς  $\leftrightarrow$  block δίσκου
- χρήσης αναζήτησης (μικρός αριθμός blocks του δισκού)
- Η χρονοθόρα διαδικασία είναι η προσπέλαση ενός block του δίσκου (μηχανικό μέρος)
- Στοιχεία παραμένουν μόνο σε αποθήκη.
- Οι εγγραφές αποδινεύονται στην εξωτερική μνήμη.

ΕΡΩΤΗΣΗ: Τισι δεύτεροι σημείοι γιατρέψειν;

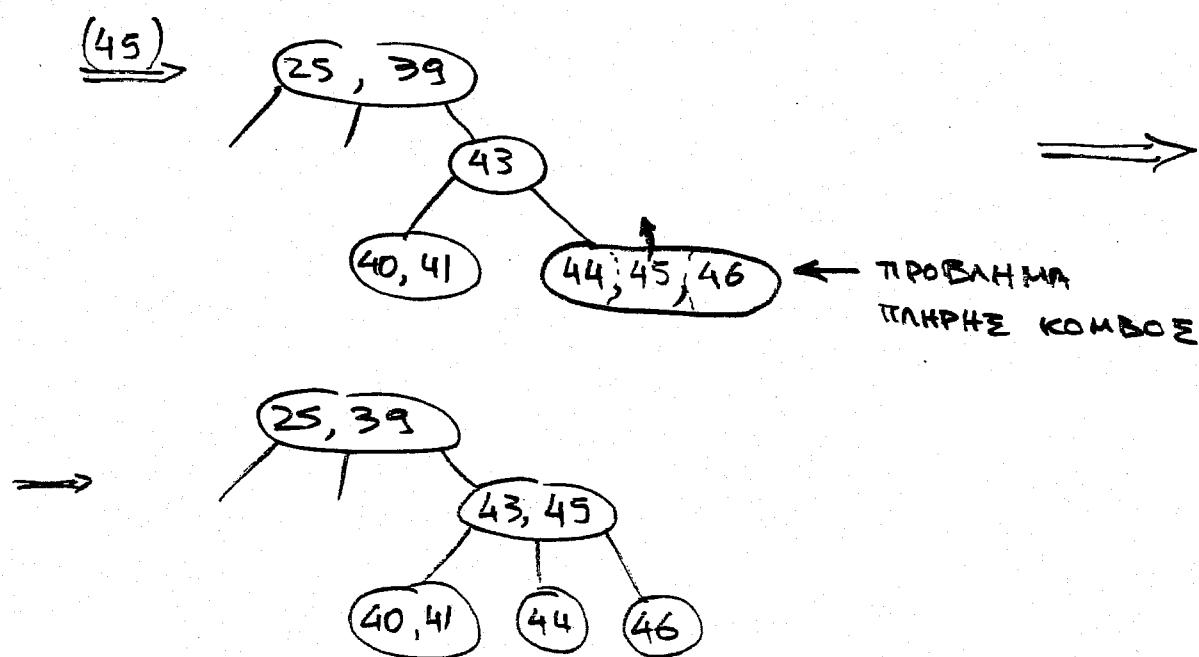
## ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ

Παραδείγμα - Β-διέρρο βαλνς 3 (2-3 δυνάρια)

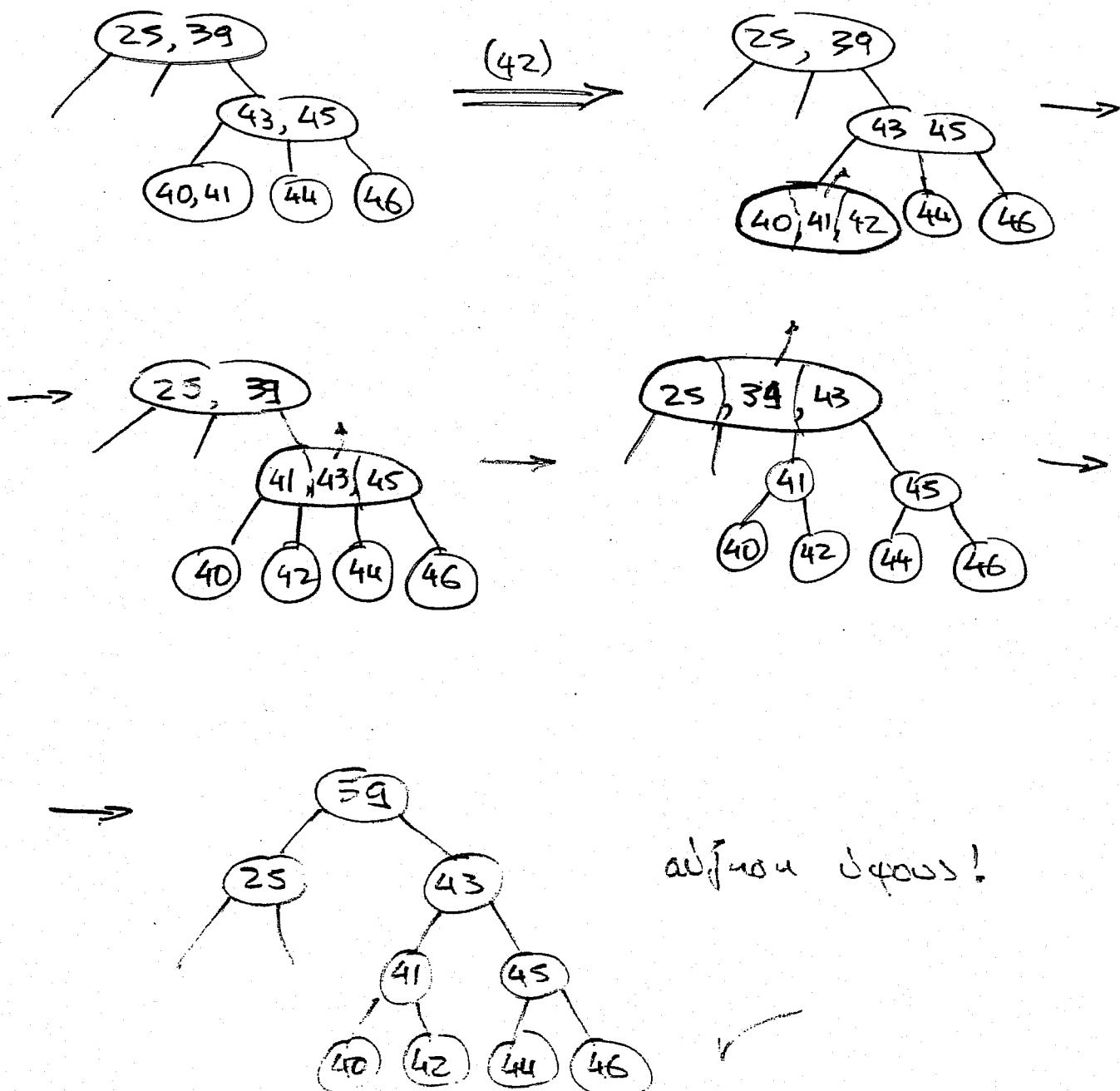
- Ότι αποτελεί γιατο τα σύνοικα κύριου μεγίστου βαλνς



- Εάν ο κύριος είναι "πλήρης" τότε δημιουργούμε αρχικά έναν προβληματικό κύριο.

