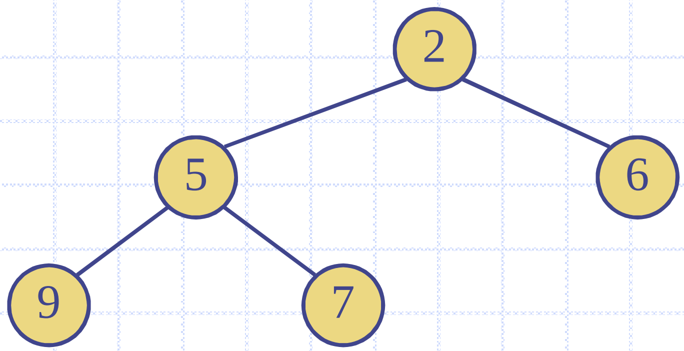
Σωροί



ΑΤΔ Ουρά Προτεραιότητας

- Μια ουρά προτεραιότητας αποθηκεύει μια συλλογή από καταχωρήσεις
- Κάθε καταχώρηση είναι ένα ζεύγος
(key, value)
- Βασικές μέθοδοι του ΑΤΔ Ουρά Προτεραιότητας
  - insert(k, x)
εισάγει μια καταχώρηση με κλειδί k και τιμή x
  - removeMin()
διαγράφει και επιστρέφει την καταχώρηση με την μικρότερη τιμή
- Επιπλέον μέθοδοι
  - min()
επιστρέφει, αλλά δεν διαγράφει, μια καταχώρηση με το μικρότερο κλειδί
  - size(), isEmpty()
- Εφαρμογές:
  - Standby πληρώματα
  - Πλειστηριασμοί
  - Αγορά μετοχών

Ταξινόμηση με Ουρά Προτεραιότητας

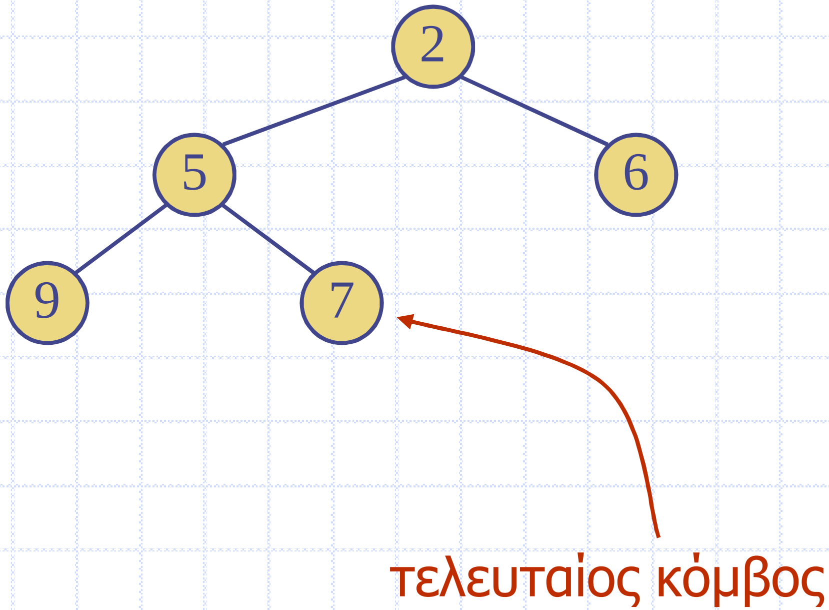
- Χρησιμοποιούμε μια ουρά προτεραιότητας
  - Εισαγωγή των στοιχείων με μια σειρά πράξεων insert
  - Διαγραφή των στοιχείων με μια σειρά πράξεων removeMin
- Ο χρόνος τρεξίματος εξαρτάται από την υλοποίηση της ουράς προτεραιότητας:
  - Μη ταξινομημένη ακολουθία δίνει ταξινόμηση με επιλογή: χρόνος $O(n^2)$
  - Ταξινομημένη ακολουθία δίνει ταξινόμηση με εισαγωγή: χρόνο $O(n^2)$
- Μπορούμε καλύτερα?

Algorithm ***PQ-Sort***(***S, C***)
  Input sequence ***S***, comparator ***C*** for the elements of ***S***
  Output sequence ***S*** sorted in increasing order according to ***C***
  P ← priority queue with comparator ***C***
  while ¬***S.isEmpty*** ()
    e ← ***S.remove*** (***S. first*** ())
    P.insertItem(***e***, ***e***)
  while ¬***P.isEmpty***()
    e ← ***P.removeMin***().***getKey***()
    S.addLast(***e***)

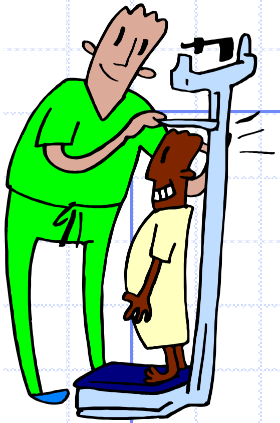
Σωροί

- Σωρός είναι ένα δυαδικό δένδρο που αποθηκεύει κλειδιά στους κόμβους του και ικανοποιεί τις παρακάτω ιδιότητες:
- Σειρά-σωρού: για κάθε εσωτερικό κόμβο v εκτός από τη ρίζα, $key(v) \geq key(parent(v))$
- Πλήρες Δυαδικό Δένδρο: έστω h το ύψος του σωρού
  - for $i = 0, \ldots, h - 1$, υπάρχουν 2^i κόμβοι βάθους of i
  - σε βάθος $h - 1$, οι εσωτερικοί κόμβοι είναι αριστερά των εξωτερικών κόμβων

