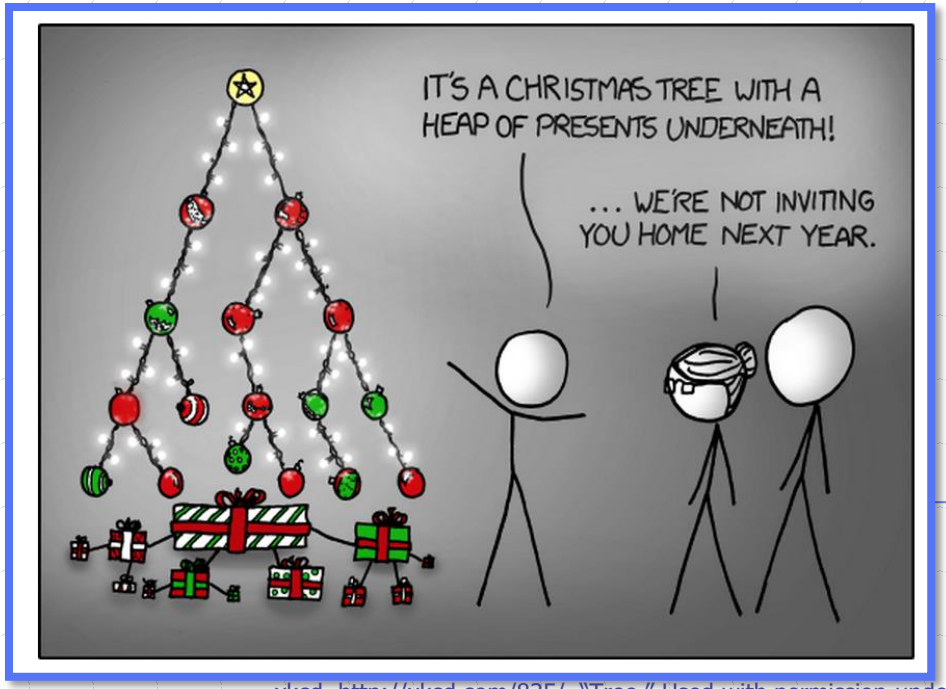
Παρουσίαση για χρήση με το σύγγραμμα, Αλγόριθμοι Σχεδίαση και Εφαρμογές, των M. T. Goodrich and R. Tamassia, Wiley, 2015 (στα ελληνικά από εκδόσεις Μ. Γκιούρδας)

Σωροί



xkcd. http://xkcd.com/835/. "Tree." Used with permission under Creative Commons 2.5 License.

Θυμηθείτε τις λειτουργίες ουράς προτεραιότητας

- Μία ουρά προτεραιότητας αποθηκεύει μία συλλογή εγγραφών
- Κάθε εγγραφή είναι ένα ζεύγος (κλειδί, τιμή)
- Κυρίες μέθοδοι:
  - insert(k, v)
    εισάγει μία εγγραφή με κλειδί k και τιμή v
  - removeMin()
    αφαιρεί και επιστρέφει την εγγραφή με το μικρότερο κλειδί ή null εάν η ουρά προτεραιότητας είναι κενή
- Επιπλέον μέθοδοι:
  - min()
    επιστρέφει χωρίς να αφαιρεί την εγγραφή με το μικρότερο κλειδί ή null εάν η ουρά προτεραιότητας είναι κενή
  - size()
  - isEmpty()
- Εφαρμογή:
  - Πελάτες πτήσεων εν αναμονή
  - Δημοπρασίες
  - Χρηματιστήρια

Θυμηθείτε την ταξινόμηση με ουρά προτεραιότητας

- Μπορούμε να χρησιμοποιήσουμε μία ουρά προτεραιότητας για να ταξινομήσουμε μία λίστα συγκρίσιμων στοιχείων
  1. Είσοδος των στοιχείων ένα προς ένα με μία σειρά από λειτουργίες insert
  2. Αφαίρεση των στοιχείων με μία σειρά από λειτουργίες removeMin
- Ο χρόνος εκτέλεσης εξαρτάται από την υλοποίηση της ουράς προτεραιότητας.
  - Μη ταξινομημένη ακολουθία, ταξινόμηση με επιλογή: $O(n^2)$
  - Ταξινομημένη ακολουθία, ταξινόμηση με εισαγωγή: $O(n^2)$
- Μπορούμε καλύτερα;

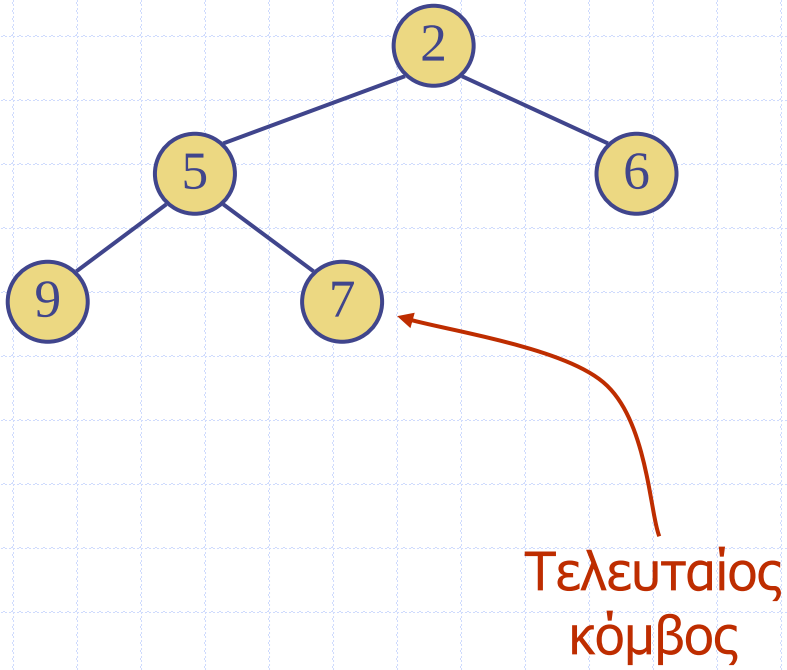
Algorithm PQ-Sort(C, P):
Input: An n-element array, C, index from 1 to n, and a priority queue P that compares keys, which are elements of C, using a total order relation
Output: The array C sorted by the total order relation

for $i \leftarrow 1$ **to** n **do**
  $e \leftarrow C[i]$
  P.insert(e, e) // the key is the element itself
for $i \leftarrow 1$ **to** n **do**
  $e \leftarrow P$.removeMin$()$ // remove a smallest element from P
  $C[i] \leftarrow e$

