

Προγραμματιστικές Τεχνικές

Δυναμικές δομές: σύνολα – ουρές προτεραιότητας –
κατακερματισμός

Union-Find, Heaps, Hashing

Διδάσκοντες

Γιώργος Αλεξανδρίδης (gealexandri@islab.ntua.gr)

Βασίλης Βεσκούκης (v.vescoukis@cs.ntua.gr)

Θάνος Βουλόδημος (thanosv@mail.ntua.gr)

Γιώργος Γκούμας (goumas@cslab.ece.ntua.gr)

Άρης Παγουρτζής (pagour@cs.ntua.gr)

Γιώργος Σιόλας (gsiolas@islab.ntua.gr)

Γιώργος Στάμου (gstam@cs.ntua.gr)

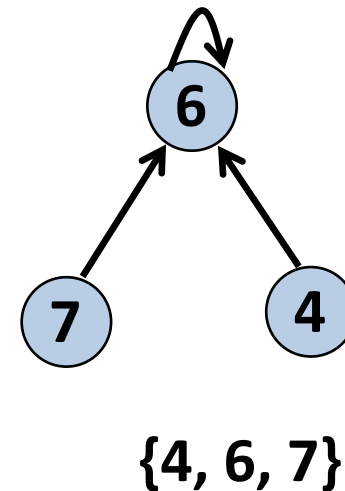
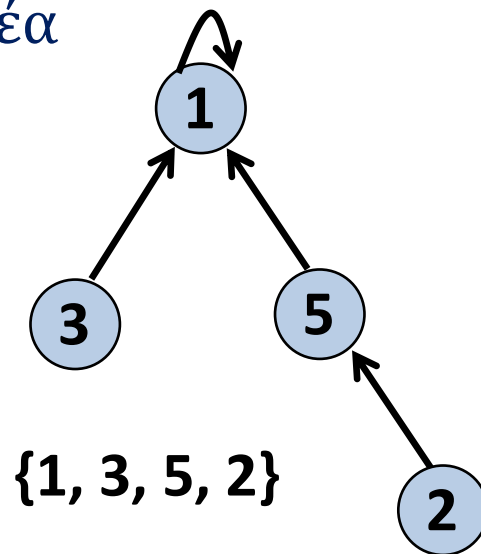
Παρασκευή Τζούβελη (tpar@image.ntua.gr)

Επιμέλεια διαφανειών: Άρης Παγουρτζής (ΕΜΠ)

Δομές UNION-FIND

Ε

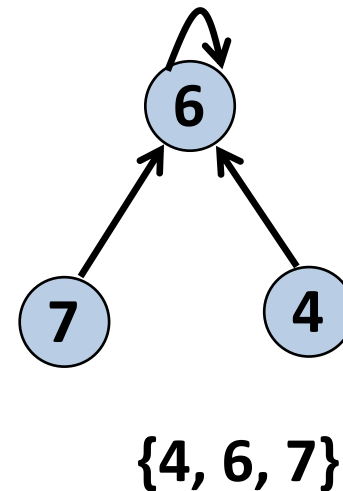
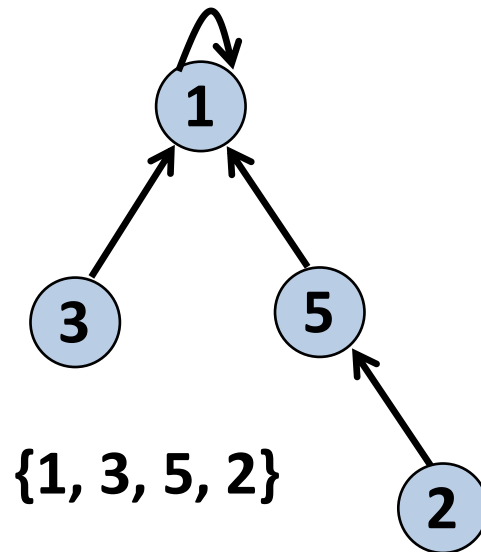
- Υποστηρίζουν πράξεις συνόλων:
 - Εύρεση (αναζήτηση) [**Find**]
 - Ένωση [**Union**]
- Υλοποίηση συνόλων με δένδρα: ένα δένδρο για κάθε σύνολο, «αντιπρόσωπος» η ρίζα, κόμβοι δείχνουν προς τον γονέα



Δομές UNION-FIND

Ε

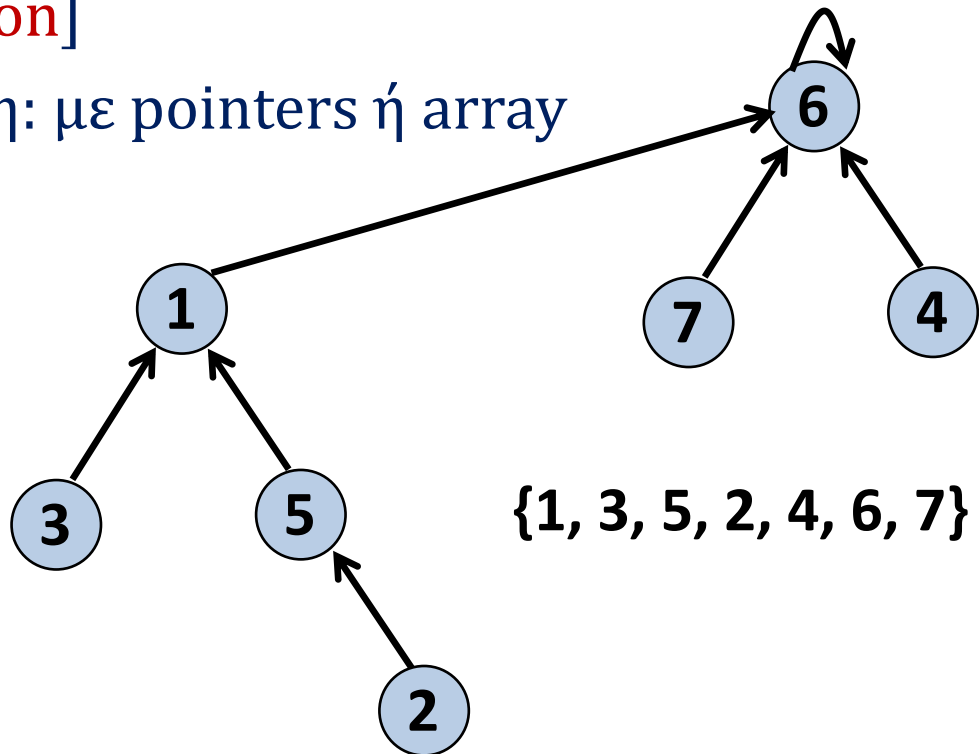
- Υποστηρίζουν πράξεις συνόλων:
 - Εύρεση (αναζήτηση) [**Find**]
 - Ένωση [**Union**]
- Απλή υλοποίηση: με pointers ή array