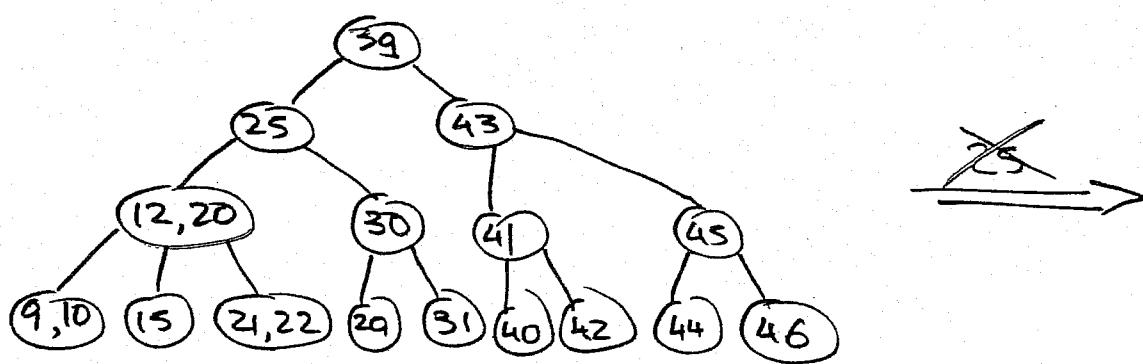


- Το ύψος των δευτερογενών μεγαλώνει εάν χωρίσει στα δύο και πήγα

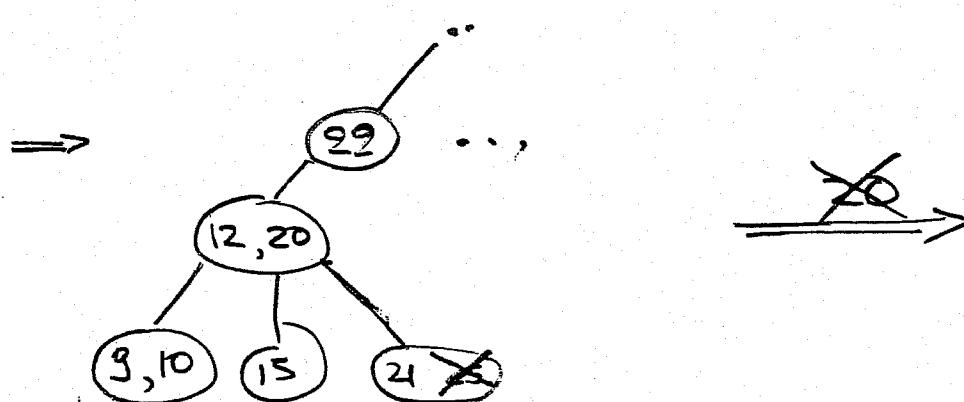


## ΔΙΑΓΡΑΦΗ ΣΤΟΙΧΕΙΩΝ

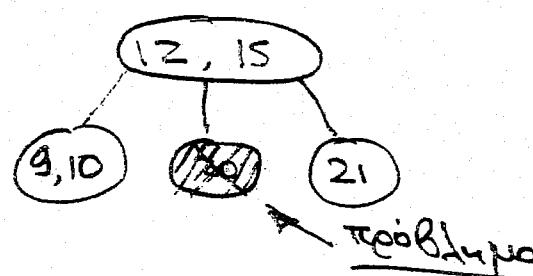
- Η διαγραφή αρχίζει από το τελευταίο στατό.  
[αναδράζει με το προηγούμενο inorder αρχείο]



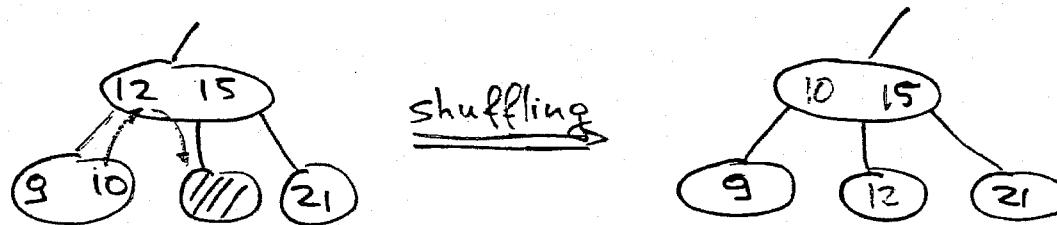
→ αναδράζει το 25 με το 22.



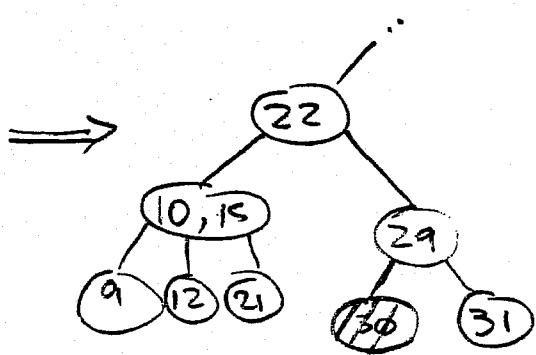
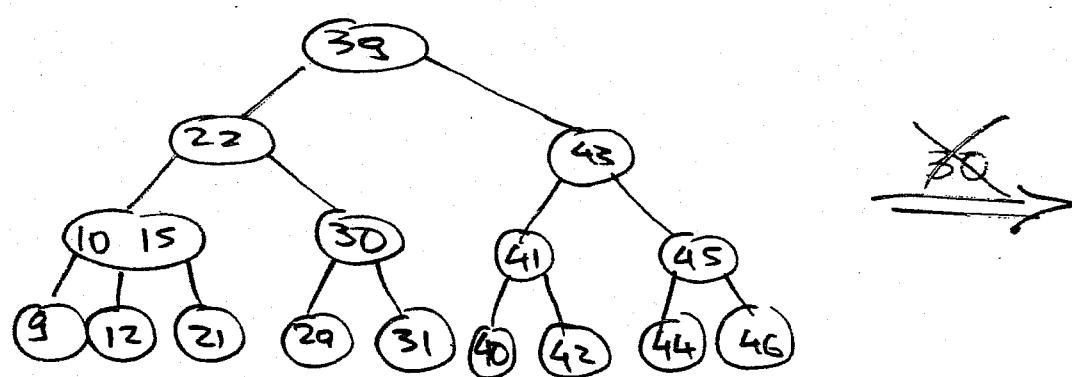
→ αναδράζει 20 με 15 →



