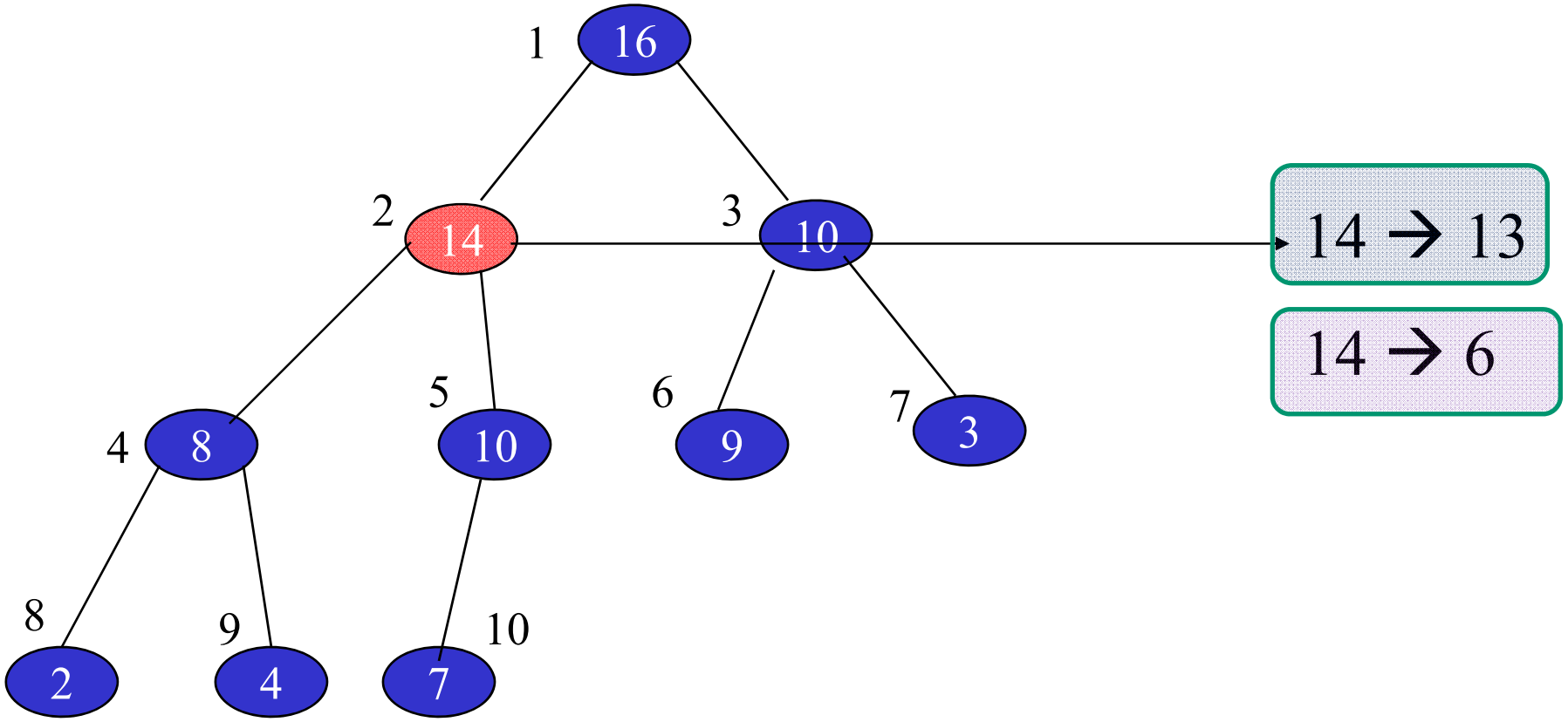
Εφαρμογές σωρού

- Max heap: χρονοπρογραμματισμός
- Min heap: simulation
- Ταξινόμηση με τη βοήθεια σωρού