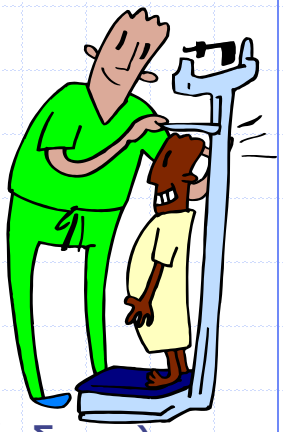
Σωροί

- Ένας σωρός είναι ένα δυαδικό δένδρο που αποθηκεύει κλειδιά στους κόμβους του και ικανοποιεί τις εξής ιδιότητες:
- Ιδιότητα σειράς σωρού: για κάθε εσωτερικό κόμβο v πέραν αυτού της ρίζας, $key(v) \geq key(parent(v))$
- Πλήρες δυαδικό δένδρο: με h το ύψος της σωρού
  - για $i = 0, \ldots, h-1$, υπάρχουν 2^i κόμβοι στο βάθος i
  - στο βάθος $h-1$, οι εσωτερικοί κόμβοι είναι αριστερά των εξωτερικών κόμβων

- Ο τελευταίος κόμβος ενός σωρού είναι ο δεξιότερος κόμβος στο μέγιστο βάθος

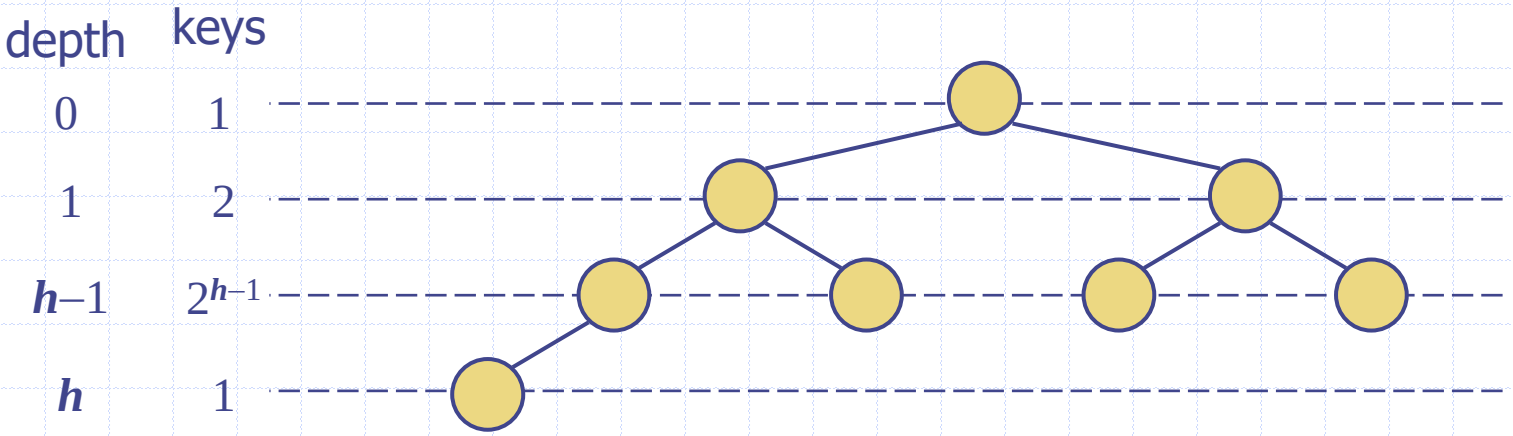


Ύψος ενός σωρού



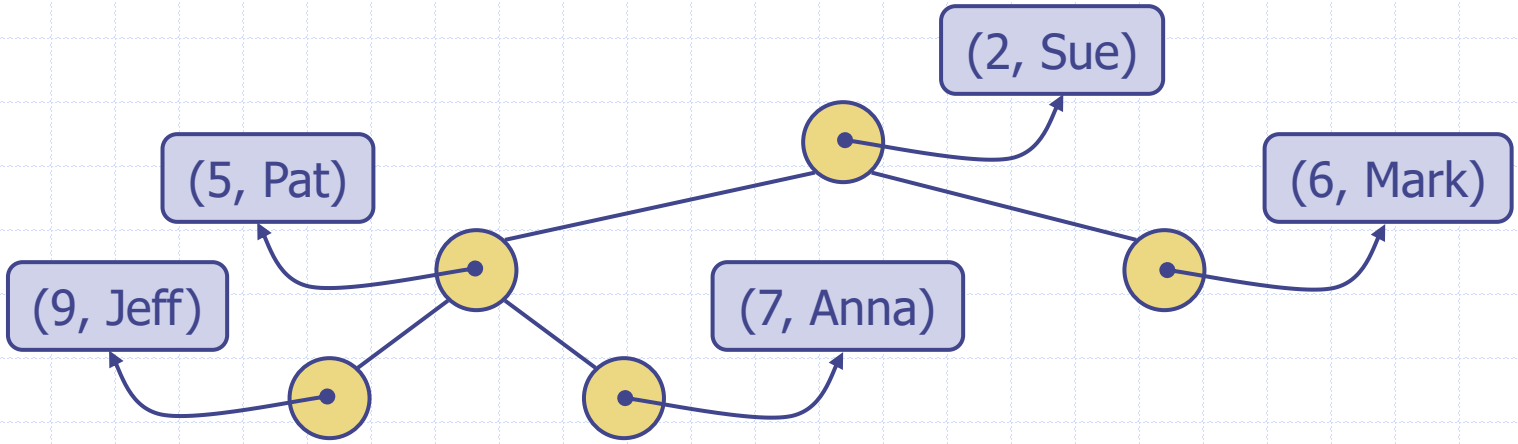
- Theorem: Ένας σωρός με αποθηκευμένα n κλειδιά έχει ύψος $O(\log n)$