→ μεταφορά στοιχίου από το αριστερό (i δεξιά)  
αδέηθι. [shuffling]



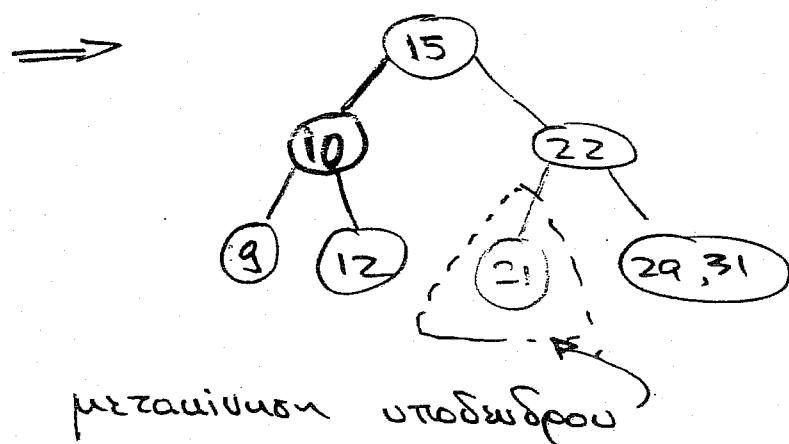
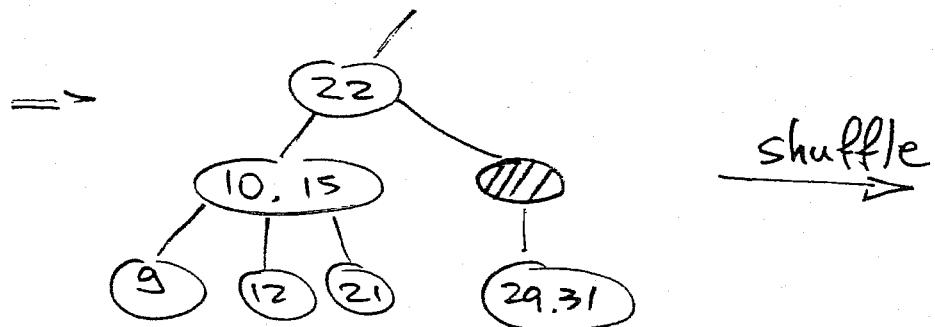
→ Μετεπι τώρα...



⇒ Αν έχει αδέηθι ~~αριστερά δίκαιο~~ για να γίνει  
μεταφορά στοιχίου!

⇒ συγχωνεύοντας με το αδέηθι  
και του κορεα από του  
"Ταξέρα".





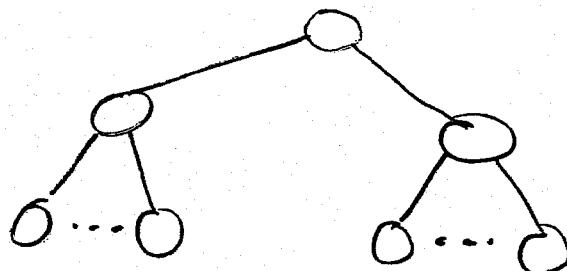
⇒ Εάν η ρίζα "ενεκλεί" τότε μετράνεται το υψός του διώρου.

⇒ Ο μέσιος δέος είναι εύκολος !!

ΘΕΩΡΗΜΑ: Εστιν  $T$  ένα  $B$ -δένδρο τάξης  $m$ , με υψος  $h$  και  $n \geq 1$  στοιχία.

Τότε:

$$n \geq 2 \lceil m/2 \rceil^{h-1} - 1$$



Βαθος

0

1

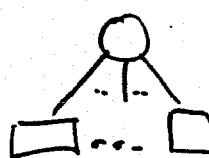
$\vdots$

Κόρδοι

1

2

$2 \lceil m/2 \rceil$



$h-1$

$\vdots$

$2 \lceil m/2 \rceil^{h-2}$

0

Εξαγονώντας τη σύνταξη, ο αριθμός των στοιχίων είναι:

$$2 + 2 \lceil m/2 \rceil + \dots + 2 \lceil m/2 \rceil^{h-2} = 2 \frac{\lceil m/2 \rceil^{h-1} - 1}{\lceil m/2 \rceil - 1}$$

Καθε νόμος περιέχει  $\lceil m/2 \rceil - 1$  στοιχία  $\implies$

$$n \geq 2(\lceil m/2 \rceil^{h-1} - 1) + 1 = 2 \lceil m/2 \rceil^{h-1} - 1$$

πίστα  $\implies$

$$\underline{2(\lceil m/2 \rceil^{h-1} - 1) + 1}$$

→ Αναζήτηση :  $O(\log m) \cdot O(\log n) = O(\log m \cdot \log n)$

δυαδικό φορτίο σε μέριμνα      βαθος δωδρου

$m = O(1) = \text{σταθερά}$

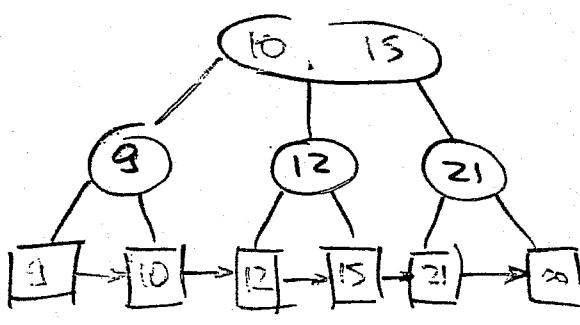
→ Εισαγωγή :  $O(m) \cdot O(\log n) = O(m \log n)$

χαρτί + ειδανή (σπρωχήματα σφρίξεων δεξιά)      βαθος δωδρου

⇒ Διαγραφή :  $O(m) \cdot O(\log n) = O(m \log n)$   
 (μορφα με εισαγωγή . πιο τελυπλόυσα)

Ερώτηση: Τις υποδομές - retrieveFrom()  
- retrieveNext()