- Ο τελευταίος κόμβος ενός σωρού είναι ο πιο δεξιός κόμβος στο μέγιστο βάθος



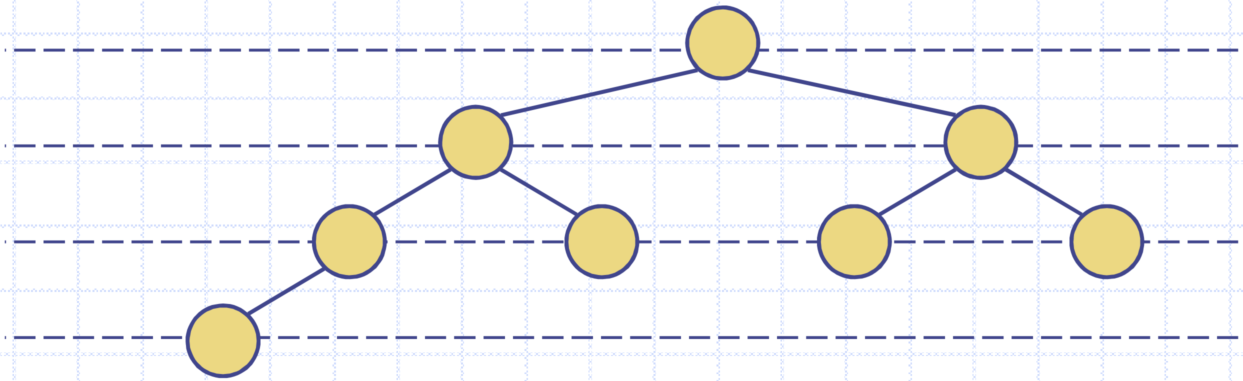
Ύψος ενός σωρού



- Θεώρημα: Ένας σωρός που αποθηκεύει n κλειδιά έχει ύψος $O(\log n)$

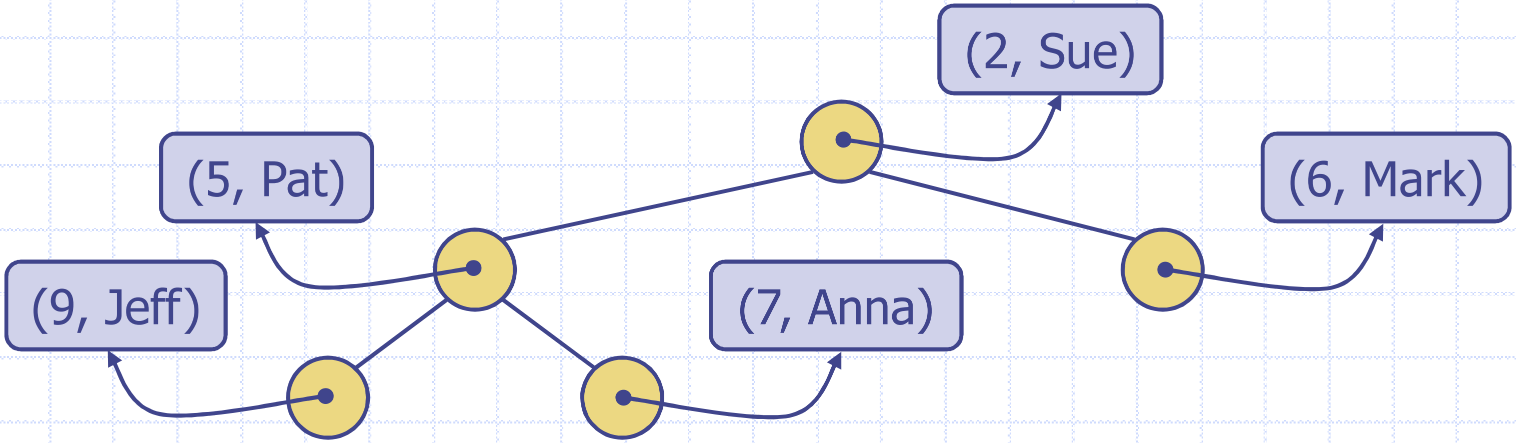
  Απόδειξη: (εφαρμόζουμε την ιδιότητα του πλήρους δυαδικού δένδρου)
  - Έστω h το ύψος ενός σωρού με n κλειδιά
  - Αφού υπάρχουν 2^i κλειδιά σε βάθος $i = 0, \ldots, h-1$ και τουλάχιστον ένα κλειδί h, έχουμε $n \geq 1 + 2 + 4 + \ldots + 2^{h-1} + 1$
  - Επομένως, $n \geq 2^h$, δηλ., $h \leq \log n$

βάθος	κλειδιά
0	1
1	2
$h-1$	2^{h-1}
h	1



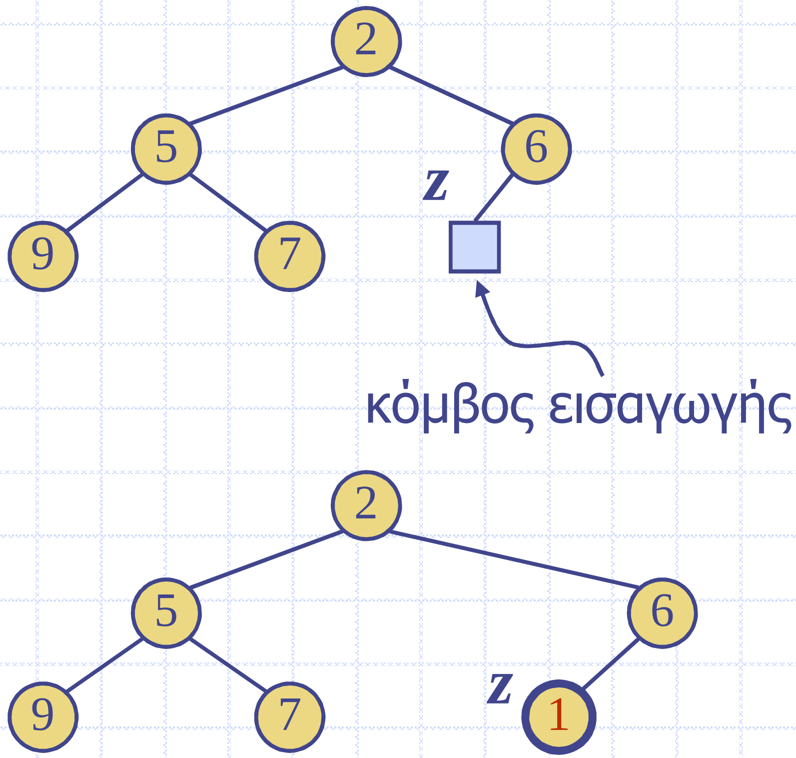
Σωροί και Ουρές Προτεραιότητας

- Μπορούμε να χρησιμοποιήσουμε ένα σωρό για να υλοποιήσουμε μια ουρά προτεραιότητας
- Αποθηκεύουμε ένα στοιχείο (κλειδί, δεδομένα) σε κάθε εσωτερικό κόμβο
- Κρατάμε τη θέση του τελευταίου κόμβου