Δομές UNION-FIND

Ε

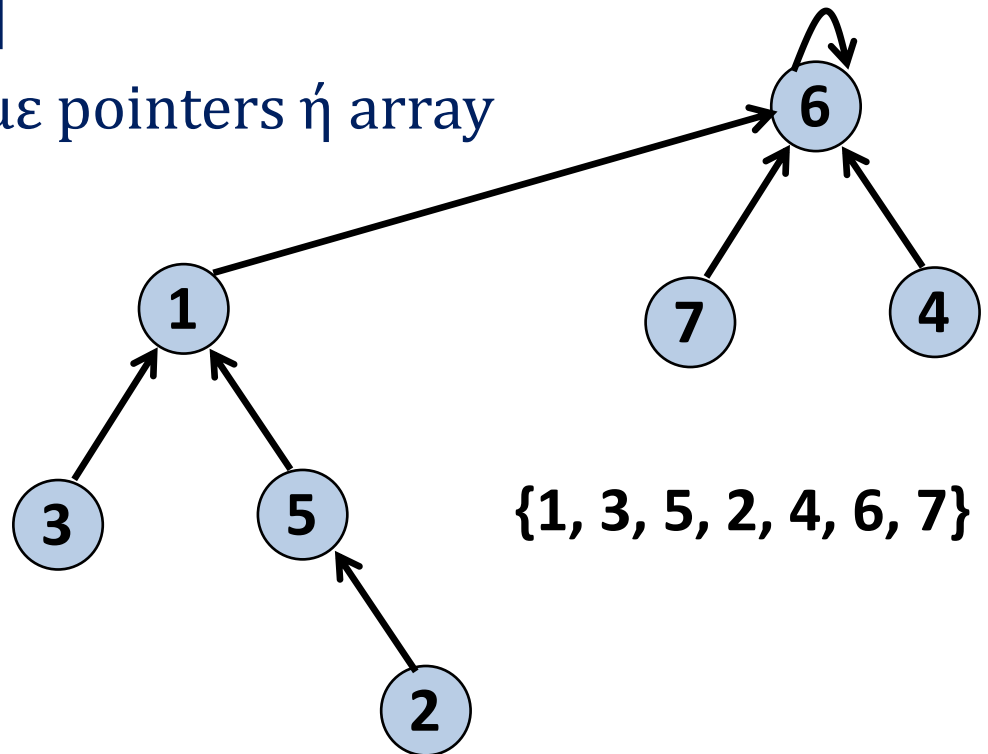
- Υποστηρίζουν πράξεις συνόλων:
 - Εύρεση (αναζήτηση) [**Find**]
 - Ένωση [**Union**]
- Απλή υλοποίηση: με pointers ή array
- Ένωση: **$O(1)$**



Δομές UNION-FIND

Ε

- Υποστηρίζουν πράξεις συνόλων:
 - Εύρεση (αναζήτηση) [**Find**]
 - Ένωση [**Union**]
- Απλή υλοποίηση: με pointers ή array
- Ένωση: $O(1)$
- Εύρεση: ?

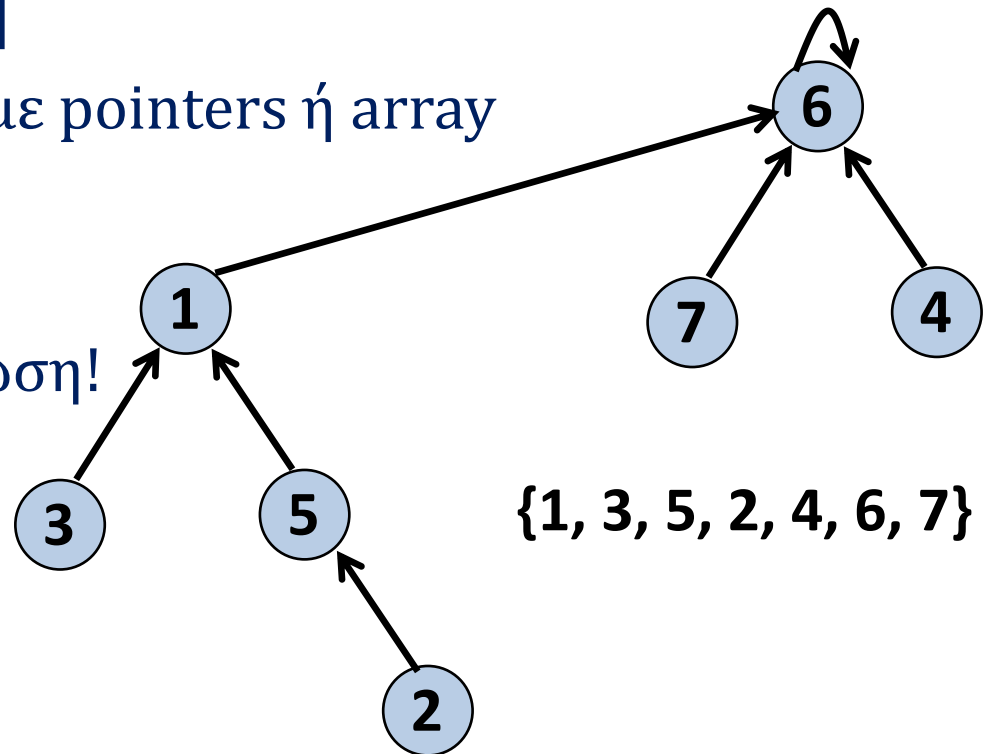


Δομές UNION-FIND

Ε

- Υποστηρίζουν πράξεις συνόλων:
 - Εύρεση (αναζήτηση) [**Find**]
 - Ένωση [**Union**]
- Απλή υλοποίηση: με pointers ή array
- Ένωση: $O(1)$
- Εύρεση: $O(n)$

χειρότερη περίπτωση!
Γίνεται καλύτερα;