- Τα συνθήματα σαν εξωτερικούς κόρδους
- Τα μεταβλήτα σαν εσωτερικούς κόρδους
- Οι εξωτερικοί κόρδοι διαδικτύουν σε ζεστά



B-tree

|               |             |                      |
|---------------|-------------|----------------------|
| insert        | $O(\log n)$ | Symbol Table         |
| retrieve      | $O(\log n)$ |                      |
| delete        | $O(\log n)$ |                      |
| retrieveFirst | $O(1)$      | Ordered Symbol Table |
| retrieveFrom  | $O(\log n)$ |                      |
| retrieveNext  | $O(?)^*$    |                      |

-  $(a,b)$  - διέργο  $a \geq 2$   $b \geq 2a$

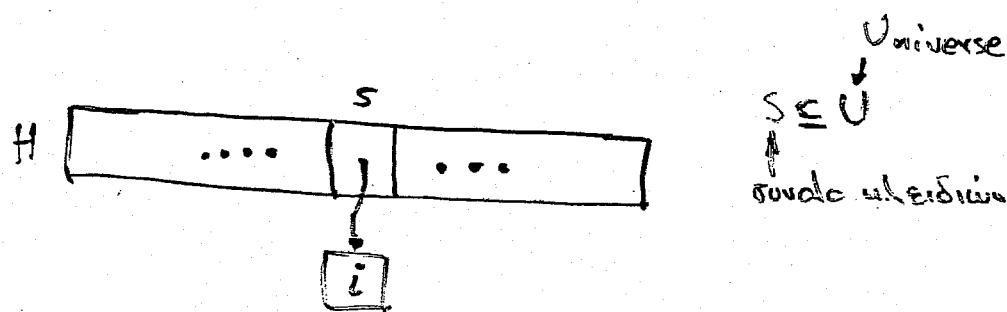
- (a) Το δίττα εχουν ίδια διάστασης
- (b) Η πίτα εξετάζεται 2 και το πάνω b πλευρά
- (c) Το πήδη από παρθενικό μέρος να περνάει από την b σερβιτούνταν μεταξύ αυτών
- (d) Διέργο πολλαπλές αναζητήσεις

- Κεφ. 9 βιβλίου
- B-trees : Ασύρμον Α.3 (a)

## KATAKERMATIΣMOΣ

### [Hashing]

IDEA: Εάν το μήδι  $s \in S$  είναι δροχικό κ. ήταν αυτόπαιος αριθμός, τότε θα μπορούσε να "είναι" η θέση ενός πινακισθόπου το οποίο αποδινεγμένο, βαθιά,  $H(s) = i$

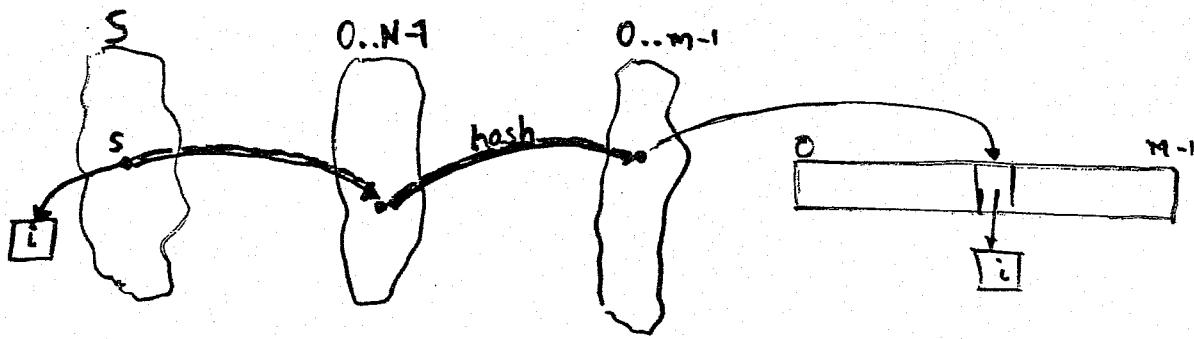


- Η πύρος αποδινεγμένος χώρος  $O(|U|)$ : αυξέντιο
- $O(1)$  χρόνος προστίλλοντος

#### Εδώτι γίνονται:

- Χρήση "διανεύσματος" της θέσης
- Συναρτησην μετατόπισης:  $S \rightarrow [0..N-1]$ ,  $N = |U|$
- Συναρτηση  $\left\{ \begin{array}{l} \text{αποδίδεσης (allocation)} \\ \text{μετίσκεψης (compression)} \\ \text{καταστροφής (hash)} \end{array} \right\} [0..N-1] \rightarrow [0..m-1]$
- Έχει  $N \ll m$   
 $N \gg |S|$

Εγκρίσια:



## Συναρτήσεις κατανεύμασης

Μέθοδος των διάφερων:

- απλή
- δυχιά χρησιμοποιούμενη

$$h(x) = x \bmod m \quad \text{όπου } m \text{ είναι πρώτος αριθμός}$$

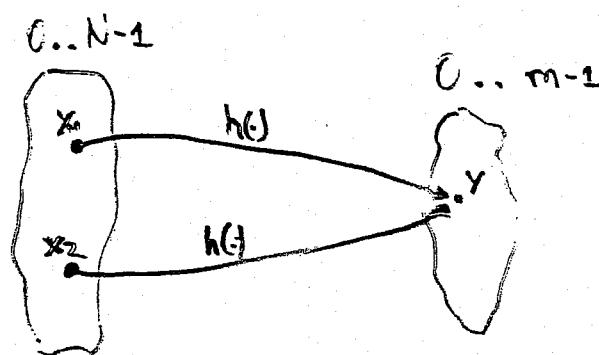
Μέθοδος Η-θετησιακού - Ταυτόσημου - Διαίρεσης

(MAC : Multiply - Add - Divide)

$$h(x) = (ax + b) \bmod m \quad a, b \text{ τυχαιοί} \\ m \text{ πρώτος αριθμός}$$

## ΠΑΡΑΤΗΡΗΣΕΙΣ

- Η δυναργίδας μεταμερισμού πρέπει να "αποτελέσει" τις συγκρόσεις (collisions)



... αδικαστήριο του ότι  $N \gg m$

⇒ Τρέπει τα σύμπα [0..N-1] να μετατίθεται  
ισομεράς στο διαφέρον διάστημα απόδικετων  
τίτλων