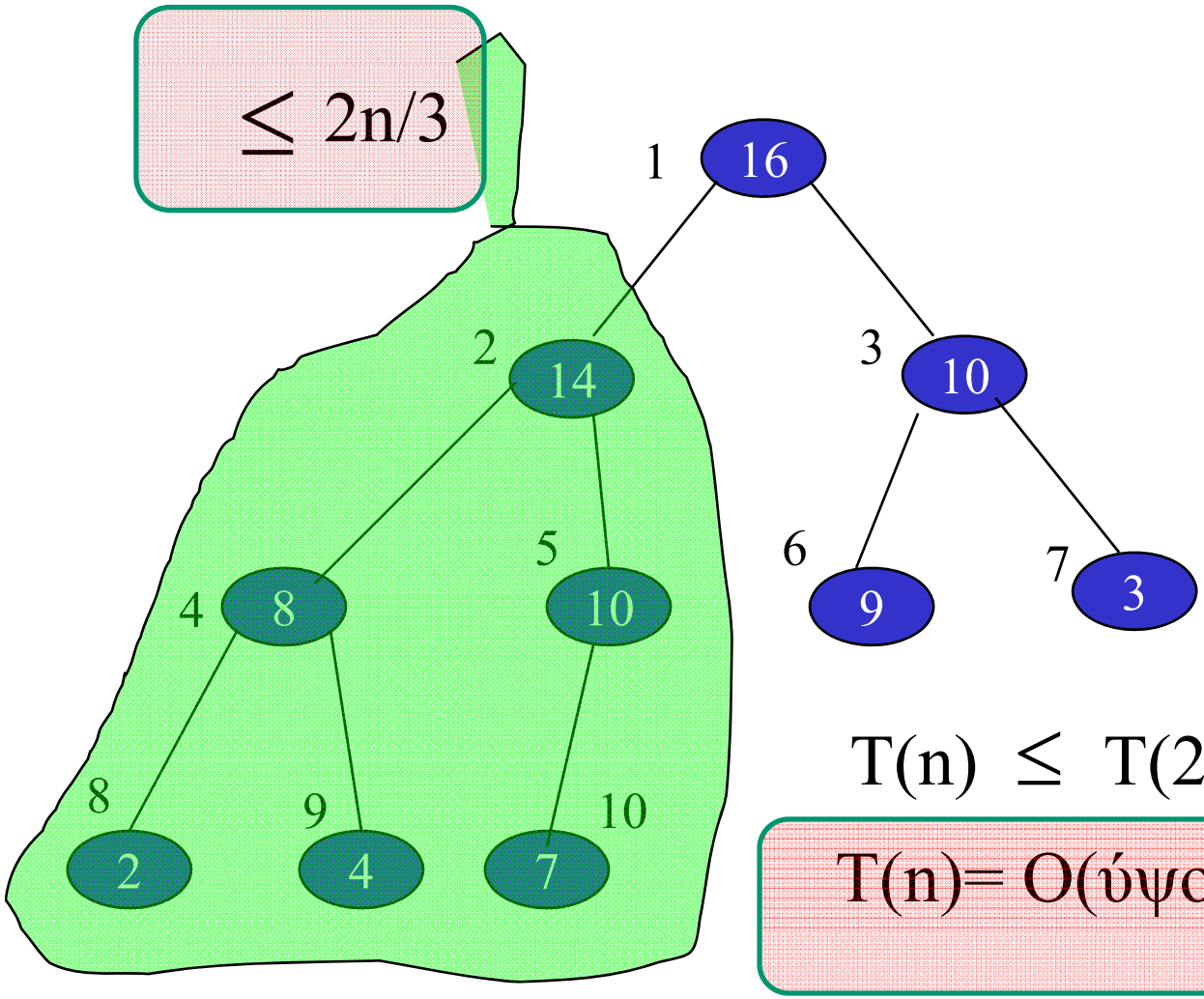
Εφαρμογές σωρού



Διατήρηση σωρού

```
MAX-HEAPIFY (A, i)
f  ←  left [i]
r  ←  right (i)
if  f ≤ heap-size [A] and A [f] >A [i]
    then largest ← f
    else largest ← i
if r ≤ heap-size [A] and A [r]>A [largest]
    then largest ← r
if largest ≠ i
    then exchange A [i] ↔ A [largest]
        MAX-HEAPIFY [A, largest]
```