  Απόδειξη: (εφαρμόζουμε την ιδιότητα του πλήρους δυαδικού δένδρου)
  - Με h το ύψος ενός σωρού με αποθηκευμένα n κλειδιά
  - Επειδή υπάρχουν 2^i κλειδιά στο βάθος $i = 0, \ldots, h-1$ και τουλάχιστον ένα κλειδί στο βάθος h, έχουμε $n \geq 1 + 2 + 4 + \ldots + 2^{h-1} + 1$
  - Έτσι, $n \geq 2^h$, δλδ., $h \leq \log n$



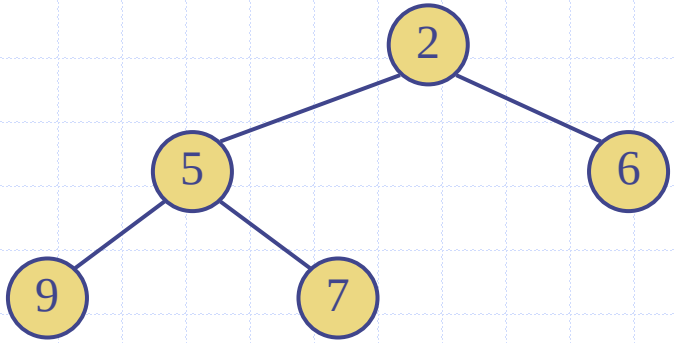
Σωροί και Ουρές προτεραιότητας

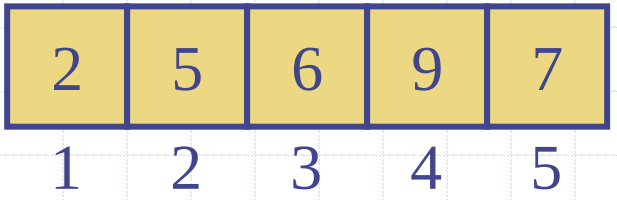
- Μπορούμε να χρησιμοποιήσουμε έναν σωρό για να υλοποιήσουμε μία ουρά προτεραιότητας
- Αποθηκεύουμε ένα αντικείμενο (κλειδί, στοιχείο) σε κάθε εσωτερικό κόμβο
- Παρακολουθούμε την θέση του τελευταίου κόμβου



Υλοποίηση σωρού με πίνακα

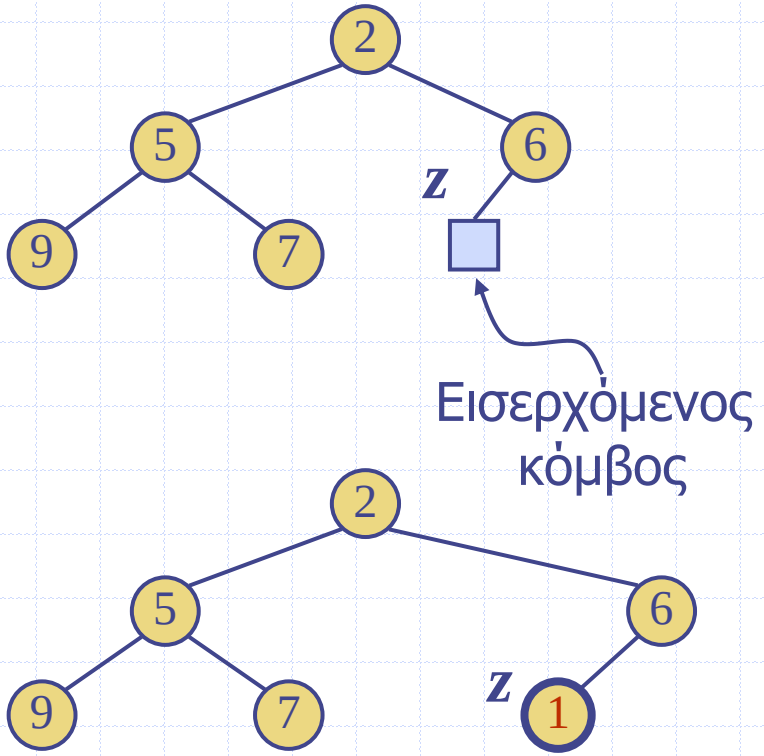
- Μπορούμε να αναπαραστήσουμε έναν σωρό με n κλειδιά με έναν πίνακα μήκους n
- Για κόμβο θέσης i
  - Το αριστερό παιδί είναι στην θέση $2i$
  - Το δεξιό παιδί είναι στην θέση $2i + 1$
- Οι σύνδεσμοι μεταξύ των κόμβων δεν αποθηκεύονται
- Η λειτουργία add ισοδυναμεί σε προσθήκη στην θέση $n + 1$
- Η λειτουργία remove_min ισοδυναμεί με αφαίρεση από τη θέση n
- Λειτουργεί ως επί τόπου ταξινόμηση με σωρό