Δομές UNION-FIND

Ε

- Υποστηρίζουν πράξεις συνόλων:
 - Εύρεση (αναζήτηση) [**Find**]
 - Ένωση [**Union**]
- Υλοποίηση συνόλων με δένδρα: ένα δένδρο για κάθε σύνολο, «αντιπρόσωπος» η ρίζα, κόμβοι δείχνουν προς τον γονέα
- Απλή υλοποίηση: με pointers ή array
- Ένωση: **$O(1)$**
- Εύρεση: **$O(\log n)$** με **Union-by-Size**

Δομές UNION-FIND

Ε

- Εύρεση: $O(\log n)$ με **Union-by-Size** :
το δέντρο με τα λιγότερα στοιχεία «κρεμιέται» στη ρίζα
του δέντρου με τα περισσότερα στοιχεία
 - παραλλαγές: **Union-by-Height**, **Union-by-Rank**

Ισχυρισμός: κάθε δένδρο n κόμβων που προκύπτει με διαδοχικές εφαρμογές Union-by-Size έχει ύψος

$$h \leq \log_2 n$$

Απόδειξη: με επαγωγή στην κατασκευή του δένδρου

Δομές UNION-FIND

Ε

Θεώρημα: κάθε δένδρο n κόμβων που προκύπτει με διαδοχικές εφαρμογές Union-by-Size έχει ύψος

$$h \leq \log_2 n$$

Απόδειξη: με επαγωγή στην κατασκευή του δένδρου :

Θεωρήστε δένδρο ύψους h με n κόμβους, από ένωση δύο δένδρων με ύψη $h_1 \leq \log_2 n_1$, $h_2 \leq \log_2 n_2$

Έστω χ.β.γ. ότι $n_1 \leq n_2$

Αν $h_1 < h_2$ τότε $h = h_2 \leq \log_2 n_2 < \log_2 n$

Αν $h_1 \geq h_2$ τότε $h = h_1 + 1 \leq \log_2 (2n_1) < \log_2 n$



Δομές UNION-FIND

Ε

- Υποστηρίζουν πράξεις συνόλων:
 - Ένωση: $O(1)$
 - Εύρεση: $O(\log^* n)$ *αποσβετικά* (amortized analysis) σε σύνολο m πράξεων

με **Union-by-Rank + Path Compression** (σε κάθε εύρεση όλα τα στοιχεία του μονοπατιού προς την ρίζα τα «κρεμάμε» κατευθείαν στη ρίζα)

($\log^* n \leq 5$ για $n \leq 2^{65536}$!!)

Δομές UNION-FIND

Ε

- Υποστηρίζουν πράξεις συνόλων:
 - Ένωση: $O(1)$
 - Εύρεση: $O(\alpha(n))$ *αποσβετικά* (amortized analysis)
σε σύνολο m πράξεων
(με πιο πολύπλοκη ανάλυση)

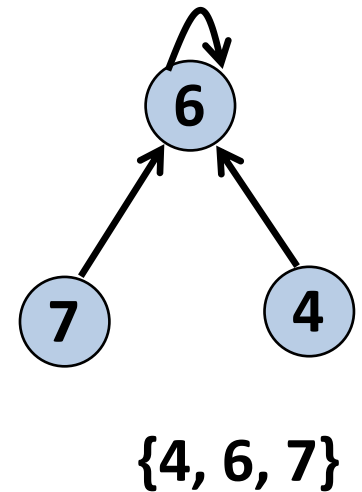
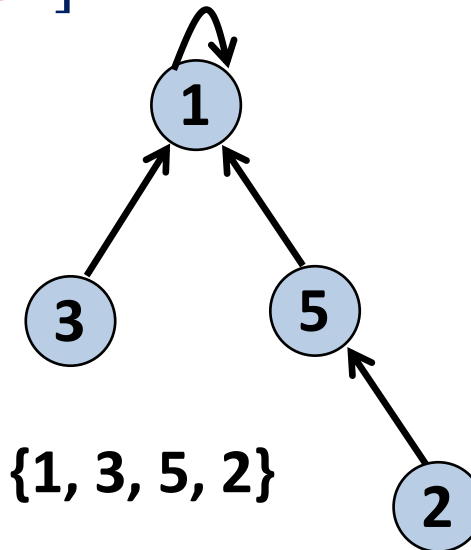
με **Union-by-Rank + Path Compression**

$\alpha(n) \leq 4$ για $n \leq 2^{2^{65536}} - 3$, αντίστροφη Ackermann

Δομές UNION-FIND

Ε

- Υποστηρίζουν πράξεις συνόλων:
 - Εύρεση (αναζήτηση) [**Find**]
 - Ένωση [**Union**]
- Υλοποίηση με **πίνακα**

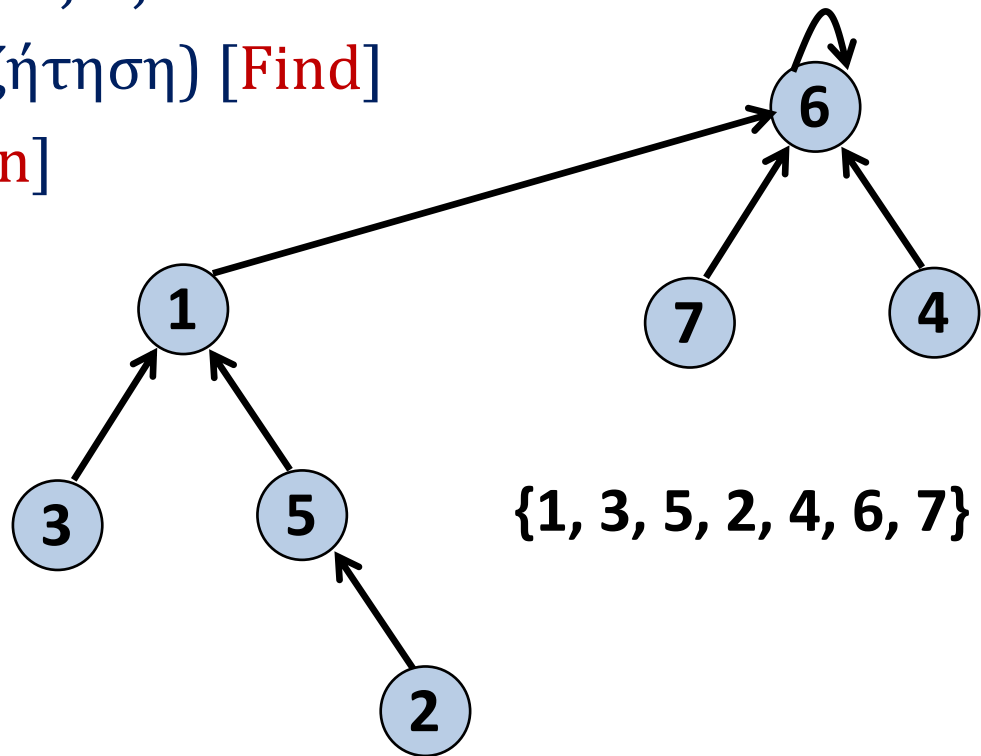


1	2	3	4	5	6	7
1	5	1	6	1	6	6

Δομές UNION-FIND

Ε

- Υποστηρίζουν πράξεις συνόλων:
 - Εύρεση (αναζήτηση) [**Find**]
 - Ένωση [**Union**]
- Υλοποίηση με **πίνακα**