⇒ Τρέπει να απάρτιι μέριμνα για τις  
αντιμετώπιση των συγκρόσεων

### Ομοιομορφή Υπόδειξη

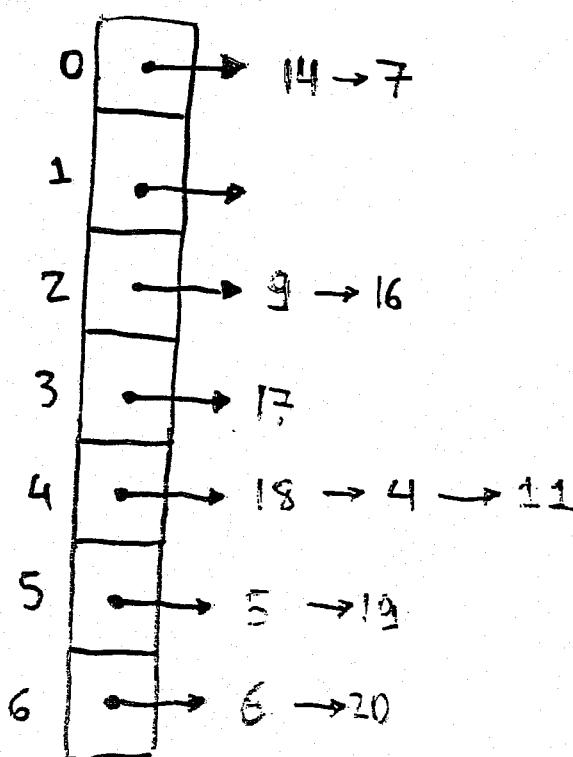
- Η δυναργίδη μεταμερισμούς μετατίθεται σε αντικαντών τις διαβίσμες δέσμων
- Ηδη τα μέρη των σύμπαντων είναι τα ίδια  
πλέοντας στην αποτέλεσμα στα ίδια  
πλέοντας

## Κατανεματικός με Λδούδες ( chaining )

- Κάθε δίσκος είναι το πίνακα κατανεματικών (hash table) "περιέχει" μια δορυφορική λιστή ως δίσκο, η οποία συγχέει όλα τα ουραχία που απεικονίζονται από τη συγάρτηση κατανεματικού στη δίσκο ή

Παραδείγμα:  $h(x) = x \bmod 7$ , αδιάστατη δίσκος

Συρά εισοδημάτων:  $\langle 6, 9, 14, 17, 5, 7, 16, 20, 18, 19, 4, 11 \rangle$



## Evaluating HashCodes

```

public class HashOperator
{
 private int m;

 HashOperator (int divisor)
 {
 m = divisor;
 }

 public int hashTo (Object k)
 {
 return k.intValue() % m;
 }

 public int hashTo (int k)
 {
 return k % m;
 }
}

```

## 3. Hash Operator

• HashCode can be evaluated  
 • HashCode computation uses  
 • `Object.intValue()`

public class HashingWithChaining

```
{
 private int size;
 private int m;
 private DictionaryNodeList hashTable[];
 private HashOperator h;
```

HashingWithChaining(int tableSize, HashOperator hs)

```
{
 h = hs;
 size = 0;
 m = tableSize;
 hashTable = new DictionaryNodeList[m];
```

for (int i=0 ; i < m ; i++)

hashTable[i] = new DictionaryNodeList(new Comparator())

```
}
```

HashingWithChaining(HashOperator hs)

```
{
```

this(101, hs);

```
}
```

```

public Item findItem (Object key)
{
 return hashTable [h.hashTo (key)]. findItem (key);
}

```

```

public void insertItem (Item item)
{

```

```

 if (findItem (item.getKey ()) == null)
 {

```

```

 hashTable [h.hashTo (item.getKey ())]. insertItem (item);
 size++;
 }
}

```

```

public void deleteItem (Object key)
{

```

```

 if (findItem (key) != null)
 {

```

```

 hashTable [h.hashTo (key)]. deleteItem (key);
 size--;
 }
}

```

I // Hashing With Chaining

### Άναλυση:

ΠΗΜΑ : Σε μια δορή κατανεμούμενης με αλυσίδες.

Η στοιχείων :

- Το μέσο μήνας των αλυσίδων είναι  $a = n/m$  με πιθανότητα που τις έχει στο 1
- μια αναζήτηση (αποωχημένη ή επιωχημένη) πραγματίζεται  $O(1+a)$  χρόνο

To  $a$  ονομάζεται παράγων φορτίου (load factor)

ΤΕΣΡΗΜΑ : Μια αυτοκίνητα από  $n$  ενδέξεις, αποθέτεις ναι αναζητήσεις σε μια δορή κατανεμούμενης με αλυσίδες, απαιτεί χρόνο  $\left( n \cdot \frac{1}{2} + \frac{n}{2} \right)^a$

ΛΗΜΜΑ : Σε μια δορή κατανεμούμενης με αλυσίδες, η στοιχείων ναι παραγωγή φορτού  $a < 1$  και μέσο μήνας των μεγαλύτερης αλυσίδας είναι

## ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ ΧΩΡΙΣ ΛΙΣΤΕΣ [OPEN ADDRESSING]

- Στερεοί τα συγχρονόμενα σύστημα ενώσου του πίνακα κατακερματισμού
- Καθέριη διαχείριση του ελεύθερου χώρου του πίνακα κατακερματισμού
- Διαφοροί τρόπων διαχείρισης ελεύθερου χώρου
  - Γραμμική δομή
  - Τετραγωνική δομή
  - Διάδο υπακερματισμός

## ΓΡΑΜΜΙΚΗ ΔΟΚΙΜΗ [LINEAR PROBING]

- Εξειδικευμένες δίαιρεσης του πίνακα  
τελειώνει από την αρχική δίαιρη κατακερματισμού
- Αναλογική δομή:

$$h_j^* = \{ h(x), h(x) + 1 \bmod m, h(x) + 2 \bmod m, \dots, \\ \dots, h(x) + (m-1) \bmod m \}$$

Οι δίσεις του πίνακα υποτελεστικών  
χαρακτηριστικών είναι:

- Ημερήσιες: εάν οργάνων μέτρο οριχθύον
- Κενωδίσεις: εάν πλατύτερα είχαν οργάνων μέτρο οριχθύον, τότε αριθμός είναι διαδικτικός για πλήρωση
- Κενές: εάν ποτέ δεν ορίζονται μέτρο οριχθύον

Διαδικτικές: κενωδίσεις & κενές

Ταράδιγμα:  $m = ?$ ,  $h(x) = x \bmod ?$

| A | 0  | 1  | 2 | 3 | 4 | 5  | 6  |
|---|----|----|---|---|---|----|----|
| a | 14 |    |   |   | 4 | 12 |    |
| b | 14 |    |   |   | 4 | 12 | 11 |
| c | 14 |    |   |   | 4 |    | 11 |
| d | 14 | 13 |   |   | 4 |    | 11 |
| e | 14 | 13 |   |   | 4 | 13 | 11 |

a insert(14), insert(12), insert(14)

b insert(11)  $\rightarrow h^b = \{4, 5, 6\}$

Θίσεις 4, 5: υποτελεστικές. X

6: κενή →

c remove(12)