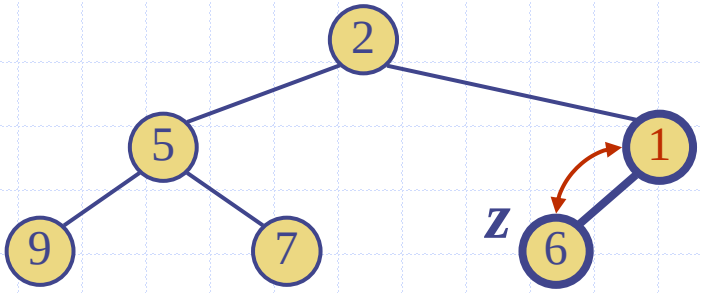
Εισαγωγή σε σωρό

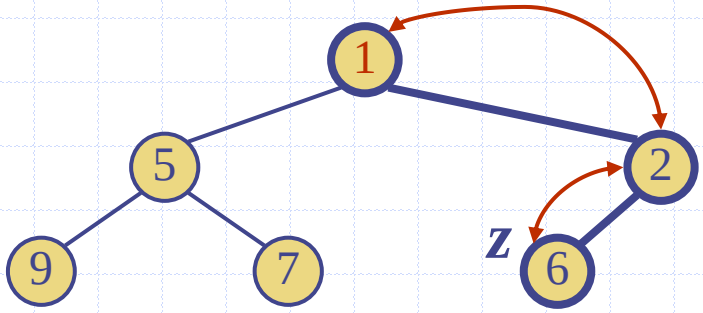
- Η μέθοδος insertItem της αφηρημένης δομής δεδομένων ουράς προτεραιότητας αντιστοιχεί στην εισαγωγή του κλειδιού k στον σωρό.
- Ο αλγόριθμος εισαγωγής αποτελείται από τρία βήματα
  - Εύρεση του κόμβου z (ο νέος τελευταίος κόμβος)
  - Αποθήκευση του k στο z
  - Αποκατάσταση της διάταξης σωρού (βλέπε την συνέχεια)



Upheap

- Μετά την είσοδο του νέου κλειδιού k, η ιδιότητα της διάταξης σωρού μπορεί να έχει παραβιαστεί
- Ο αλγόριθμος upheap αποκαθιστά την ιδιότητα διάταξης σωρού ανταλλάσσοντας το k ακολουθώντας έναν μονοπάτι προς τα πάνω.
- Ο Upheap σταματά όταν το k φτάσει στη ρίζα ή όταν φτάσει σε έναν κόμβο που ο γονέας του έχει κλειδί μικρότερο ή ίσο με το k
- Καθότι ένα σωρός έχει ύψος $O(\log n)$, ο upheap είναι $O(\log n)$





Ψευδοκώδικας εισαγωγής

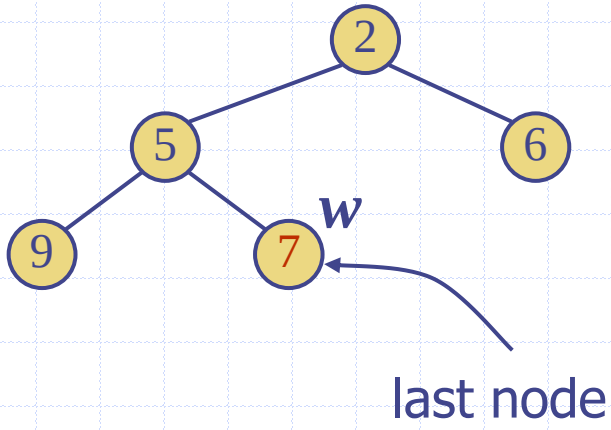
- Υποθέτουμε ότι έχουμε υλοποίηση με πίνακα.

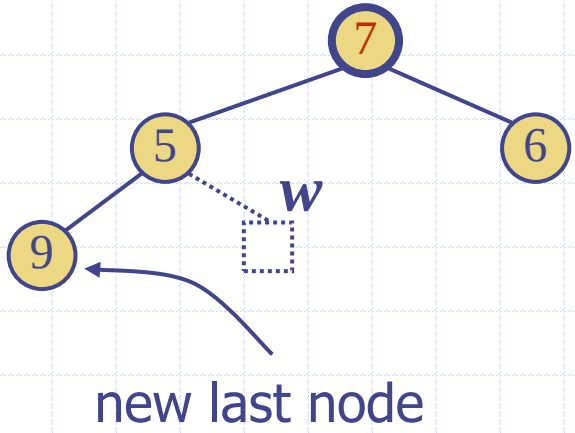
Algorithm HeapInsert(k, e):
Input: A key-element pair
Output: An update of the array, A, of n elements, for a heap, to add (k, e)

$n \leftarrow n + 1$
$A[n] \leftarrow (k, e)$
$i \leftarrow n$
while $i > 1$ **and** $A[\lfloor i/2 \rfloor] > A[i]$ **do**
    Swap $A[\lfloor i/2 \rfloor]$ and $A[i]$
    $i \leftarrow \lfloor i/2 \rfloor$

Αφαίρεση από σωρό

- Η μέθοδος removeMin της αφηρημένης δομής δεδομένων ουράς προτεραιότητας αντιστοιχεί στην αφαίρεση του κλειδιού στην ρίζα του σωρού.
- Ο αλγόριθμος αφαίρεσης αποτελείται από τρία βήματα
  - Αντικατάσταση του κλειδιού στην ρίζα με τον τελευταίο κόμβο w
  - Αφαίρεση του w
  - Αποκατάσταση της διάταξης σωρού (βλέπε την συνέχεια)





Downheap

- Μετά την αντικατάσταση του κλειδιού στη ρίζα με το κλειδί k του τελευταίου κόμβου, η ιδιότητα της διάταξης σωρού μπορεί να έχει παραβιαστεί
- Ο αλγόριθμος downheap αποκαθιστά την ιδιότητα διάταξης σωρού ανταλλάσσοντας το κλειδί k ακολουθώντας ένα καθοδικό μονοπάτι από την ρίζα
- Ο downheap σταματά όταν το κλειδί k φτάσει σε ένα φύλλο ή έναν κόμβο που τα παιδία του έχουν κλειδιά μεγαλύτερα ή ίσα του k
- Καθότι ένα σωρός έχει ύψος $O(\log n)$, ο downheap είναι $O(\log n)$

Ψευδό-κώδικας RemoveMin

- Υποθέτουμε ότι έχουμε υλοποίηση με πίνακα.