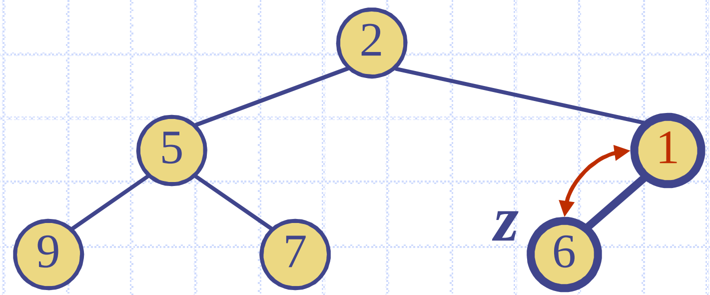
Εισαγωγή σε ένα Σωρό

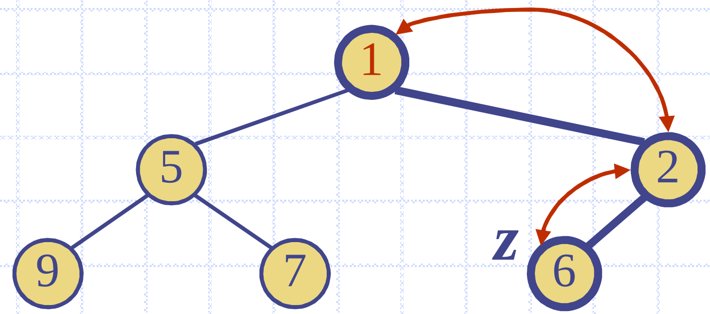
- Η μέθοδος insertItem του ΑΤΔ ουρά προτεραιότητας αντιστοιχεί στην εισαγωγή ενός κλειδιού k στο σωρό
- Ο αλγόριθμος εισαγωγής αποτελείται από τρία βήματα
  - Εύρεση του κόμβου εισαγωγής z (ο νέος τελευταίος κόμβος)
  - Αποθήκευση του k στο z
  - Αποκατάσταση της διάταξης που ορίζει η ιδιότητα του σωρού (εξετάζεται παρακάτω)



Αλγόριθμος Upheap

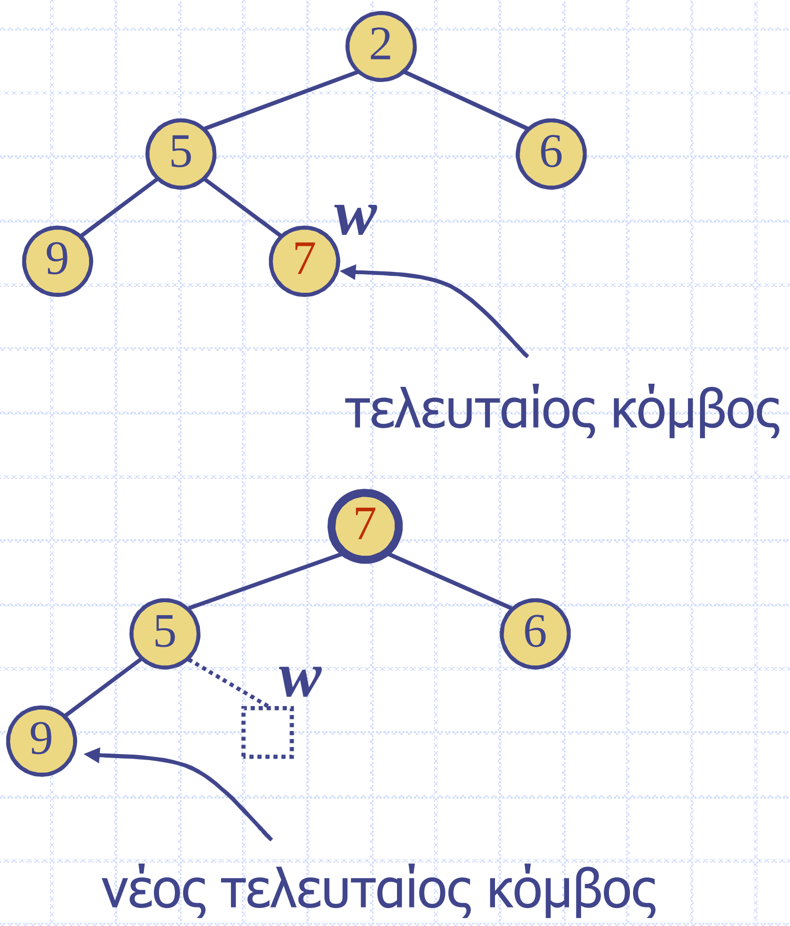
- Μετά την εισαγωγή ενός νέου κλειδιού k, μπορεί να παραβιάζεται η ιδιότητα της διάταξης του σωρού
- Ο αλγόριθμος upheap αποκαθιστά την ιδιότητα διάταξης του σωρού ανταλάσσοντας το k σε προς τα πάνω διαδρομή από τον κόμβο εισαγωγής
- Ο Upheap τερματίζει όταν το κλειδί k φθάσει στη ρίζα ή ένα κόμβο του οποίου ο γονέας έχει κλειδί μικρότερο ή ίσο με το k
- Αφού ο σωρός έχει ύψος $O(\log n)$, ο upheap τρέχει σε χρόνο $O(\log n)$





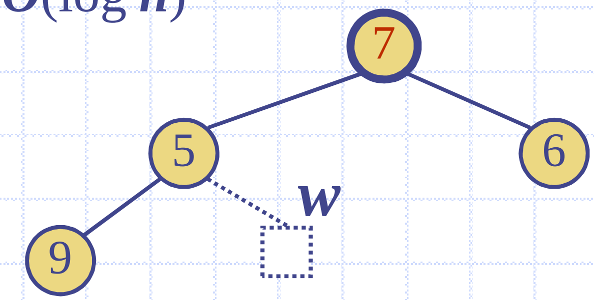
Διαγραφή από ένα Σωρό

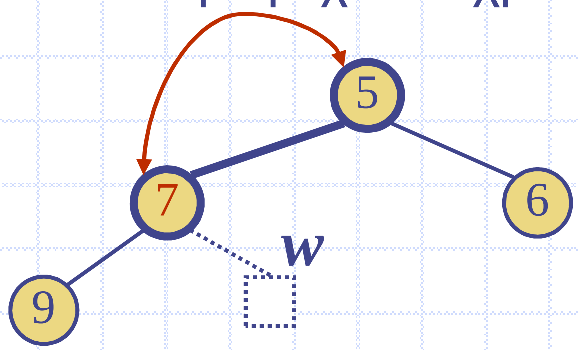
- Η μέθοδος removeMin του ΑΤΔ ουράς προτεραιότητας αντιστοιχεί στη διαγραφή της ρίζας του σωρού
- Ο αλγόριθμος διαγραφής ακολουθεί τρία βήματα
  - Αντικατάσταση του κλειδιού της ρίζας με το κλειδί του τελευταίου κόμβου *w*
  - Διαγραφή του *w*
  - Αποκατάσταση της ιδιότητας διάταξης του σωρού