Διατήρηση σωρού: πολυπλοκότητα



$$T(n) \leq T(2n/3) + \Theta(1)$$

$T(n) = O(\text{ύψος}) = O(\log n)$

Κατασκευή σωρού on - line

Input: τα στοιχεία καταφθάνουν το ένα μετά το άλλο: a[i], $1 \leq i \leq n$

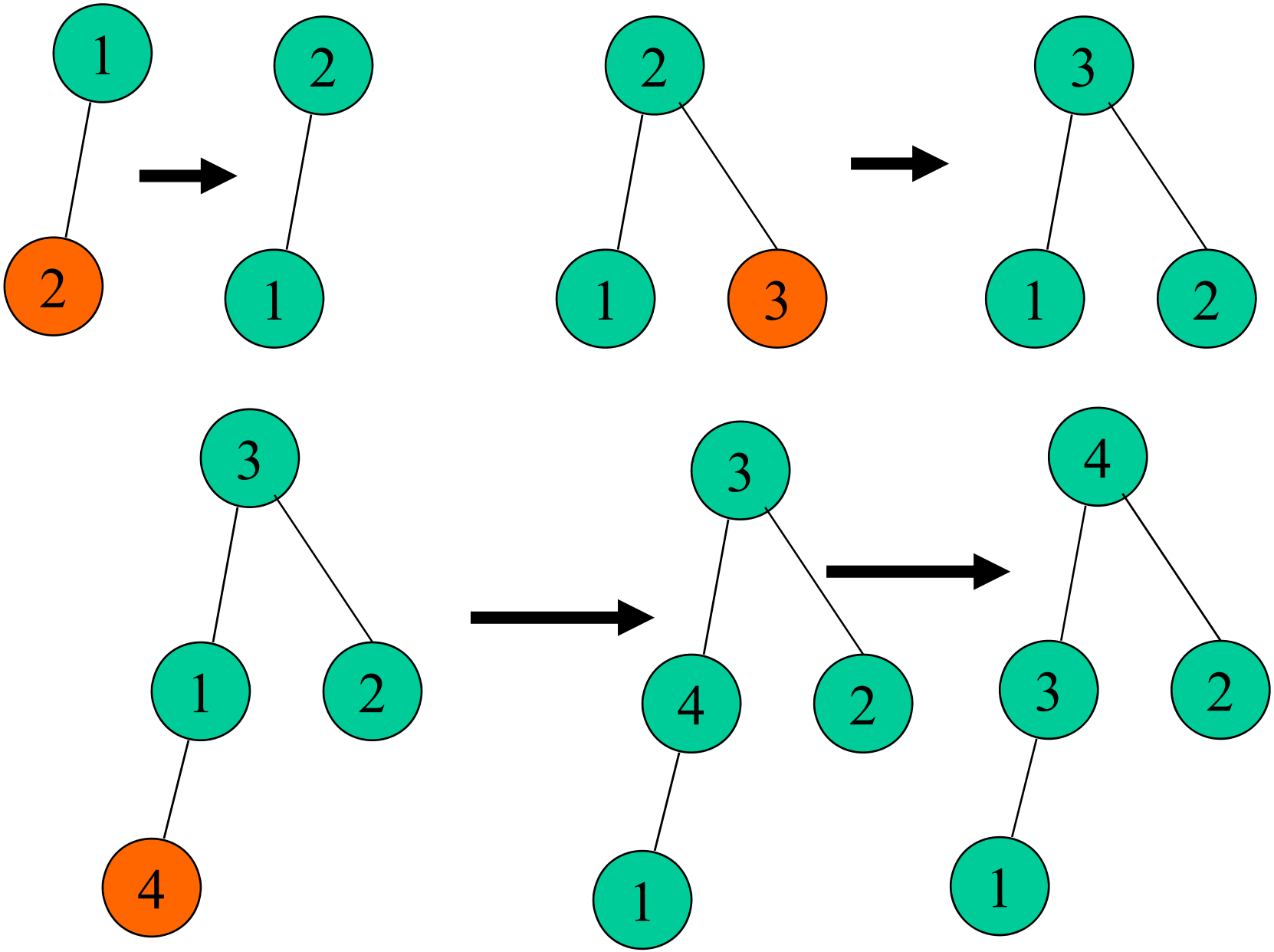
(θεωρούμε ότι τα στοιχεία δεν είναι διαθέσιμα σε ένα πίνακα)

Output: Δομή σωρού

Παράδειγμα: Ακολουθία στοιχείων

1 2 3 4 . . .

Παράδειγμα (κατασκευή σωρού on – line)



Κατασκευή Σωρού / on - line

```
ConstructHeap;
```

```
nheap := 0;
for i := 1 to n do
    insert(a[i]);
end; ****
```

Κατασκευή Σωρού on – line / Πολυπλοκότητα

insert(a[i]) → O(log i)