1	2	3	4	5	6	7
6	5	1	6	1	6	6

Ουρά προτεραιότητας (priority queue)



- Δομή για αποθήκευση συνόλου στοιχείων με **σχέση προτεραιότητας** ανάμεσα στα στοιχεία του (ολική διάταξη)
 - π.χ. προηγούνται τα μικρότερα στοιχεία
- Απαραίτητες λειτουργίες:
 - **δημιουργία** ουράς
 - **εισαγωγή** στοιχείου
 - **διαγραφή** στοιχείου
 - **ανάκτηση** (και διαγραφή) στοιχείου **μέγιστης προτεραιότητας**

Ουρές προτεραιότητας: απλοϊκή υλοποίηση



- Με γραμμική δομή (list, array):
 - εισαγωγή σε χρόνο $O(1)$
 - ανάκτηση (και διαγραφή) στοιχείου μέγιστης προτεραιότητας: χρόνος $O(n)$

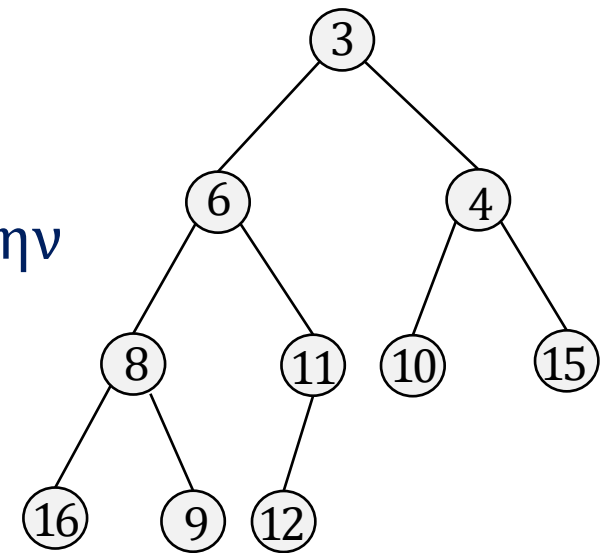
Ουρές προτεραιότητας: απλοϊκή υλοποίηση



- Με γραμμική δομή (list, array):
 - εισαγωγή σε χρόνο $O(1)$
 - **ανάκτηση (και διαγραφή)** στοιχείου μέγιστης προτεραιότητας: χρόνος $O(n)$
- Εναλλακτικά (list, array):
 - εισαγωγή σε χρόνο $O(n)$
 - **ανάκτηση (και διαγραφή)** στοιχείου μέγιστης προτεραιότητας: χρόνος $O(1)$

Σωρός (heap)

- Ένας καλύτερος τρόπος υλοποίησης ουράς προτεραιότητας
- **Δυαδικός σωρός** (binary heap):
 - πλήρες δυαδικό δένδρο
 - ελάχιστο (ή μέγιστο) στοιχείο στην ρίζα
 - κάθε υποδένδρο είναι επίσης δυαδικός σωρός
 - απλή υλοποίηση με **πίνακα**

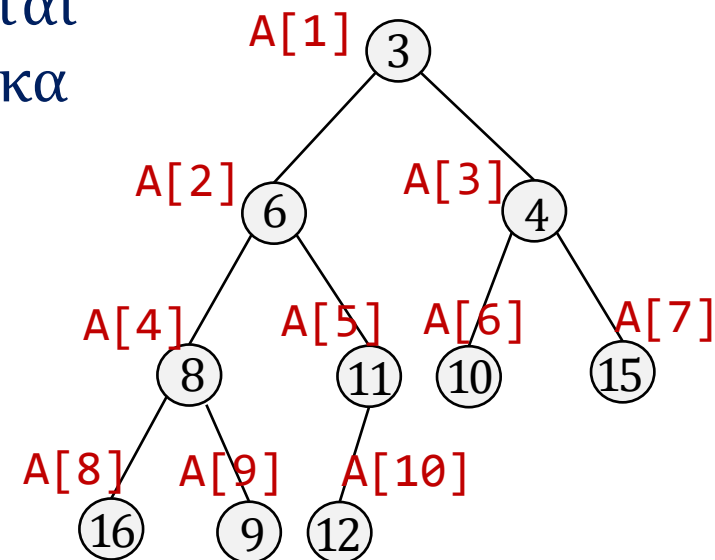


Δυαδικός σωρός
ελαχίστου

Υλοποίηση σωρού με πίνακα

- Παιδιά του $A[i]$: $[2i], A[2i+1]$
- Γονέας του $A[i]$: $A[i/2]$
- Τα στοιχεία του σωρού γράφονται σε συνεχόμενες θέσεις του πίνακα σε σειρά αντίστοιχη της εξερεύνησης BFS