Αλγόριθμος Downheap

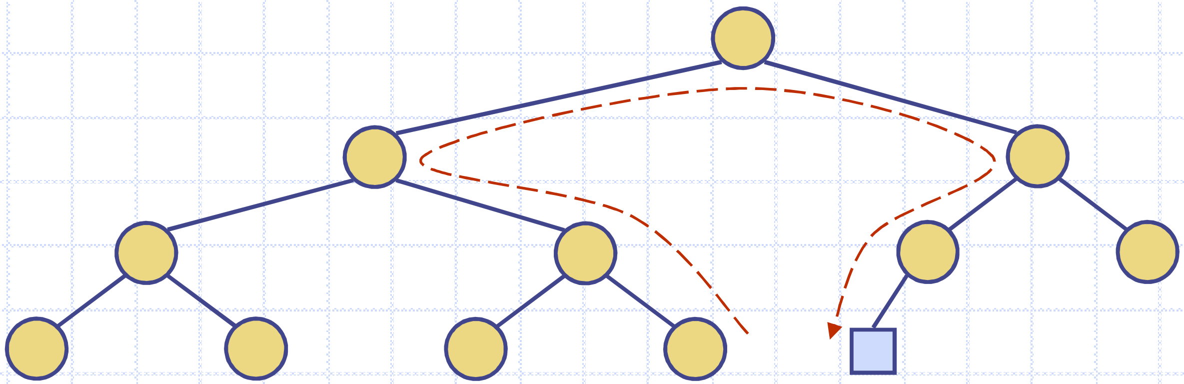
- Αφού αντικατασταθεί το κλειδί της ρίζας με το κλειδί k του τελευταίου κόμβου, μπορεί να έχει παραβιασθεί η διάταξη του σωρού
- Ο αλγόριθμος downheap αποκαθιστά την ιδιότητα διάταξης του σωρού ανταλλάσοντας το κλειδί k στη διαδρομή προς τα κάτω από τη ρίζα
- Ο downheap τερματίζει όταν το κλειδί k φθάσει σε ένα φύλο ή σε ένα κόμβο του οποίου τα παιδά έχουν κλειδιά μεγαλύτερα ίσα με το k
- Αφού ο σωρός έχει ύψος $O(\log n)$, ο downheap τρέχει σε χρόνο $O(\log n)$



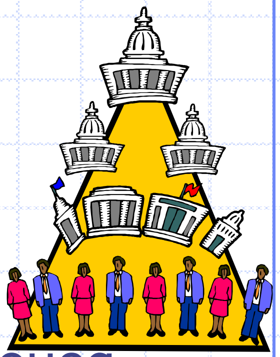


Ενημέρωση του τελευταίου κόμβου

- Ο κόμβος εισαγωγής μπορεί να βρεθεί διασχίζοντας μια διαδρομή από $O(\log n)$ κόμβους
  - Κίνηση προς τα πάνω μέχρι να φθάσει ένα αριστερό παιδί ή τη ρίζα
  - Αν φθάσει σε αριστερό παιδί, πήγαινε στο δεξιό παιδί
  - Κίνηση προς τα κάτω μέχρι να φθάσει σε ένα φύλο
- Μοιάζει με τον αλγόριθμο για ενημέρωση του τελευταίου κόμβου μετά τη διαγραφή



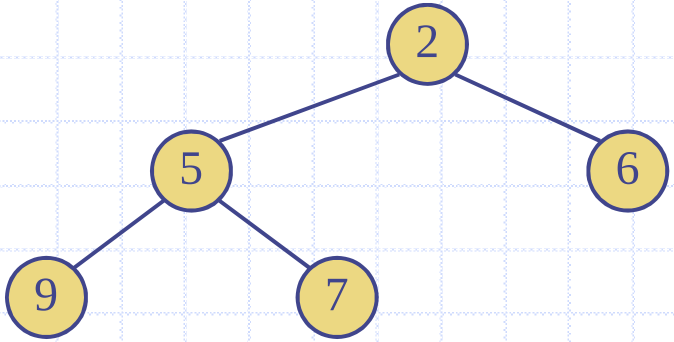
Ταξινόμηση Σωρού

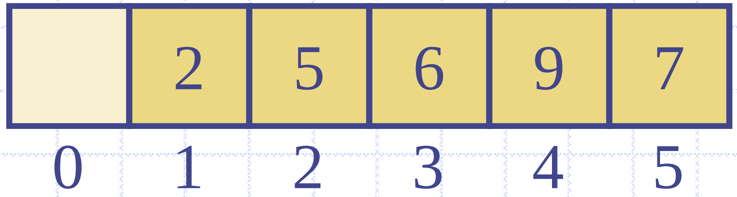


- Έστω μια ουρά προτεραιότητας με n στοιχεία που υλοποιείται με σωρό
  - ο χώρος που απαιτείται είναι $O(n)$
  - οι μέθοδοι insert και removeMin απαιτούν χρόνο $O(\log n)$
  - οι μέθοδοι size, isEmpty, και min απαιτούν χρόνο $O(1)$
- Χρησιμοποιώντας μια ουρά προτεραιότητας που βασίζεται σε σωρό, μπορούμε να ταξινομήσουμε μια ακολουθία από n στοιχεία σε χρόνο $O(n \log n)$
- Ο αλγόριθμος που προκύπτει λέγεται heap-sort
- Ο Heap-sort είναι πολύ πιο γρήγορος από τους τετραγωνικούς αλγόριθμους ταξινόμησης, όπως ο αλγόριθμος με εισαγωγή και με επιλογή