$T(n)= \log n + \log (n-1) + \ldots + = \log (n!)$

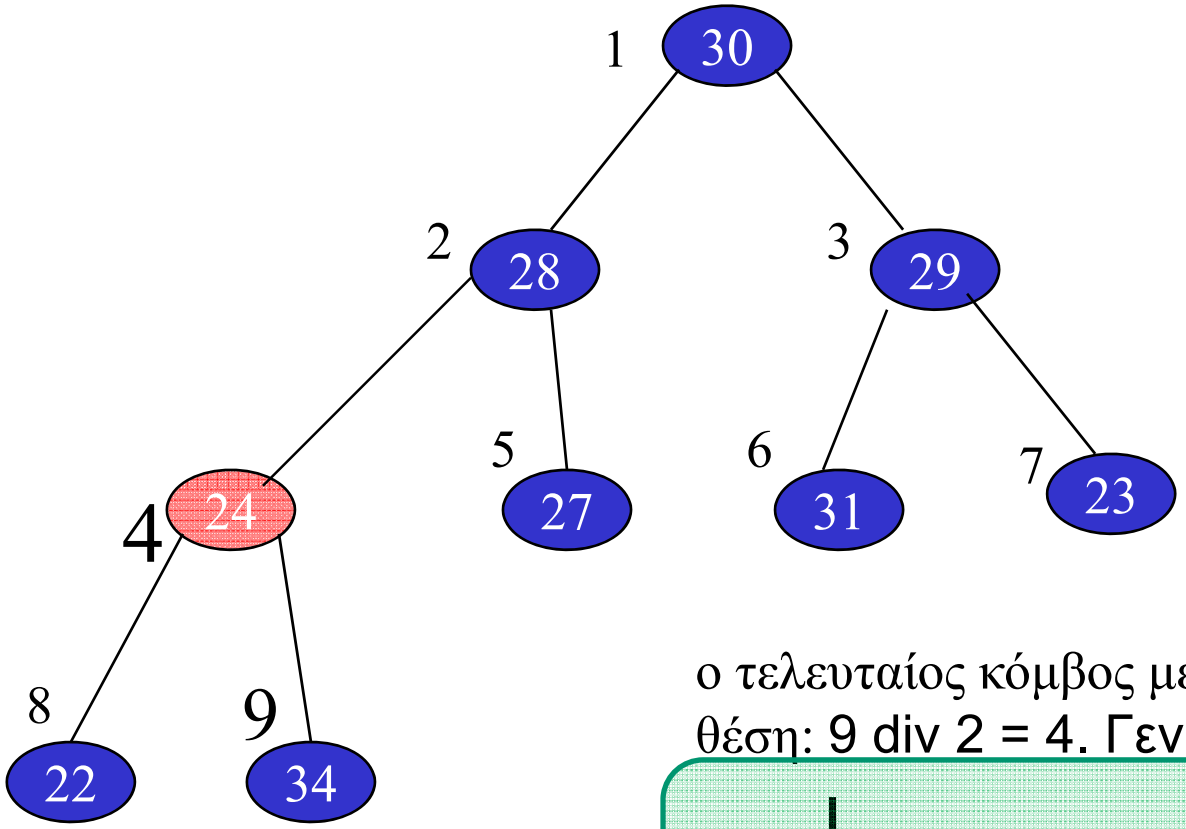
$$n! \leq n^n$$

$$n! \geq (n/2)^{n/2}$$

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

Κατασκευή σωρού σε O(n)