Όταν 5: επιναδιστά

d insert(13):  $h^d = \{6, 0, 1\}$  ✓

e find(18):  $h^e = \{4, 5, 6, 0, 1, 2\}$ : 2 κενή X not in

f insert(18):  $h^f = \{4, 5\}$ : 5 διαδικτικός →

Στιγμή: Μια επιτωχήν αναζήσων χρειάζεται,  
κατά μήνας άρο,

$$\frac{1}{2} \left( 1 + \frac{1}{1-a} \right)$$

συγκρίσεις, εώς μια αποτωχήν.

$$\frac{1}{2} \left( 1 + \frac{1}{(1-a)^2} \right)$$

συγκρίσεις, οπου  $a$  είναι ο περίγυρης δόρυς

$$a = \frac{n}{m} \rightarrow \text{στοιχεία στη πίνακα}$$

$$m \rightarrow \text{μελέτη πίνακα.}$$

- \* Όταν  $a \rightarrow 1$  (γενάρως πίνακας) έχουμε  
"άρριψη" αναζήσων δόρυ περιττά αυθούδια.  
από παρελθόμενες δέξεις

## Τεραγωνική δομή (quadratic probing)

- Αυτόνομη δίστανση:

$$h_j^* = \begin{cases} h(x), \\ h(x) + 1 \mod m, \\ h(x) + 2 \mod m, \\ \vdots \\ h(x) + j^2 \mod m, \end{cases}$$

}

$$\Rightarrow h_j^* = h_{j+1}^* + 2j - 1 \mod m$$

\* Πρέπει να γίνει σωστή επιλογή του μεγίστου του πίνακα ( $m$ )!

→ Υπάρχει αινιδύνος να μην εντοπιζούνται σε αυτήν ιδιαίς επίπεδες ενώ άλλες υπάρχουν.

ΛΗΜΜΑ Οταν για  $m$  επιλέγεται πρώτος αριθμός και οι τις πλήρεις σοβαρές των εγγύων περιβάλλοντας τον μηδένιον και μηδενιόντας, τότε είναι πάντας γνωστή η αρχική θέση της δίστανσης.

## ΔΙΠΛΟΣ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ (Double hashing)

- Προσπαθεί να αποφύγει τις μηγέτες αναζητήσεις και για περιμένεια δεξιών

$$h_g^x = h_1(x) + f h_2(x) \bmod m$$

- Η συναρτηση  $h_2(1) \cdot \delta w$  πρέπει να επιστρέψει  $\emptyset$ .

• Ταυτοτελείς υπολογίζει αριθμό μηδεδύτη του  $m$

- Σε πραγματικές εφαρμογές:  $f_2(x) = x \bmod 97 + 1$ .

Θεωρία: Μια επιωχυμένη αναζήτηση χρειάζεται, ώστε μέσου όρο

$$\frac{1}{a} \ln \frac{1}{1-a}$$

συγκρίσεις, ώστε με αποωχυμένη

$$\frac{1}{1-a}$$

συγκρίσεις, όπου  $a$  είναι ο ποσός δέρου.

## Ανακατεμπαρισμός (Rehashing)

- Όταν ο παρόγονας φόρου, δόξως των ενδέσων, γίνεται  $\frac{1}{2}$ ,

→ Σύμφωνα με την πίνακα "διπλόσιο" μεταβιβάζεται  
→ ανακατεμπαρισμός στην πίνακα

- Όταν ο παρόγονας φόρου, δόξως αποδέσων,

γίνεται  $\frac{1}{8}$

→ Σύμφωνα με την πίνακα "καρτών" μεταβιβάζεται  
→ ανακατεμπαρισμός στην πίνακα

Θεώρημα: Μια αυθούδια από n ενδέσεις, αποδέσεις και αναζήτησες απαιτεί χρόνο O(n) και γραμμικό, στο πλίθος των αποτυπωμένων στοιχείων, χωρό

Συγκριτική ημερησίας με δωδεκάες δομές

### Τίτλων Συμβόλων

|                | ΛΙΣΤΑ ΤΑΞΙΝ. | ΔΕΝΔΡΟ ΑΝΑΖΗΤΗΣΗΣ | B-TREE      | Hashing     |
|----------------|--------------|-------------------|-------------|-------------|
| INSERT         | $O(n)$       | $O(n)$            | $O(\log n)$ | $O(\log n)$ |
| RETRIEVE       | $O(n)$       | $O(n)$            | $O(\log n)$ | $O(1)$      |
| DELETE         | $O(1)$       | $O(n)$            | $O(\log n)$ | $O(1)$      |
| RETRIEVE FIRST | $O(1)$       | $O(n)$            | $O(\log n)$ | $O(\log n)$ |
| RETRIEVE FROM  | $O(n)$       | $O(n)$            | $O(\log n)$ | $O(\log n)$ |
| RETRIEVE NEXT  | $O(1)$       | $O(n)$            | $O(\log n)$ | $O(1)$      |

↑

↑

↑

Average

WORST  
CASEAVERAGE  
CASEWORST  
CASE

CASE

## ΕΝΑ ΣΥΝΟΔΑ [Disjoint Sets]

### • ΑΤΑ DisjointSet

- Τα σύνοδα αποτελούν διαμέριση του  $U = \{0..n-1\}$   $n \in \mathbb{N}$
- Αρχικά:  $n$  μονοσύνοδα  
 $\{\emptyset\}, \{1\}, \dots, \{n-1\}$
- `DisjointSet(int n)` Κατασκευαστής. Δημιουργεί  
 $n$  μονοσύνοδα
- `find(int x)` Επιστρέφει το "σύνοδο" σε  
 οποιο ανήκει το συγκεκίνο  $x$
- `union(int x, int y)` Συνίνωση των "συνόδων"  $x$   
 και  $y$ .
- Κάθε σύνοδο προσδιορίζεται από ένα μέδος του,  
 τον εκπρέποντέ του (representative)
- Τα μέδοντα συνόδων είναι συντόνων

Travelling: To oīgaoīpoin's diūd or Rosavaia vai Iīvāidis

To 1800:

- $$D = \{ \{ \text{London} \}, \{ \text{Birmingham} \}, \{ \text{Liverpool} \}, \\ \{ \text{Manchester} \}, \{ \text{Edinburg} \}, \\ \{ \text{Belfast} \}, \{ \text{Dublin} \} \}$$
- }