Υλοποίηση σωρού που βασίζεται σε διάνυσμα

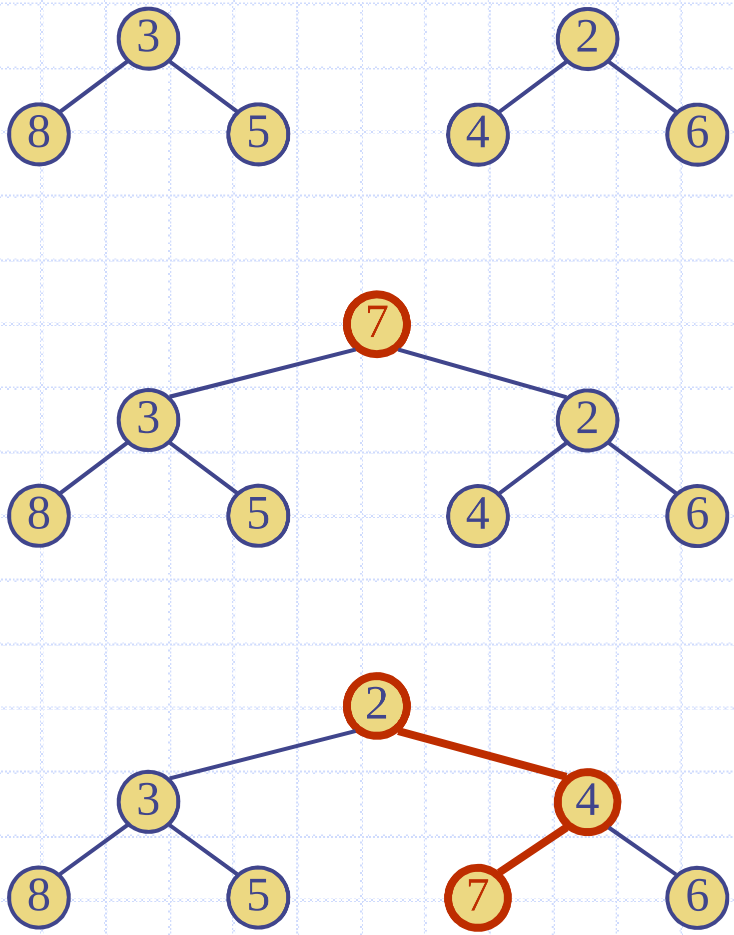
- Μπορούμε να παραστήσουμε ένα σωρό με n κλειδιά με χρήση ενός διανύσματος μήκους $n + 1$
- Για τον κόμβο στη θέση i
  - το αριστερό παιδί είναι στη θέση $2i$
  - το δεξιό παιδί είναι στη θέση the $2i + 1$
- Οι σύνδεσμοι μεταξύ κόμβων δεν αποθηκεύονται ρητά
- Η θέση 0 δεν χρησιμοποιείται
- Η πράξη εισαγωγή αντιστοιχεί σε εισαγωγή στη θέση $n + 1$
- Η πράξη removeMin αντιστοιχεί στη διαγραφή από τη θέση n
- Παράγει ταξινόμηση θέσης με σωρό





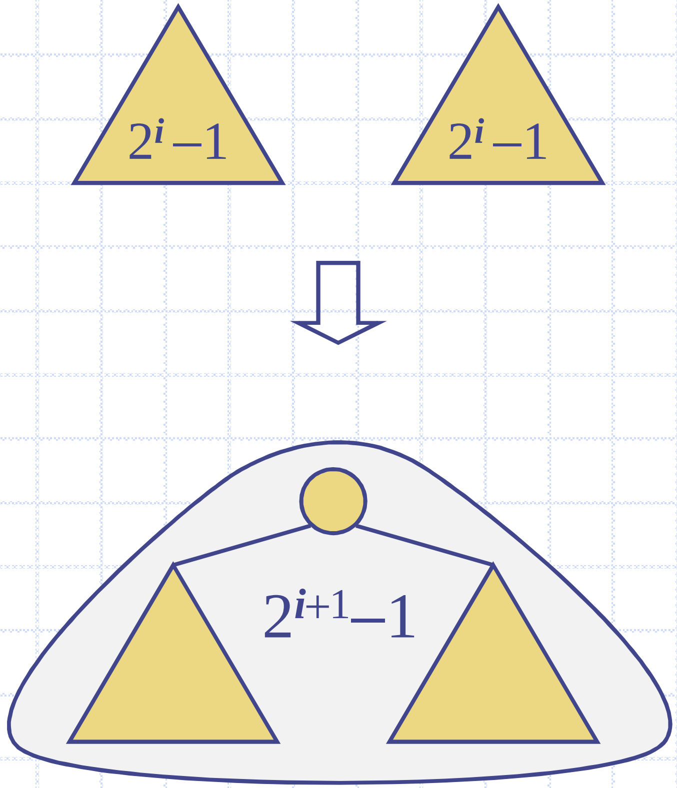
Συγχώνευση δύο Σωρών

- Δίδονται δυο σωροί και ένα κλειδί k
- Δημιουργούμε ένα νέο σωρό με τον κόμβο της ρίζας να αποθηκεύει το k και με δύο σωρούς σαν υποδένδρα
- Εκτελούμε ένα downheap για αποκατάσταση της ιδιότητας διάταξης του σωρού