Τα στοιχεία είναι όλα διαθέσιμα στο πίνακα A

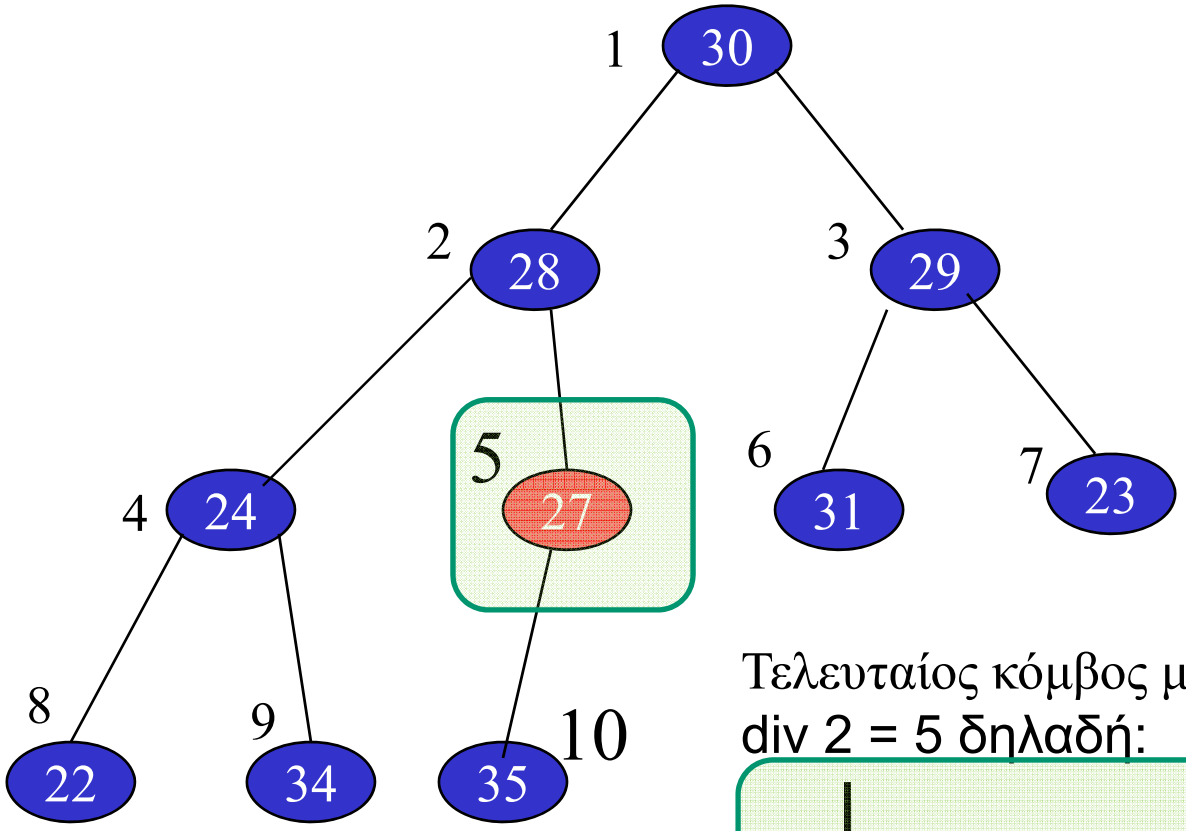


ο τελευταίος κόμβος με παιδί είναι στη θέση: 9 div 2 = 4. Γενικότερα στη θέση:

$\lfloor$ length [A]/2 $\rfloor$

Κατασκευή σωρού σε O(n)