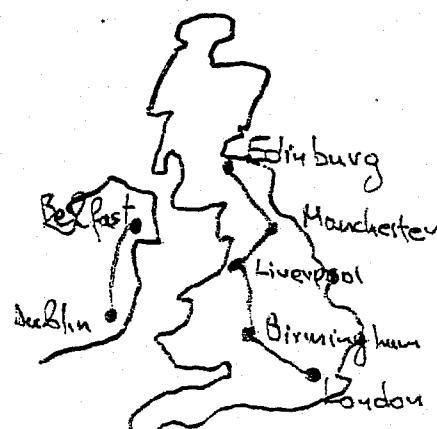


To 1830:

- $$D = \{ \{ \text{London} \}, \{ \text{Birmingham} \}, \\ \{ \text{Liverpool}, \text{Manchester} \}, \\ \{ \text{Edinburg} \}, \{ \text{Dublin} \}, \\ \{ \text{Belfast} \} \}$$
- }

Empire:

- $$D = \{ \{ \text{London}, \text{Birmingham}, \text{Liverpool}, \\ \text{Manchester}, \text{Edinburg} \}, \\ \{ \text{Belfast}, \text{Dublin} \} \}$$
- }



## Anaparastasis

ε) Η.3 σιγουρα:

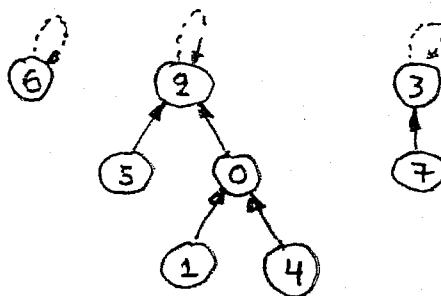
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 2 | 2 | 2 | 3 | 2 | 2 | 6 | 3 |

$$D = \{ \{0, 1, \underline{2}, 4, 5\}, \\ \{3, 7\}, \\ \{6\} \}$$

- Αποδεικνύεται ο αντίθετος των συνόλων
- $\text{find}(x) \rightarrow O(1)$
- $\text{union}(x, y) \rightarrow O(n)$
- Απλή υλοποίηση.
- Μια αναλογία από  $n-1$  συνόλων απαιτεί  $O(n^2)$  χρόνο

• Αναφορικό μοντέλο

- με δίνεται



$$\mathcal{D} = \left\{ \begin{array}{l} \{0, 1, \underline{2}, 4, 5\} \\ \{3, 7\} \\ \{6\} \end{array} \right\}$$

- με διάνυσμα

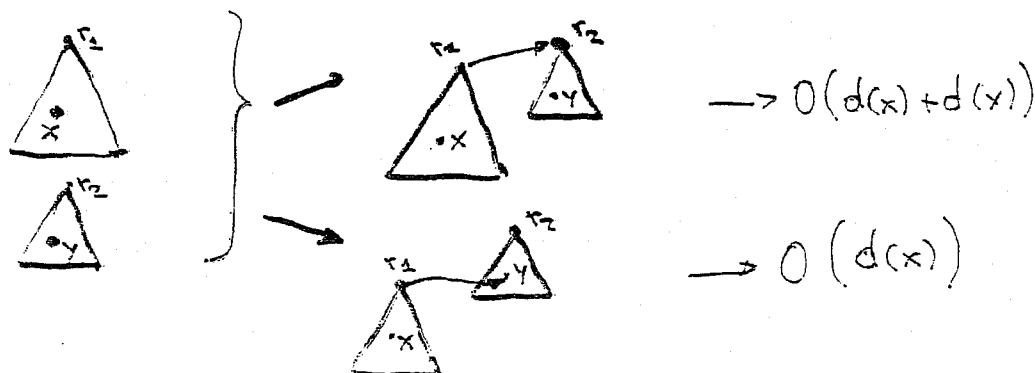
|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 2 | 0 | 2 | 3 | 0 | 2 | 6 | 3 |

- οργανωμένο οι δέρματα στους πίνακες είναι ο αντιπρόσωπος του συνόλου.

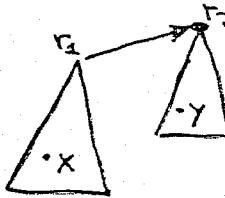
- $\text{find}(x) \rightarrow O(d(x))$

$d(x)$  = βαθος του  $x$   
στο δέρμα τη στοιχία  
ανήκει.

- $\text{union}(x, y) \rightarrow O(d(x) + d(y))$

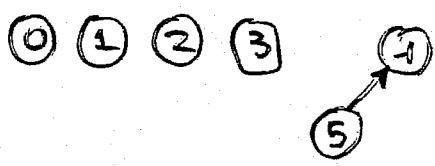


Παράδειγμα :

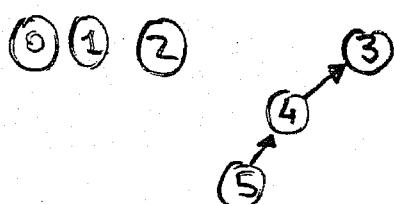


① ② ③ ④ ⑤

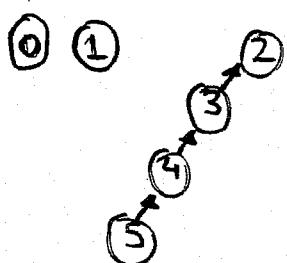
↓ union (5,4)



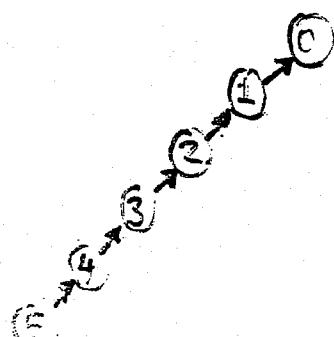
↓ union (5,3)



↓ union (5,2)



↓ union (5,1),  
union (5,0)



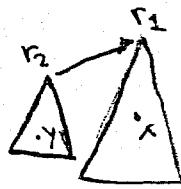
- Το ύψος των δένδρων μπορεί να είναι  $O(n)$

- $n-1$  διαδοχικές συνώνυμες απαιτούν  $O(n^2)$  χρόνου

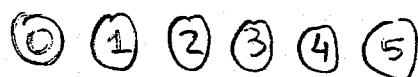
⇒ Στο παρόντα,  
αντιπρόσωπος της ένωσης  
γίνεται ο αντιπρόσωπος του  
"μηρόζερον" συνόλου.