Δυαδικός σωρός
ελαχίστου

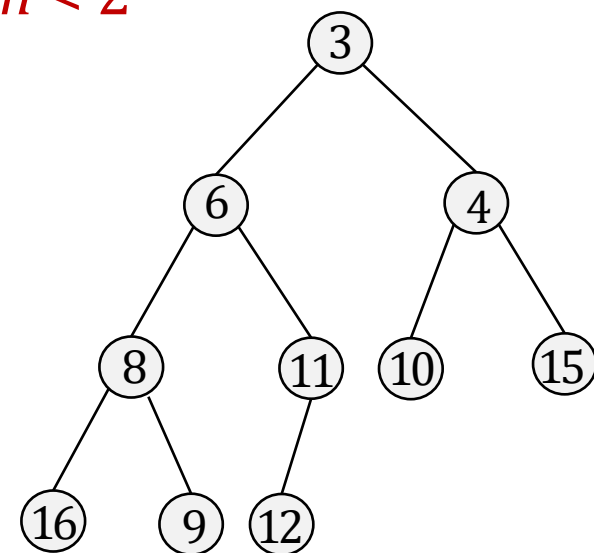


A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
3	6	4	8	11	10	15	16	9	12

Σημαντικές ιδιότητες δυαδικών σωρών



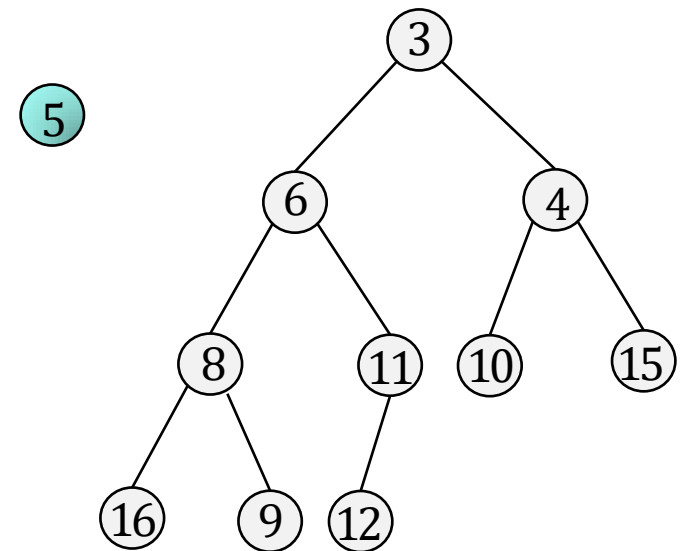
- ύψος δένδρου: $O(\log_2 n)$
#κόμβων σε δένδρο ύψους h : $2^h \leq n < 2^{h+1}$
- άμεση ανάκτηση στοιχείου μέγιστης προτεραιότητας: $O(1)$
- εισαγωγή και διαγραφή σε χρόνο $O(\log_2 n)$



Δυαδικός σωρός
ελαχίστου

Εισαγωγή σε δυαδικό σωρό

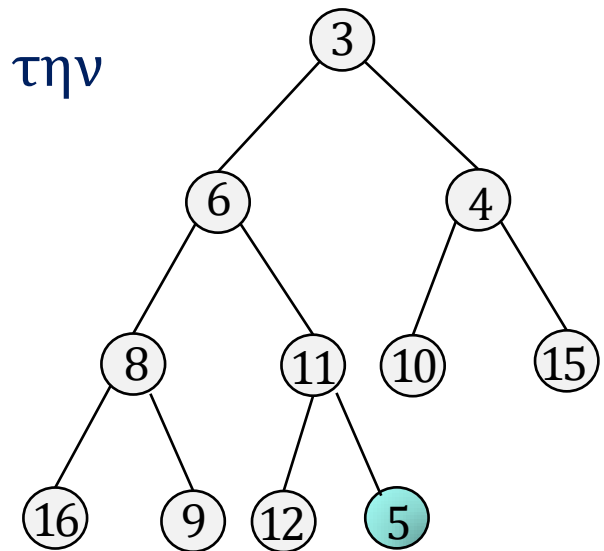
- Έστω ότι θέλουμε να εισαγάγουμε ένα νέο στοιχείο με τιμή 5



**Δυαδικός σωρός
ελαχίστου**

Εισαγωγή σε δυαδικό σωρό

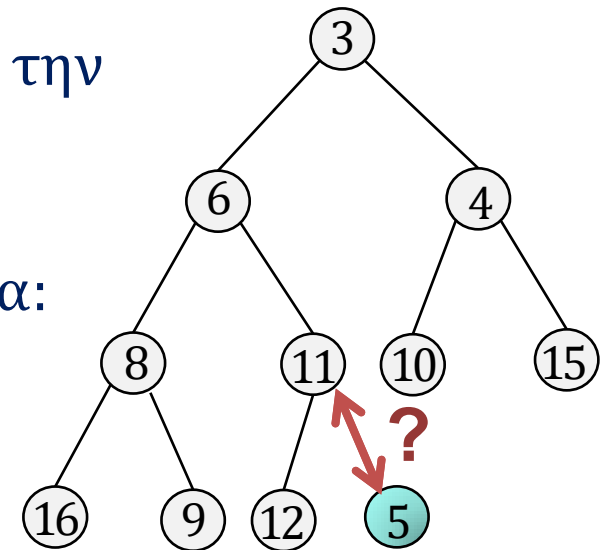
- Έστω ότι θέλουμε να εισαγάγουμε ένα νέο στοιχείο με τιμή 5
- Αρχικά τοποθετούμε το στοιχείο μετά την τελευταία θέση ($A[n+1]$)



Δυαδικός σωρός
ελαχίστου

Εισαγωγή σε δυαδικό σωρό

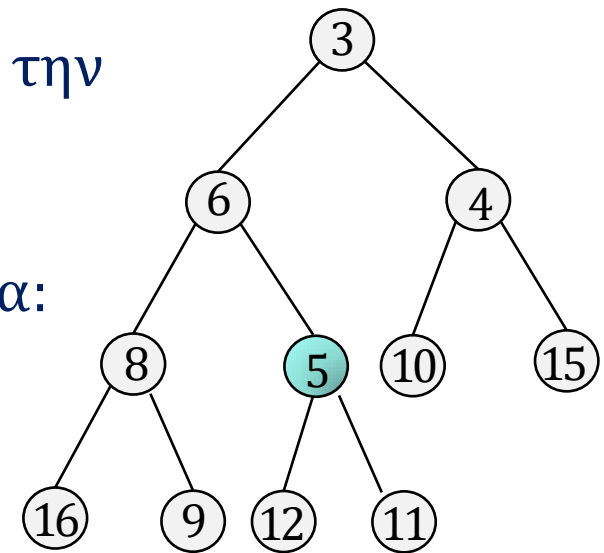
- Έστω ότι θέλουμε να εισαγάγουμε ένα νέο στοιχείο με τιμή 5
- Αρχικά τοποθετούμε το στοιχείο μετά την τελευταία θέση ($A[n+1]$)
- Συγκρίνουμε με τον γονέα και αν χρειάζεται ανταλλάσσουμε τα στοιχεία:
 $\text{swap}(A[i], A[i/2])$