Algorithm HeapRemoveMin():
Input: None
Output: An update of the array, A, of n elements, for a heap, to remove and return an item with smallest key

```
temp ← A[1]
A[1] ← A[n]
n ← n − 1
i ← 1
while i < n do
    if 2i + 1 ≤ n then   // this node has two internal children
        if A[i] ≤ A[2i] and A[i] ≤ A[2i + 1] then
            return temp        // we have restored the heap-order property
        else
            Let j be the index of the smaller of A[2i] and A[2i + 1]
            Swap A[i] and A[j]
            i ← j
    else   // this node has zero or one internal child
        if 2i ≤ n then   // this node has one internal child (the last node)
            if A[i] > A[2i] then
                Swap A[i] and A[2i]
        return temp        // we have restored the heap-order property
return temp        // we reached the last node or an external node
```

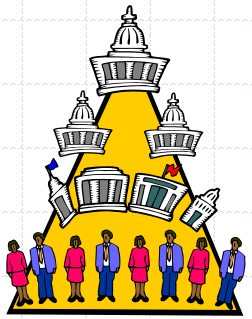
Απόδοση του σωρού

- Ένας σωρός έχει την εξής απόδοση για τις λειτουργίες τις ουράς προτεραιότητας.

Operation	Time
insert	$O(\log n)$
removeMin	$O(\log n)$

- Η παραπάνω ανάλυση βασίζεται στα:
  - Το ύψος ενός σωρού T είναι O(log n), καθώς ο T είναι πλήρης.
  - Στην χειρότερη περίπτωση, ο up-heap και ο down-heap απαιτούν χρόνο ανάλογο με το ύψος του T.
  - Η εύρεση της θέσης εισαγωγής και η removeMin απαιτούν σταθερό χρόνο.
  - Ο σωρός T έχει n εσωτερικούς κόμβους, ο καθένας αποθηκεύει μία αναφορά σε κάποιο κλειδί και μία αναφορά σε κάποιο στοιχείο.

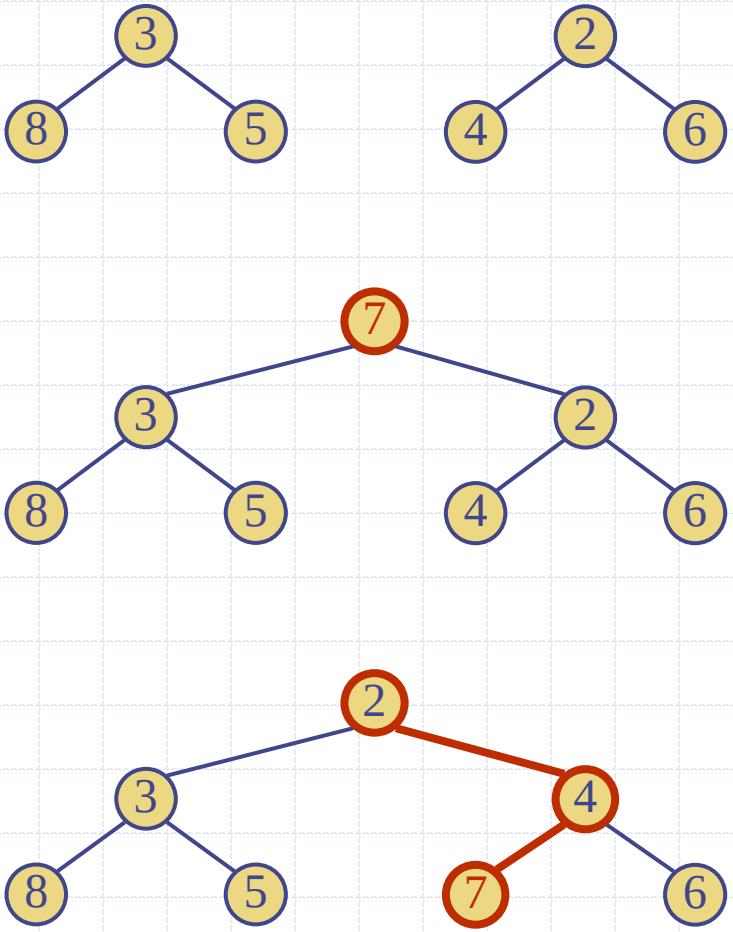
Ταξινόμηση σωρού



- Θεωρείστε μία ουρά προτεραιότητας με n αντικείμενα υλοποιημένη με σωρό
  - Ο χώρος που χρησιμοποιείται είναι $O(n)$
  - Οι μέθοδοι insert και removeMin είναι $O(\log n)$
  - Οι μέθοδοι size, isEmpty, και min είναι $O(1)$
- Χρησιμοποιώντας μία ουρά προτεραιότητας βάση σωρού μπορούμε να ταξινομήσουμε μία ακολουθία με n στοιχεία σε χρόνο $O(n \log n)$
- Ο αλγόριθμος αυτός ονομάζεται ταξινόμηση σωρού
- Η ταξινόμηση σωρού είναι κατά πολύ γρηγορότερη από τετραγωνικούς αλγορίθμους ταξινόμησης όπως η ταξινόμηση με επιλογή και ταξινόμηση με εισαγωγή

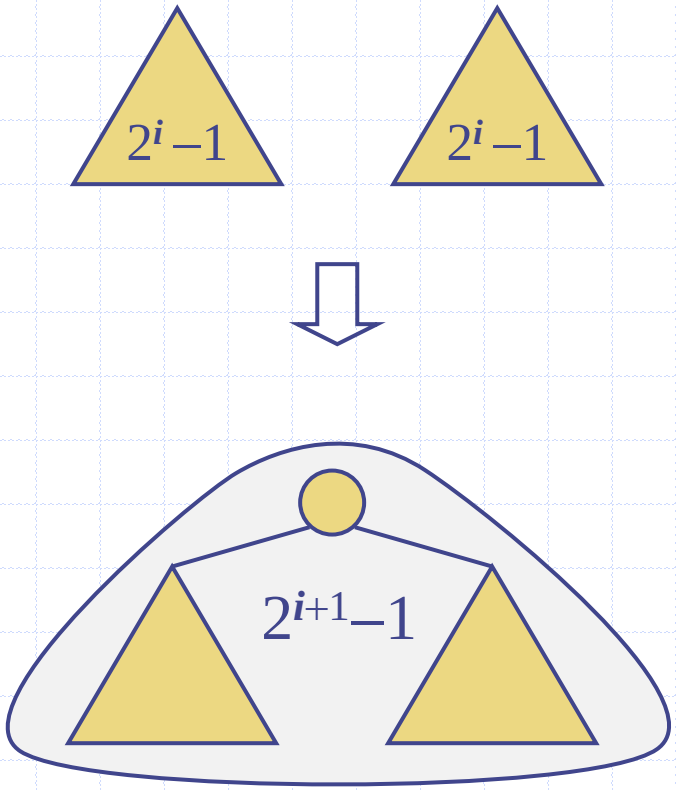
Συγχώνευση δύο Σωρών

- Δίδονται δύο σωροί και ένα κλειδί k
- Δημιουργούμε έναν νέο σωρό με το k στην ρίζα και τους δύο σωρούς ως υπό-δένδρα
- Πραγματοποιούμε downheap για να αποκαταστήσουμε την ιδιότητα της διάταξης σωρού