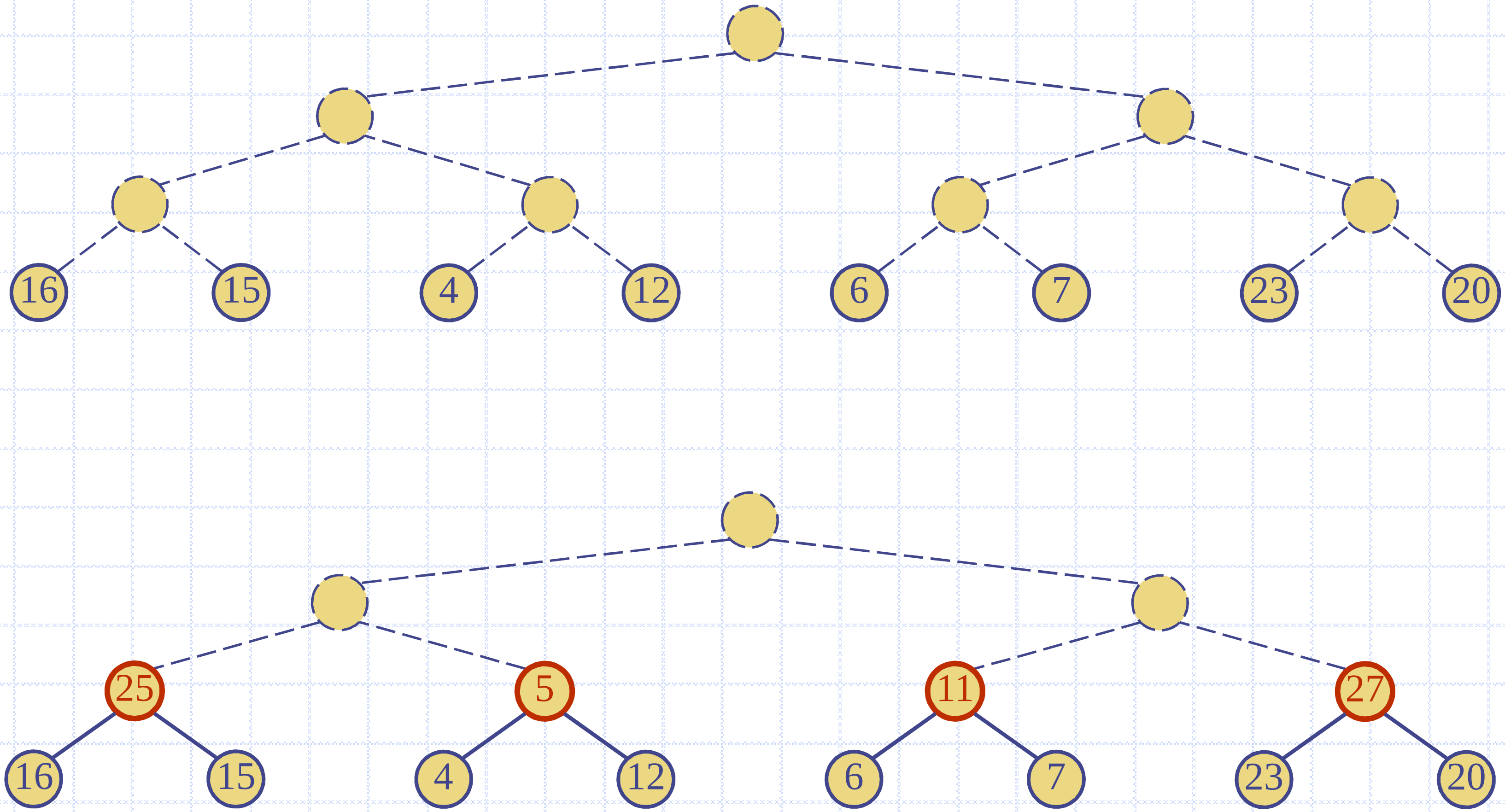


Προς τα πάνω κατασκευή σωρού

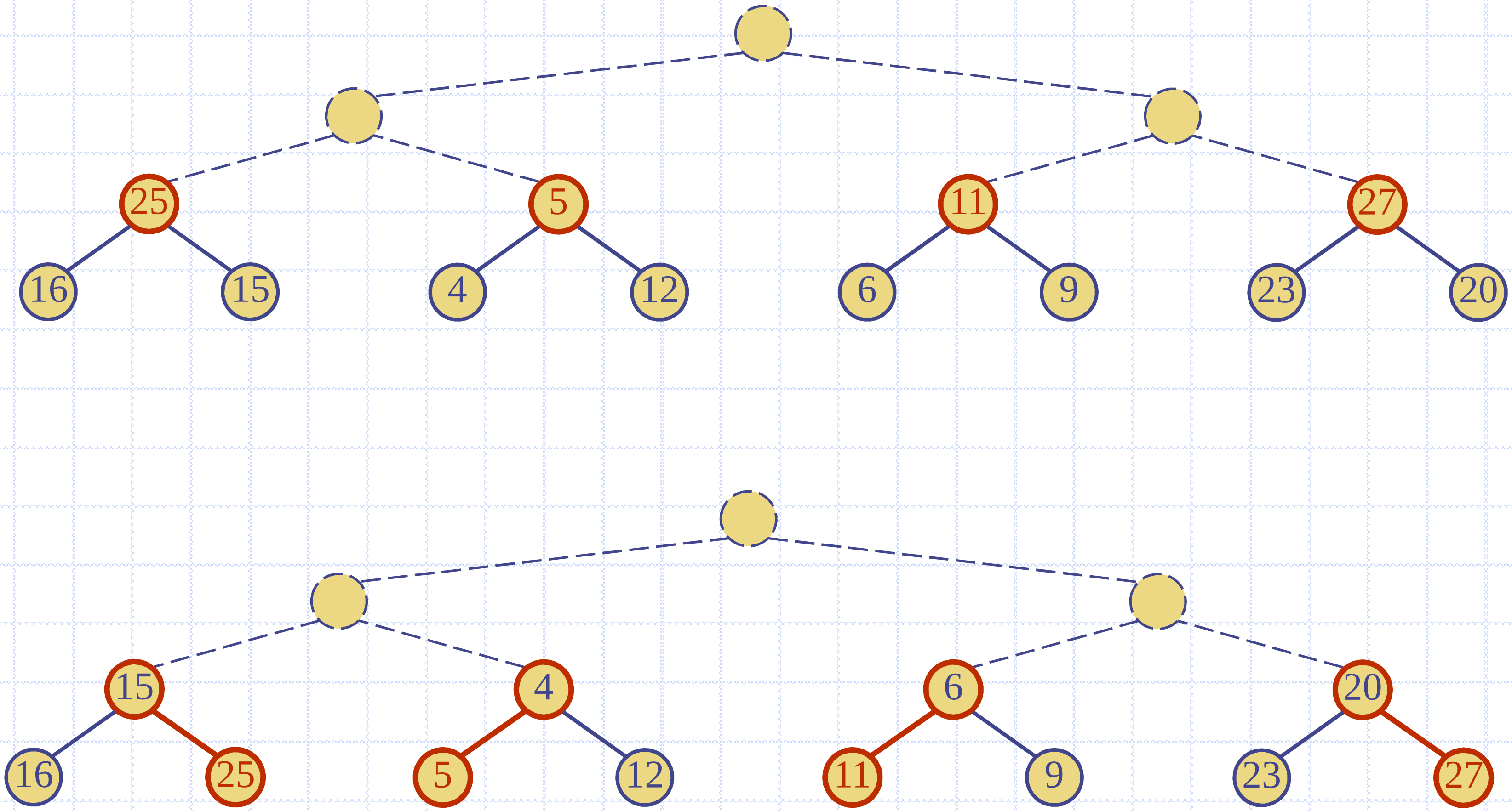
- Μπορούμε να κατασκευάσουμε ένα σωρό που αποθηκεύει n δοθέντα κλειδιά με χρήση μιας προς τα πίσω κατασκευής με $\log n$ φάσεις
- Στη φάση i, συγχωνεύονται ζεύγη σωρών με $2^i - 1$ κλειδιά σε σωρούς με $2^{i+1} - 1$ κλειδιά