Δυαδικός σωρός
ελαχίστου

Εισαγωγή σε δυαδικό σωρό

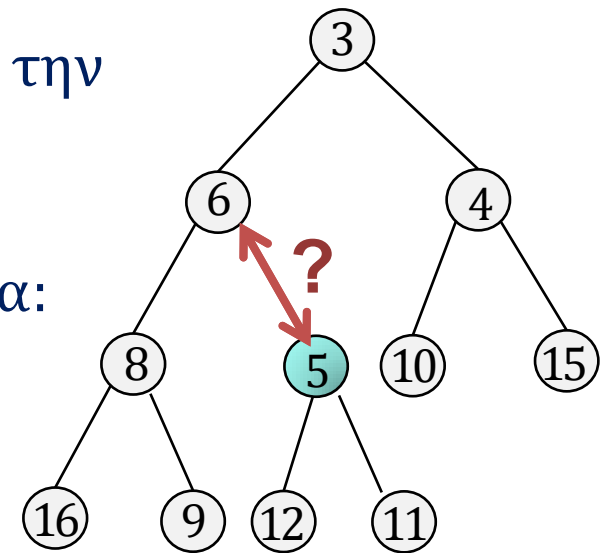
- Έστω ότι θέλουμε να εισαγάγουμε ένα νέο στοιχείο με τιμή 5
- Αρχικά τοποθετούμε το στοιχείο μετά την τελευταία θέση ($A[11]$)
- Συγκρίνουμε με τον γονέα και αν χρειάζεται ανταλλάσσουμε τα στοιχεία:
 $\text{swap}(A[i], A[i/2])$
- Επαναλαμβάνουμε ...



Δυαδικός σωρός
ελαχίστου

Εισαγωγή σε δυαδικό σωρό

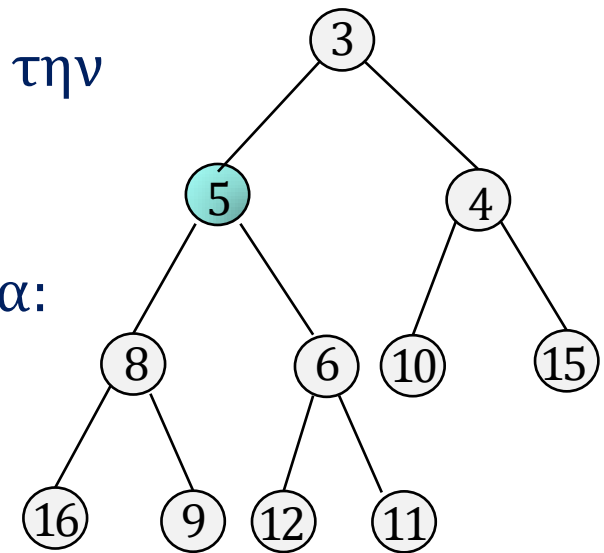
- Έστω ότι θέλουμε να εισαγάγουμε ένα νέο στοιχείο με τιμή 5
- Αρχικά τοποθετούμε το στοιχείο μετά την τελευταία θέση ($A[11]$)
- Συγκρίνουμε με τον γονέα και αν χρειάζεται ανταλλάσσουμε τα στοιχεία:
 $\text{swap}(A[i], A[i/2])$
- Επαναλαμβάνουμε ...



Δυαδικός σωρός
ελαχίστου

Εισαγωγή σε δυαδικό σωρό

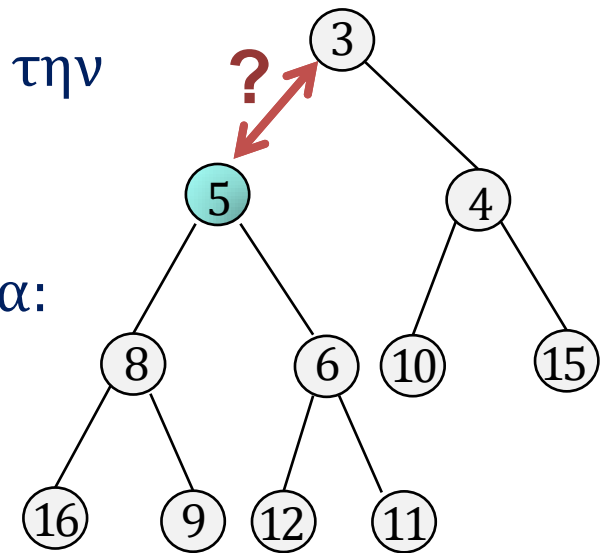
- Έστω ότι θέλουμε να εισαγάγουμε ένα νέο στοιχείο με τιμή 5
- Αρχικά τοποθετούμε το στοιχείο μετά την τελευταία θέση ($A[11]$)
- Συγκρίνουμε με τον γονέα και αν χρειάζεται ανταλλάσσουμε τα στοιχεία:
 $\text{swap}(A[i], A[i/2])$
- Επαναλαμβάνουμε ...



Δυαδικός σωρός
ελαχίστου

Εισαγωγή σε δυαδικό σωρό

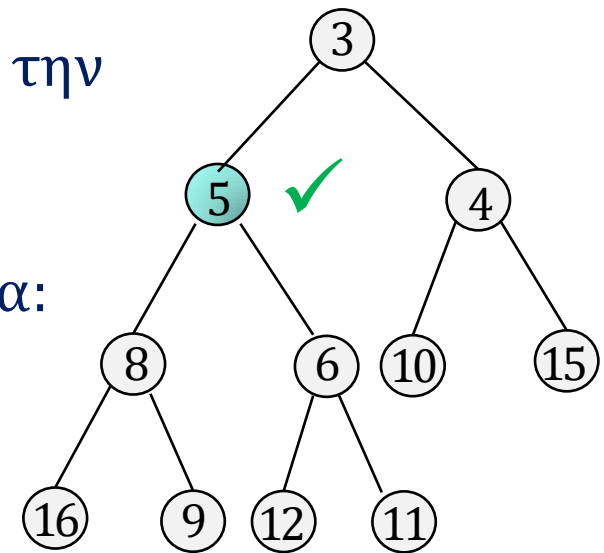
- Έστω ότι θέλουμε να εισαγάγουμε ένα νέο στοιχείο με τιμή 5
- Αρχικά τοποθετούμε το στοιχείο μετά την τελευταία θέση ($A[11]$)
- Συγκρίνουμε με τον γονέα και αν χρειάζεται ανταλλάσσουμε τα στοιχεία:
 $\text{swap}(A[i], A[i/2])$
- Επαναλαμβάνουμε ...