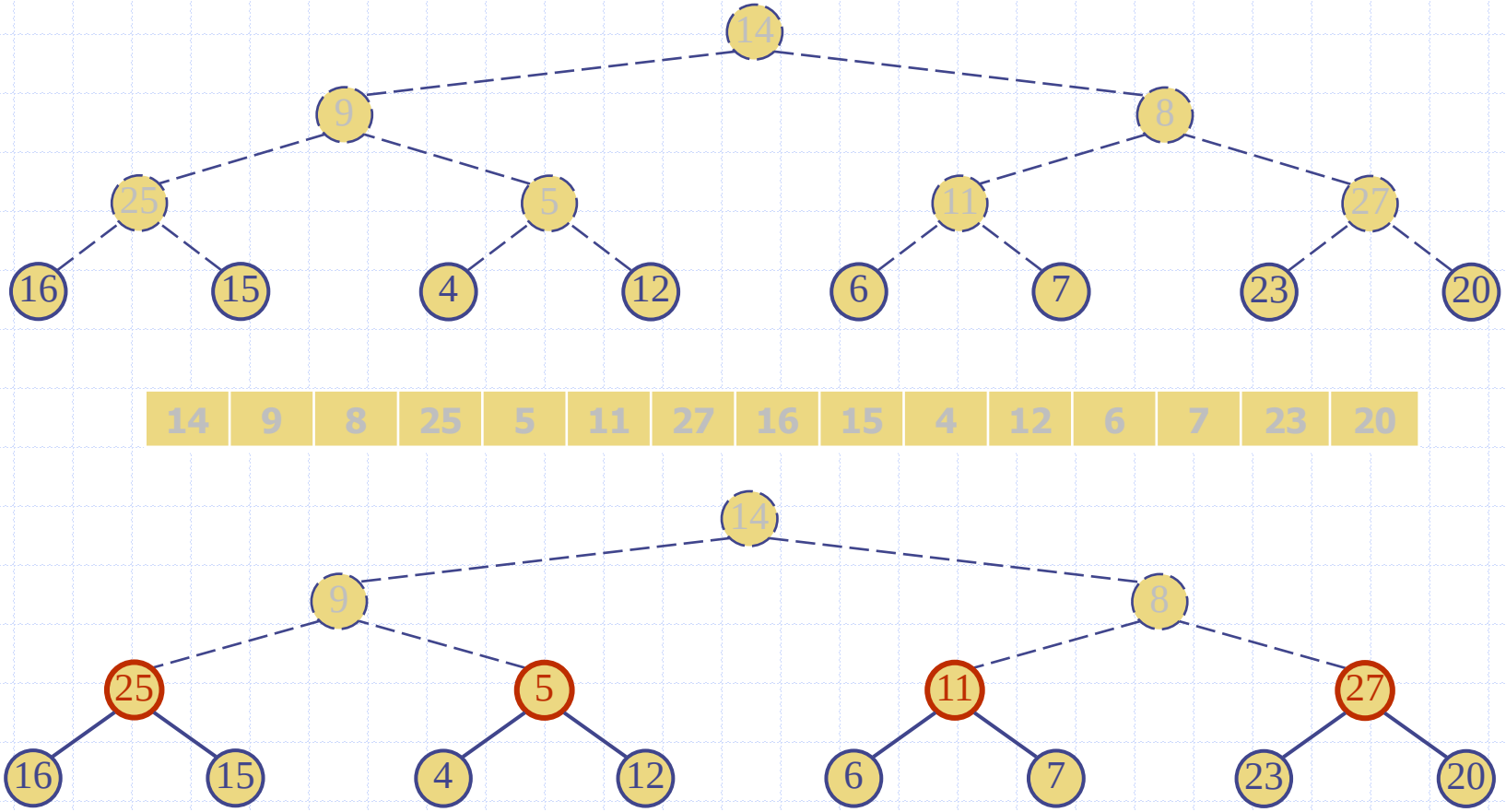


Κατασκευή σωρού από κάτω προς τα πάνω

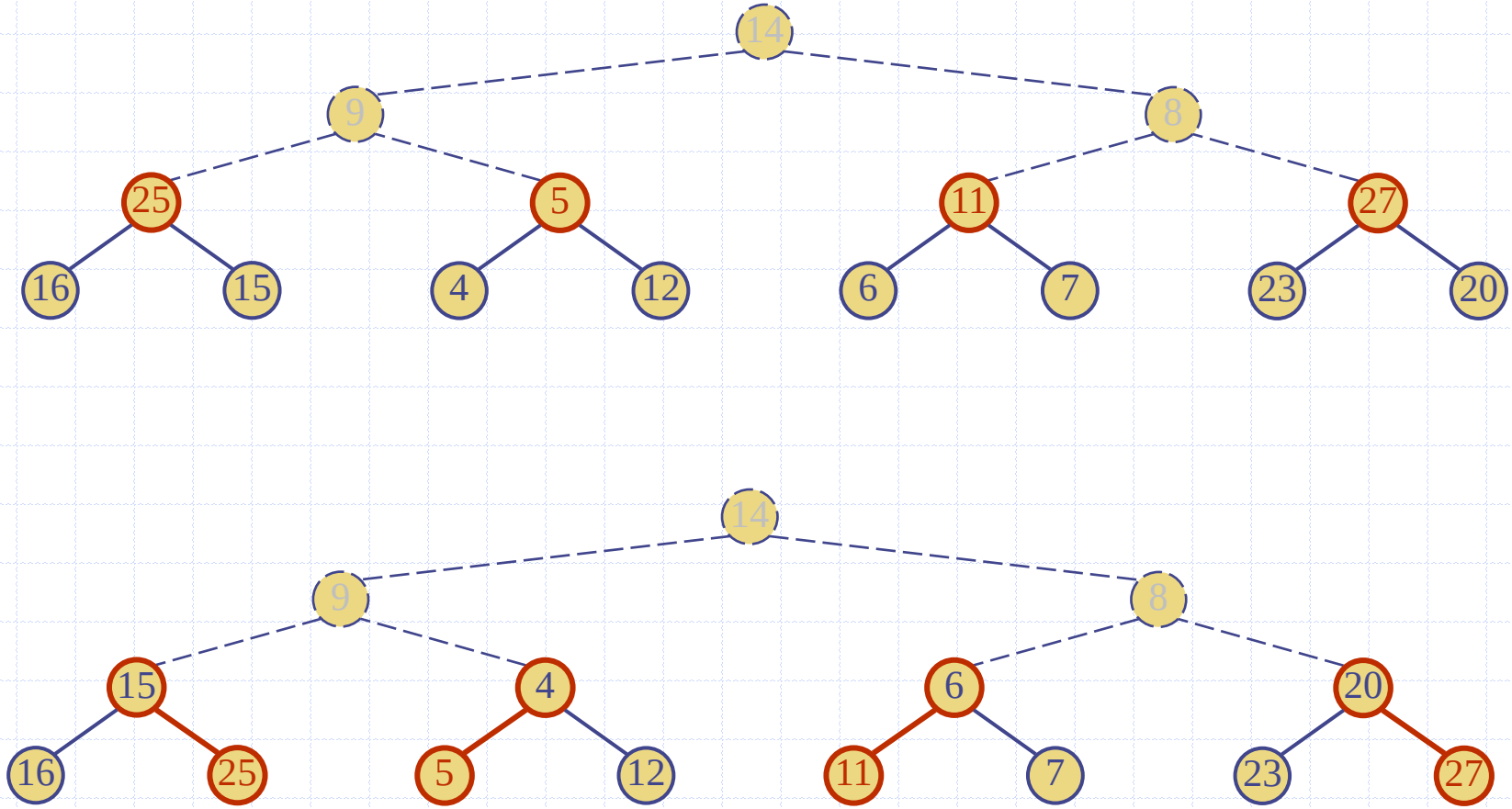
- Μπορούμε να κατασκευάσουμε έναν σωρό με n κλειδιά χρησιμοποιώντας κατασκευή από κάτω προς τα πάνω σε $\log n$ φάσεις
- Στην φάση i, ζεύγη σωρών με $2^i - 1$ κλειδιά συγχωνεύονται σε σωρούς με $2^{i+1} - 1$ κλειδιά