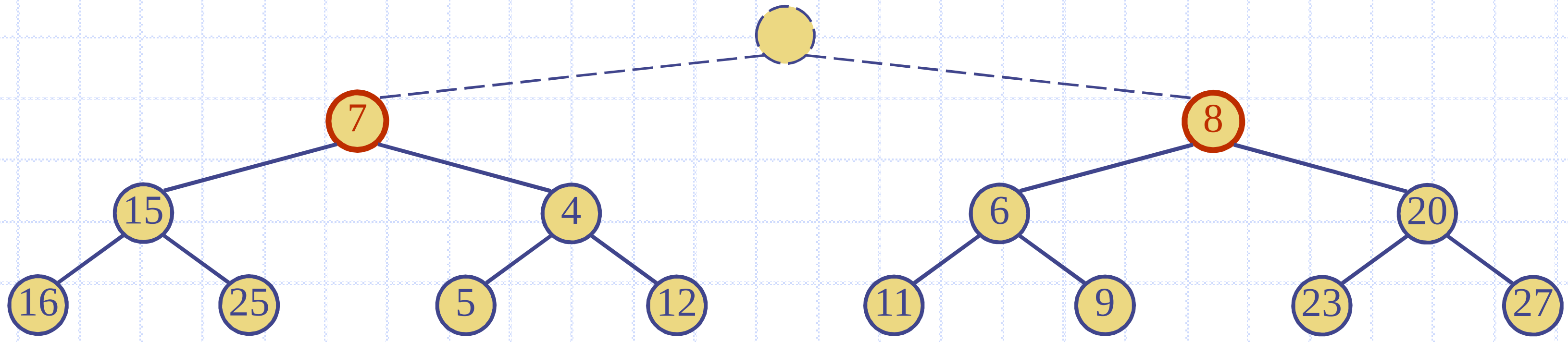
Παράδειγμα

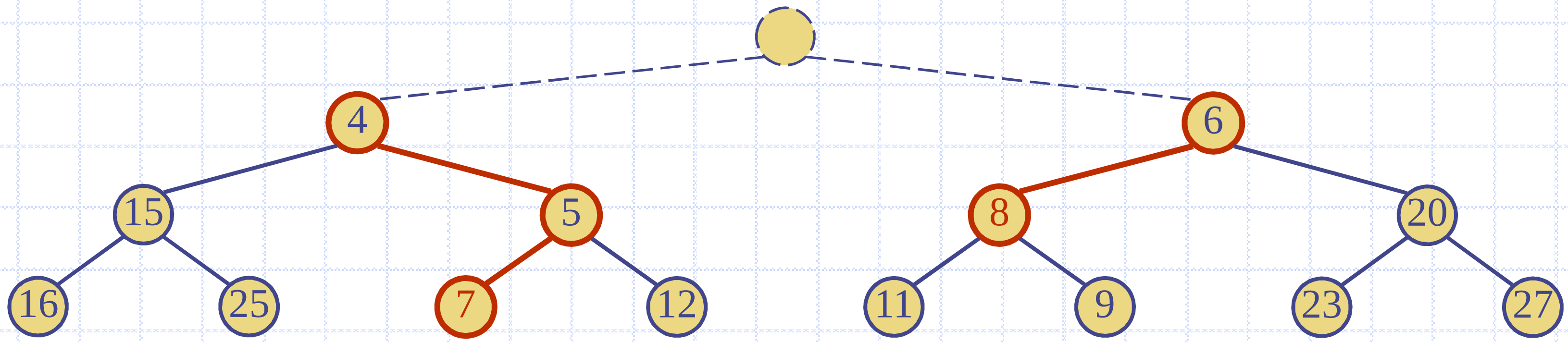


Παράδειγμα (συν.)