Δυαδικός σωρός
ελαχίστου

Εισαγωγή σε δυαδικό σωρό

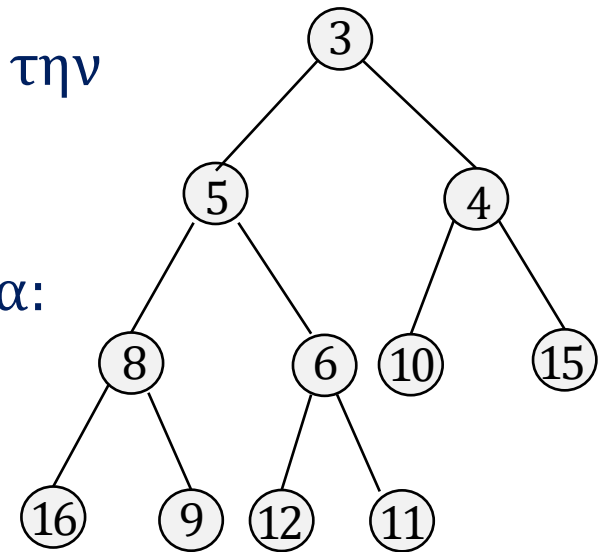
- Έστω ότι θέλουμε να εισαγάγουμε ένα νέο στοιχείο με τιμή 5
- Αρχικά τοποθετούμε το στοιχείο μετά την τελευταία θέση ($A[n+1]$)
- Συγκρίνουμε με τον γονέα και αν χρειάζεται ανταλλάσσουμε τα στοιχεία:
 $\text{swap}(A[i], A[i/2])$
- Επαναλαμβάνουμε ...



Δυαδικός σωρός
ελαχίστου

Εισαγωγή σε δυαδικό σωρό

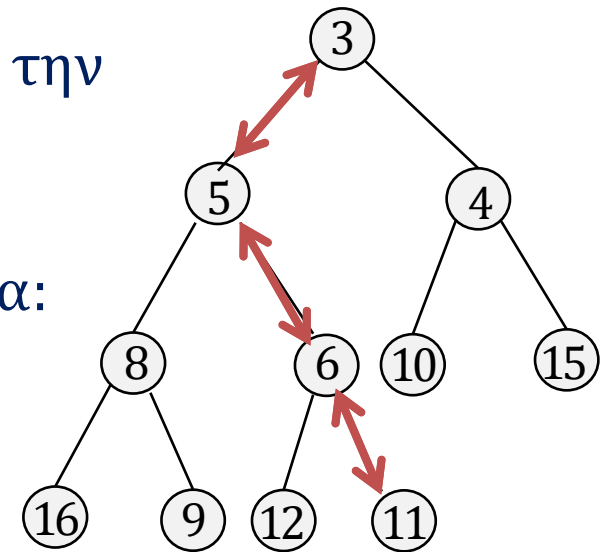
- Έστω ότι θέλουμε να εισαγάγουμε ένα νέο στοιχείο με τιμή 5
- Αρχικά τοποθετούμε το στοιχείο μετά την τελευταία θέση ($A[n+1]$)
- Συγκρίνουμε με τον γονέα και αν χρειάζεται ανταλλάσσουμε τα στοιχεία:
 $\text{swap}(A[i], A[i/2])$
- Επαναλαμβάνουμε ...
- Χρόνος εισαγωγής: $O(\log n)$ – γιατί;



Δυαδικός σωρός
ελαχίστου

Εισαγωγή σε δυαδικό σωρό

- Έστω ότι θέλουμε να εισαγάγουμε ένα νέο στοιχείο με τιμή 5
- Αρχικά τοποθετούμε το στοιχείο μετά την τελευταία θέση ($A[n+1]$)
- Συγκρίνουμε με τον γονέα και αν χρειάζεται ανταλλάσσουμε τα στοιχεία:
 $\text{swap}(A[i], A[i/2])$
- Επαναλαμβάνουμε ...
- Χρόνος εισαγωγής: $O(\log n) \sim$ ύψος δένδρου

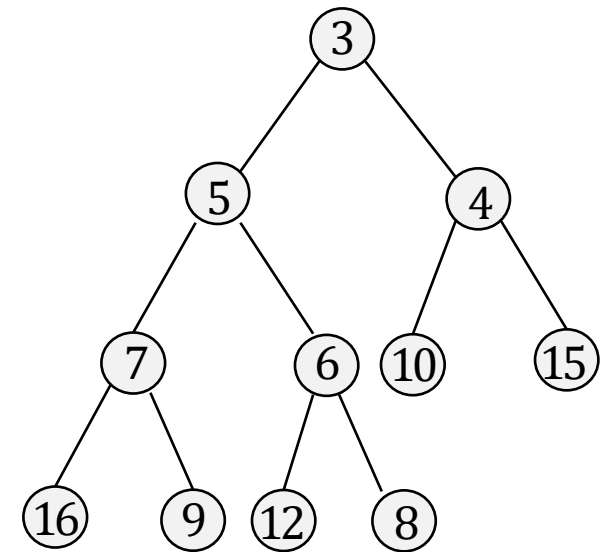


Δυαδικός σωρός
ελαχίστου

Μικρή βελτίωση: δεν χρειάζεται να κάνουμε swap , απλά
 $A[i] \leftarrow A[i/2]$ ώσπου να “αδειάσει” η σωστή θέση

Διαγραφή από δυαδικό σωρό

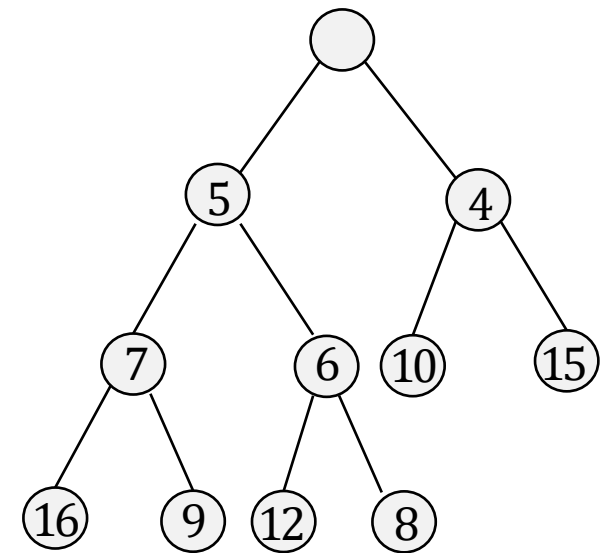
- Έστω ότι θέλουμε να διαγράψουμε το ελάχιστο στοιχείο (ρίζα)