Τα στοιχεία είναι όλα διαθέσιμα στο πίνακα A

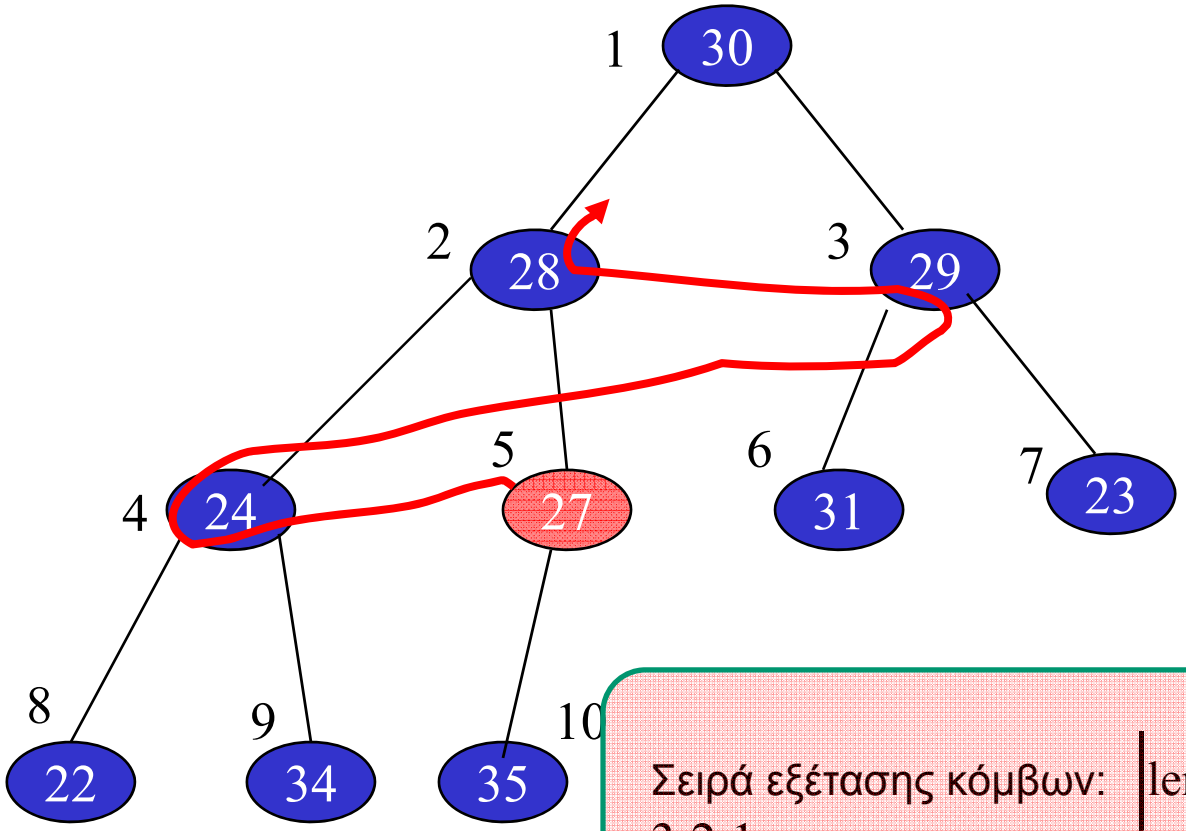


Τελευταίος κόμβος με παιδί είναι στη θέση 10
div 2 = 5 δηλαδή:

$\lfloor$ length [A]/2 $\rfloor$

Κατασκευή σωρού σε O(n)

Τα στοιχεία είναι όλα διαθέσιμα στο πίνακα A



Σειρά εξέτασης κόμβων: $\lfloor \text{length}[A]/2 \rfloor, \lfloor \text{length}[A]/2 \rfloor - 1, \ldots, 3, 2, 1$

Κατασκευή σωρού σε O(n)

Τα στοιχεία είναι όλα διαθέσιμα στο πίνακα A

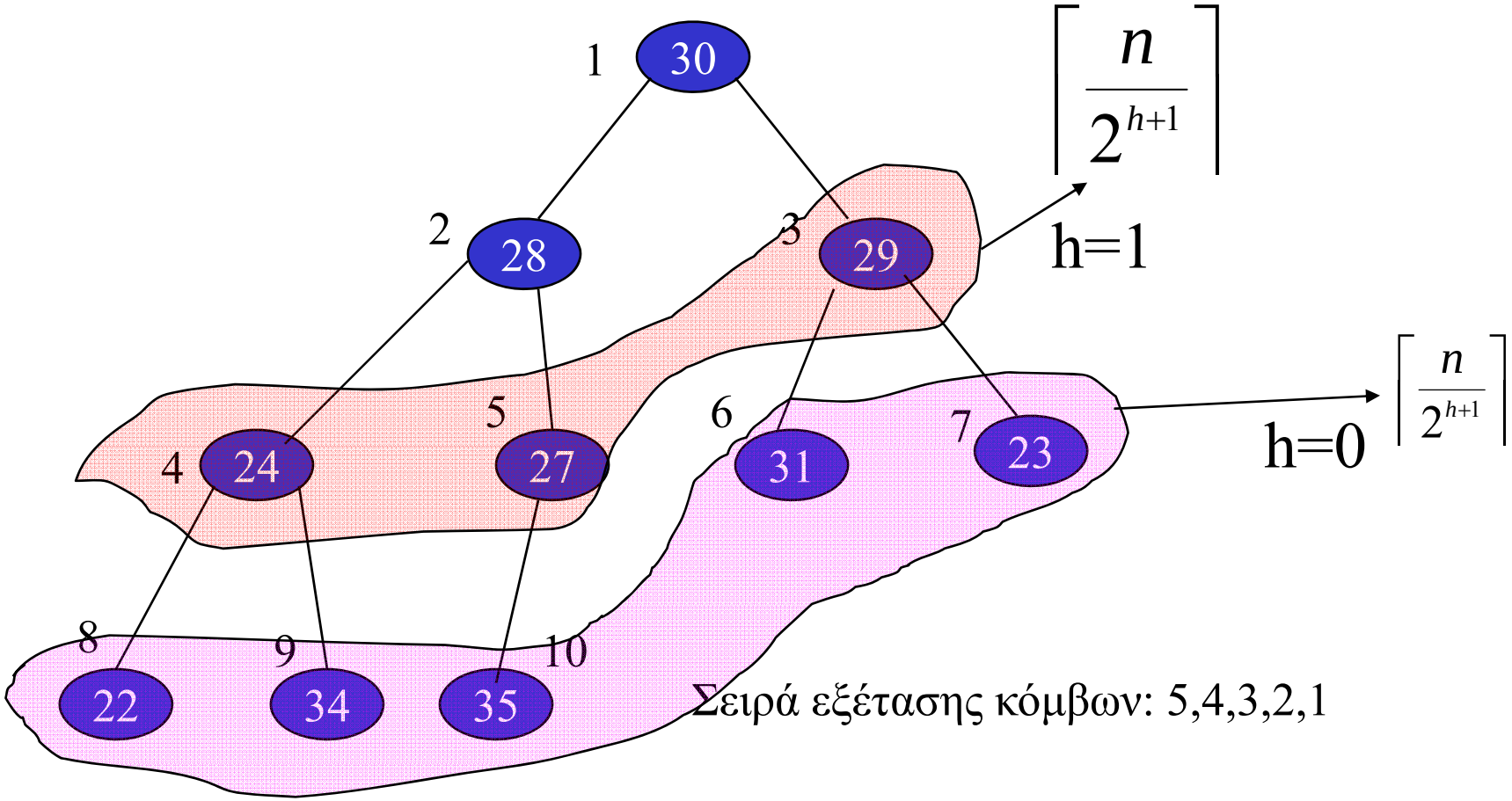
BUILD-MAX-HEAP (A)