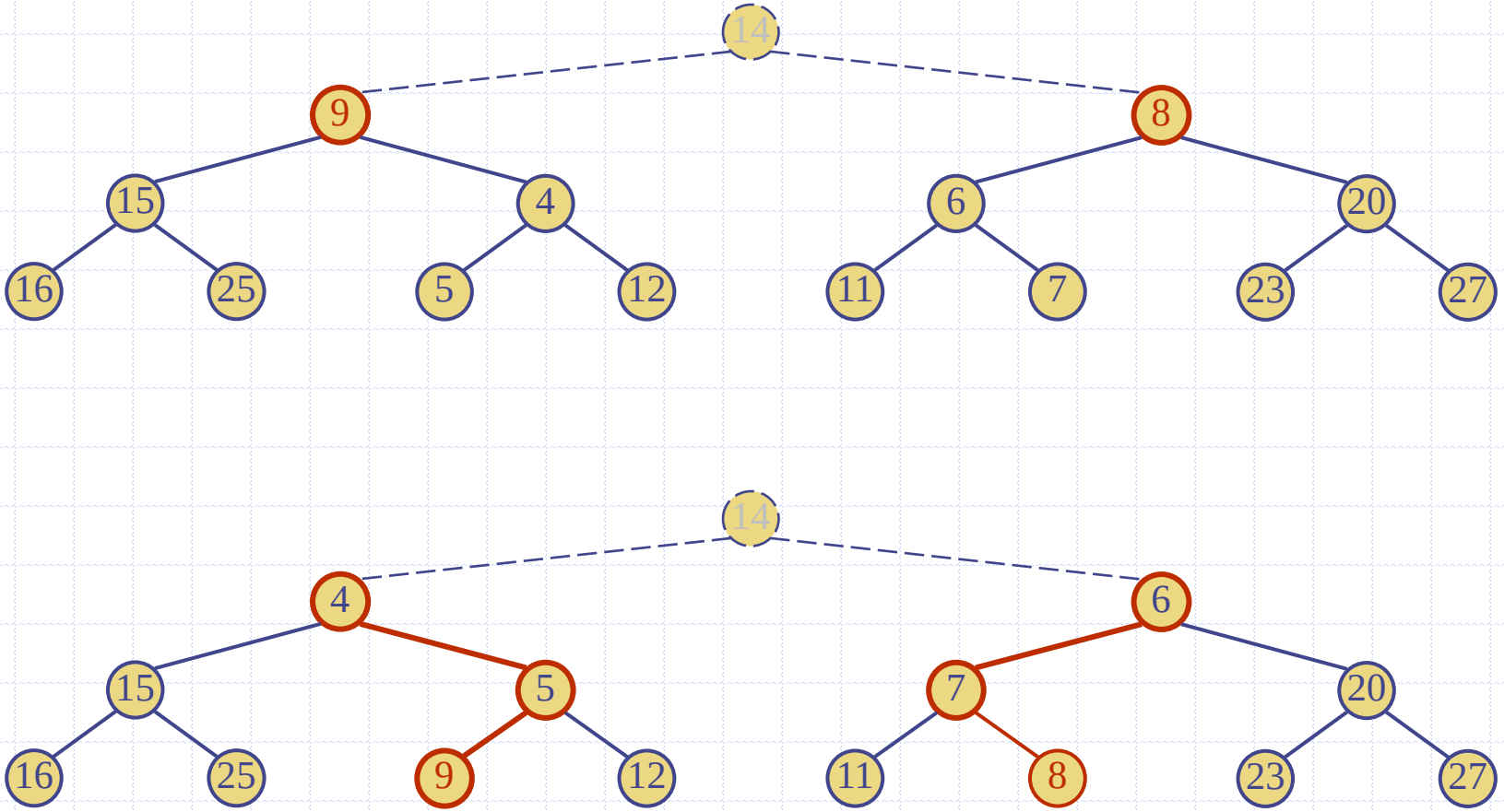
Παράδειγμα


14
9
8
25
5
11
27
16
15
4
12
6
7
23
20
14 9 8 25 5 11 27 16 15 4 12 6 7 23 20
14
9
8
25
5
11
27
16
15
4
12
6
7
23
20

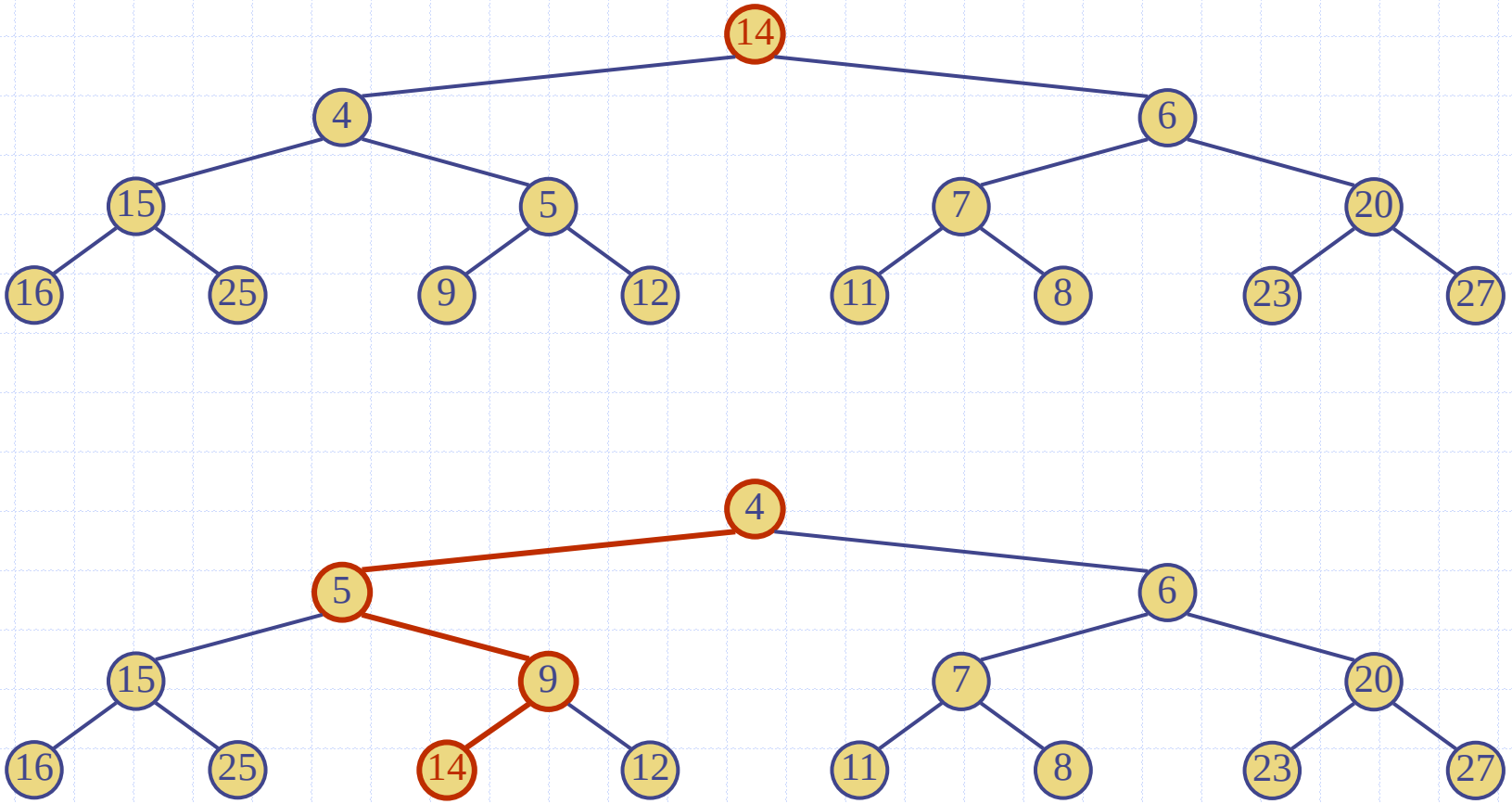
Παράδειγμα (συνέχεια.)



Παράδειγμα (συνέχεια.)



Παράδειγμα (τέλος)



Ανάλυση

- Οπτικοποιούμε το χρόνο χειρότερης περίπτωσης ενός downheap με ένα μονοπάτι που πηγαίνει πρώτα δεξιά και μετά επαναλαμβανόμενα πηγαίνει αριστερά μέχρι το κάτω μέρος (αυτό το μονοπάτι μπορεί να διαφέρει από το πραγματικό μονοπάτι του downheap)
- Καθώς κάθε κόμβος θα διασχισθεί το πολύ από δύο μονοπάτια ο συνολικός αριθμός των κόμβων στα μονοπάτια είναι $O(n)$
- Οπότε η κατασκευή σωρού από κάτω προς τα πάνω είναι $O(n)$
- Η κατασκευή σωρού από κάτω προς τα πάνω είναι γρηγορότερη από n συνεχόμενες εισαγωγές και επιταχύνει την πρώτη φάση της ταξινόμησης σωρού, που απαιτεί $O(n \log n)$ χρόνο στην δεύτερη φάση της.