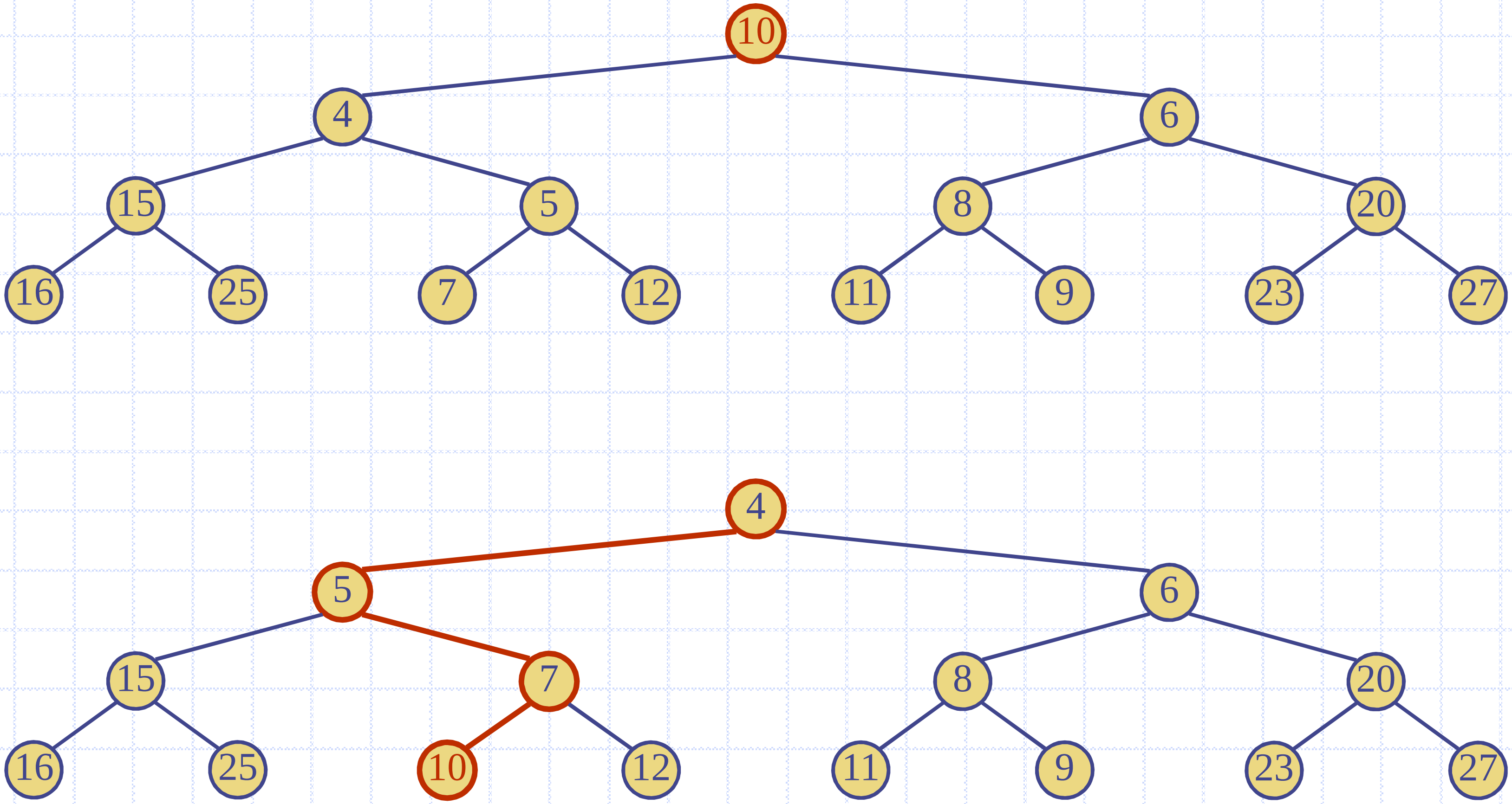


Παράδειγμα (συν.)

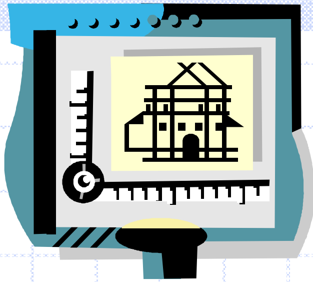




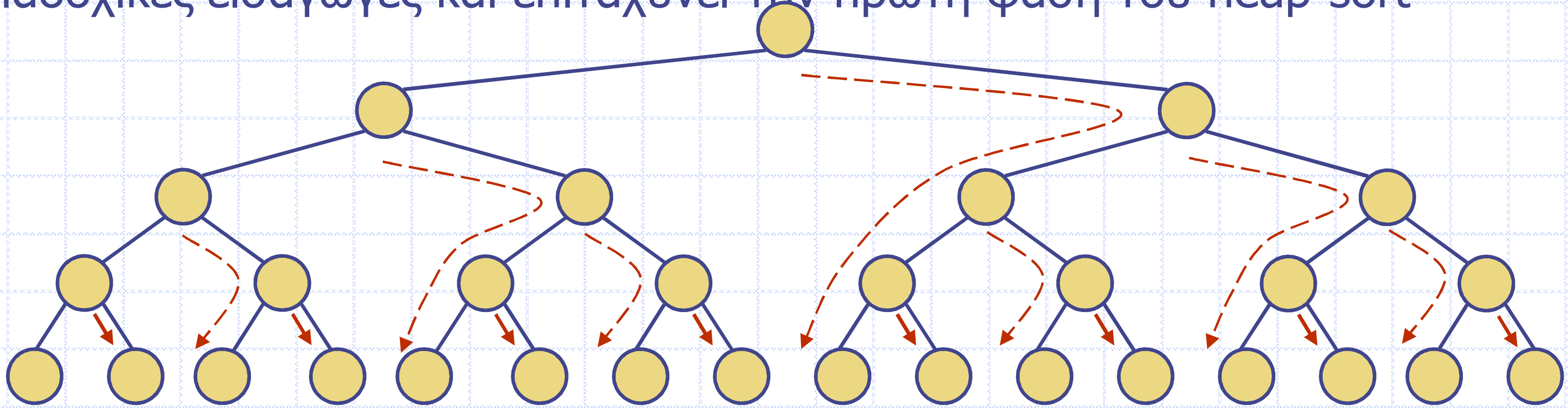
Παράδειγμα (τέλος)



Ανάλυση



- Η οπτικοποίηση του χειρότερου χρόνου ενός downheap με μια διαδρομή που πάει πρώτα δεξιά και στη συνέχεια πηγαίνοντας επαναληπτικά αριστερά μέχρι το τέλος του σωρού (αυτή η διαδρομή μπορεί να διαφέρει από την πραγματική διαδρομή του downheap)
- Αφού από κάθε κόμβο περνάνε τουλάχιστον δύο διαδρομές, το συνολικό πλήθος των κόμβων των διαδρομών είναι $O(n)$
- Επομένως, η προς τα πάνω κατασκευή του σωρού τρέχει σε χρόνο $O(n)$
- Η προς τα πάνω κατασκευή του σωρού είναι πιο γρήγορος από n διαδοχικές εισαγωγές και επιταχύνει την πρώτη φάση του heap-sort



Προσαρμοζόμενες Ουρές Προτεραιότητας

- Υπάρχουν περιπτώσεις, όπως οι παρακάτω που θέλουμε α παρακάμψουμε την σειρά διαγραφής:
  - Ένας επιβάτης βαριέται να περιμένει και ζητάει να διαγραφεί από την ουρά προτεραιότητας (χωρίς να είναι ο πρώτος αυτής)
  - Άλλος βρίσκει την κάρτα του σαν τακτικού επιβάτη και την δείχνει στον υπάλληλο.
  - Επιβάτης ζητάει να διορθωθεί το λανθασμένα γραμμένο όνομά του.

Επιπλέον Μέθοδοι

- remove(e): διαγραφή από την P και επιστροφή της καταχώρησης e.
- replace(e,K): αντικατάσταση με k και επιστροφή της καταχώρησης της P
- replaceValue(e,x): αντικατάσταση με το x και επιστροφή της καταχώρησης e της P.

Υλοποίηση Προσαρμοζόμενης Ουράς προτεραιότητας

- Επεκτείνουμε την καταχώρηση ενός αντικειμένου με μια μεταβλητή στιγμιότυπου *Position* (θέση). Λέγεται καταχώρηση ενήμερη θέσης.
  - Υλοποίηση με ταξινομημένη λίστα
  - Υλοποίηση με σωρό.