heap-size [A] ← length [A]

for i ← ⌊length [A]/2⌋ downto 1 do

MAX-HEAPIFY (A, i)

Κατασκευή σωρού σε O(n)



Υπενθύμιση

Για $x \neq 1$

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

Για $|x| < 1$

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1 - x}$$

$$\sum_{k=0}^{\infty} k x^k = \frac{x}{(1-x)^2}$$

Κατασκευή σωρού σε O(n)

Το πολύ $\left\lceil \frac{n}{2^{h+1}} \right\rceil$ κόμβοι ύψους h

$$T(n) = \sum_{h=1}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = \ldots O(n)$$

- Υπενθύμιση: $\sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{\frac{1}{2}}{\left(1 - \frac{1}{2}\right)^2} = 2$

Αλγόριθμος ταξινόμησης με σωρό - Heapsort

Input: πίνακας a[i], $1 \le i \le n$
Output: πίνακας $a[i_j], a[i_j] \le a[i_{j+1}]$

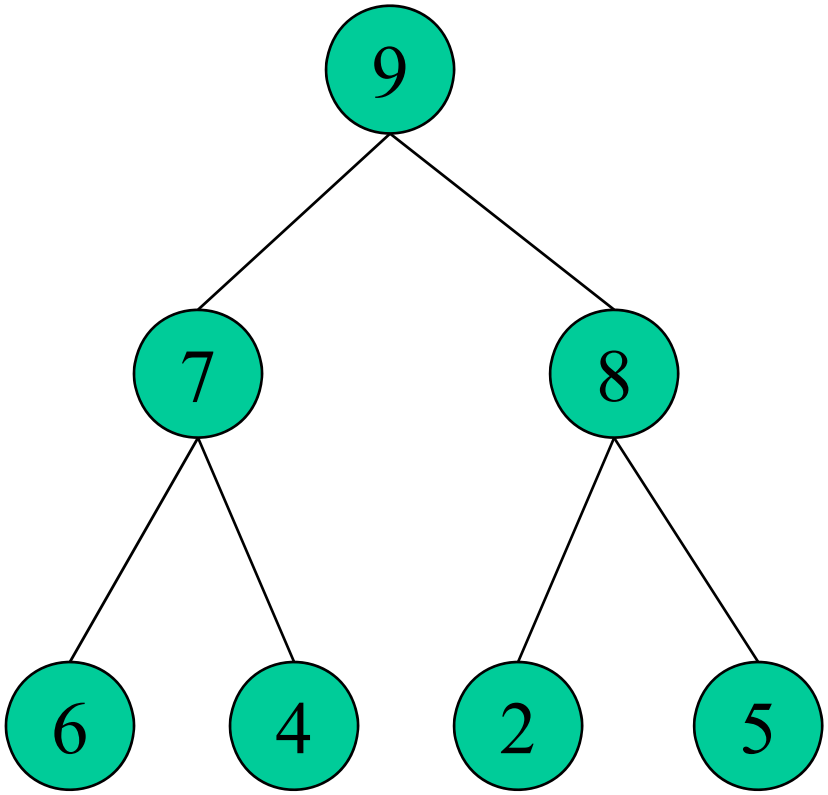
Βήμα 1: Διαμόρφωση του πίνακα σε σωρό

Βήμα 2: Επανέλαβε

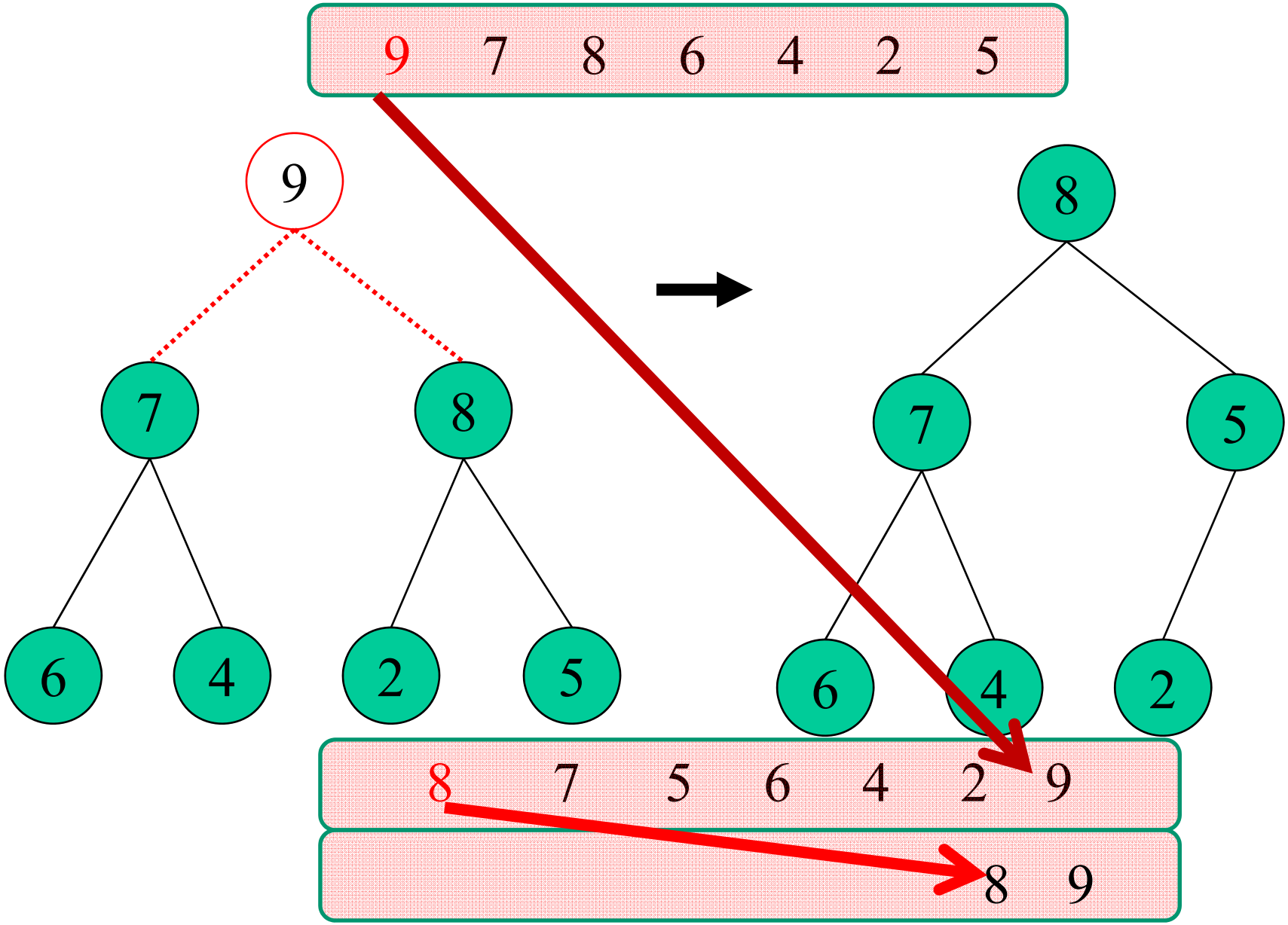
- Πάρε το μεγαλύτερο στοιχείο
- Αφαίρεσέ το από το σωρό και αναδιάρθρωσε το σωρό
- Τοποθέτησέ το στην ελεύθερη θέση του τμήματος του πίνακα που δεν είναι ταξινομημένο ****

Παράδειγμα ταξινόμησης με σωρό - Heapsort

6 9 2 7 4 5 8



Παράδειγμα ταξινόμησης με σωρό (ΔΙΑΓΡΑΦΗ 1ου στοιχείου)



Heapsort

```
HeapSort()

    nheap = 0;
    for i = 0 το n-1
        Insert(a[i])

    for i = n-1 το 0
        v = Maximum()
        Delete()
        a[i] = v
```

****

Αλγόριθμος Heapsort / Πολυπλοκότητα

$$\sum_{k=1}^{n} \log k = \log\left(\prod_{k=1}^{n} k\right) = \log(n!)$$

Βήμα 1: Θ(n log n) ή O(n)

Βήμα 2: Θ(n log n)

Συνολικά: Θ(n log n)