Υπάρχει δεξιωματικός αντι-  
πρόσωπος γιατί ο αντιπρό-  
σωπος του "μηρόζερον"  
συνόλου?

Параллель (сравнение)



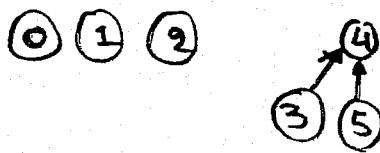
$$|T_{r_1}| \geq |T_{r_2}|$$



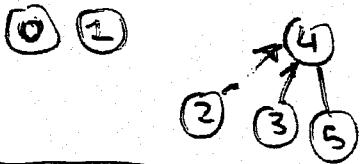
$\Downarrow \text{union}(5, 4)$



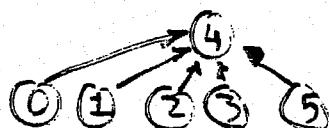
$\Downarrow \text{union}(5, 3)$



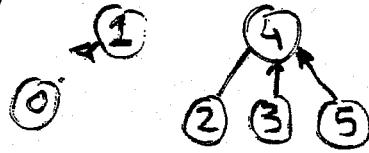
$\Downarrow \text{union}(5, 2)$



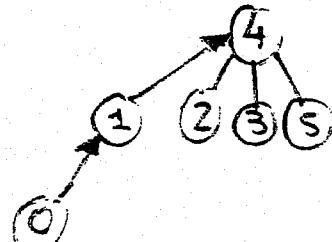
$\Downarrow \text{union}(5, 1)$   
 $\Downarrow \text{union}(5, 0)$



$\Downarrow \text{union}(0, 1)$



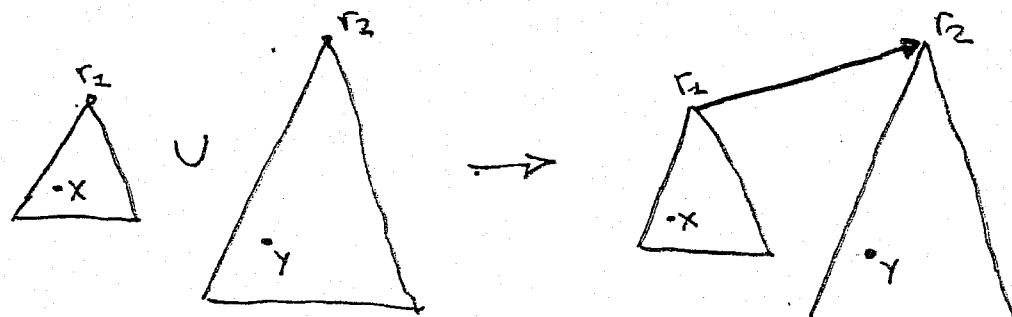
$\Downarrow \text{union}(5, 0)$



Задание: Что произойдет если вverts и edges поменять местами?

Karónas bávaoríneus énwas [weighted union rule]

- Lípes omónias = apólytos taktikón zoi
- O anátipróσompos zou elaphrúzou συνίδou δυσχειάζεται με αυτόν zou bávúzou συνόλou



$$|T_{r_1}| \leq |T_{r_2}|$$

Λήμμα: Γia naðe stoixhío  $x$ , zou píndos zan atoðouna zou  $w(x)$  enai zetaxízou  $2^{h(x)}$ , orou  $h(x)$  zou náðou zou oso gáðos zan zetaxízou  $\pi$  exi bávúzou hixhi zetaxízou

Απόδικη: (Me exaxwú zou náðou zan δáðrou)

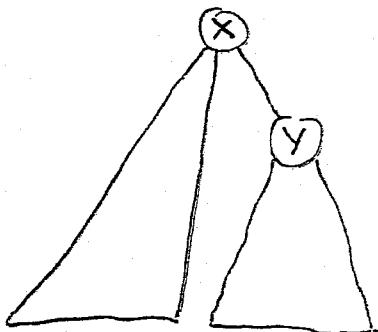
Στοιχ:  $h(x) = \dots \Rightarrow w(x) \geq 2^0 = 1$  toxízou ✓

Επαγγελμ. Λήμμα: Eozou cu tia naðe tópos  $h(x) < k$  iðxou:

$$w(x) \geq 2^{h(x)}$$

ta díktoumi oso iðxou na  $h(x) = k$ . →

Εστω  $X$  αριθμός με  $h(x) = k > 0$  και  
 $Y$  οι παροχές του ενώπιου για τον  $X$ , οπου  $h(y) \leq h(x)$



- $w'(x) \equiv$  βάρος του  $x$  πριν την  $y$
- $w(y) = w'(y)$
- $w'(x) \geq w'(y)$  [weighted union]

Θελαγχεί να δειχνύει ότι:  $2^{h(x)} \leq w(x)$

$$\begin{aligned}
 2^{h(x)} &= 2^{\max(1+h(y), h'(x))} & h'(x) &= \text{βάρος πριν} \\
 &= \max\left(2^{h(y)+1}, 2^{h'(x)}\right) & \text{την } y \\
 &\leq \max(2w(y), w'(x)) \\
 &\leq \max(w(x), w(x)) \\
 &= w(x)
 \end{aligned}$$

Apa

$$2^{h(x)} \leq w(x)$$

□

**Θεώρημα** Το υψος του δενδρου ενός συνόλου  $n$  στοιχίων αυτού είναι το πολ.  $\log n$

**Απόδειξη:** • Εσώ  $x$  το στοιχίο στη γένη  
του δενδρου

$$\bullet \quad w(x) \geq 2^{h(x)} \quad \text{Λήγεια} \quad \textcircled{1}$$

$$\bullet \quad n = w(x) \quad \text{γιατίδια} \quad \textcircled{2}$$

$$\textcircled{1} \textcircled{2} \Rightarrow n \geq 2^{h(x)} \Rightarrow h(x) \leq \log_2 n$$

□

**Θεώρημα** Μια αναμεταγράψιν ακολουθία από  $n-1$  ενώσεις και  $m$  αναζητήσεις παρίσταται  $O(n \log n + m \log n)$  χρόνο, εάν ούτων χρησιμοποιούνται ο κανόνας βεβαρημένης επερτών

**Απόδειξη:**  $n-1$  ενώσεις:  $\textcircled{1} \quad O(n \log n) \quad \left\{ \Rightarrow O(n \log n + m \log n) \right.$   
 $m$  αναζητήσεις:  $\rightarrow O(m \log n) \quad \left. \right\}$

\*-Ευκίνηση του "μερίδιου" του καθι. συρόμενου  
= απαρτίζει "find"