Δυαδικός σωρός
ελαχίστου

Διαγραφή από δυαδικό σωρό

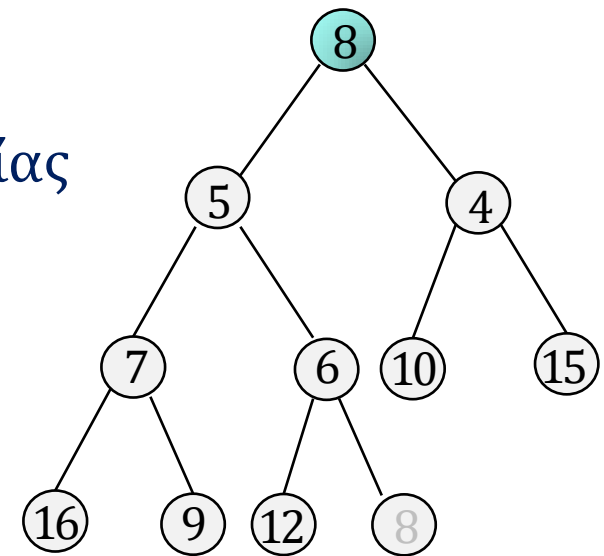
- Έστω ότι θέλουμε να διαγράψουμε το ελάχιστο στοιχείο (ρίζα)
- Η τελευταία θέση «περισσεύει»



**Δυαδικός σωρός
ελαχίστου**

Διαγραφή από δυαδικό σωρό

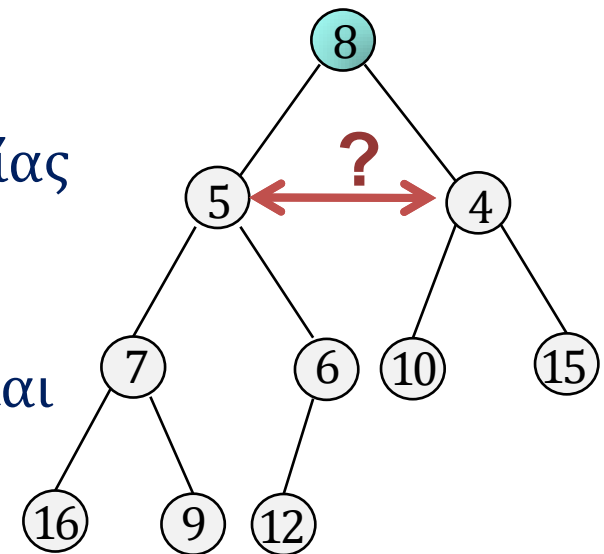
- Έστω ότι θέλουμε να διαγράψουμε το ελάχιστο στοιχείο (ρίζα)
- Η τελευταία θέση «περισσεύει»
- Τοποθετούμε το στοιχείο της τελευταίας θέσης ($A[n]$) στη ρίζα



Δυαδικός σωρός
ελαχίστου

Διαγραφή από δυαδικό σωρό

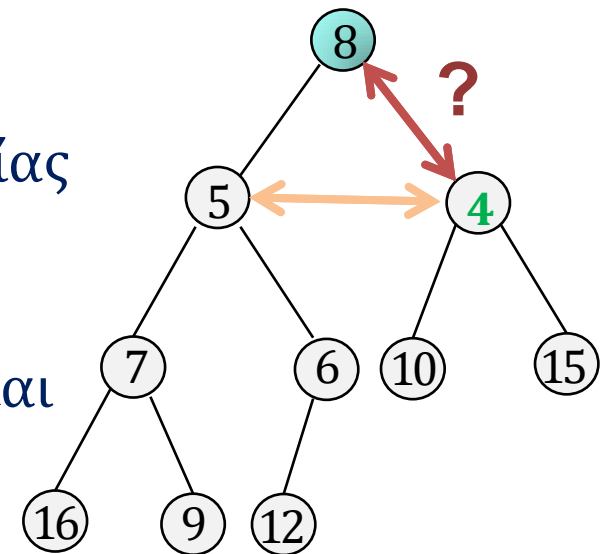
- Έστω ότι θέλουμε να διαγράψουμε το ελάχιστο στοιχείο (ρίζα)
- Η τελευταία θέση «περισσεύει»
- Τοποθετούμε το στοιχείο της τελευταίας θέσης ($A[n]$) στη ρίζα
- «Κυλάμε» το στοιχείο προς τα κάτω συγκρίνοντας με το μικρότερο παιδί και ανταλλάσσοντας (swap) αν χρειάζεται



Δυαδικός σωρός
ελαχίστου

Διαγραφή από δυαδικό σωρό

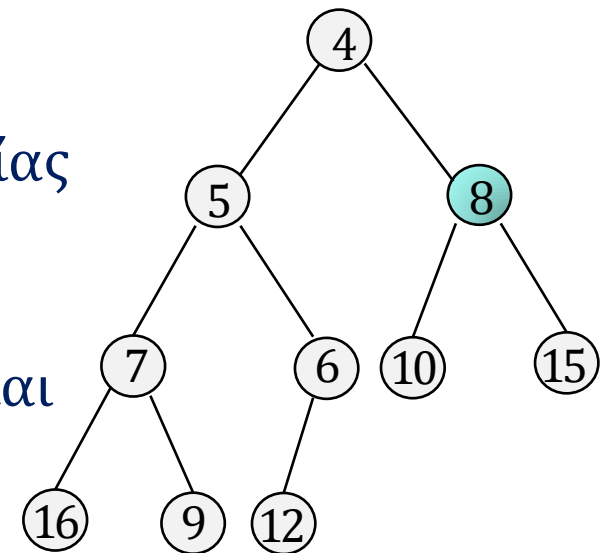
- Έστω ότι θέλουμε να διαγράψουμε το ελάχιστο στοιχείο (ρίζα)
- Η τελευταία θέση «περισσεύει»
- Τοποθετούμε το στοιχείο της τελευταίας θέσης ($A[n]$) στη ρίζα
- «Κυλάμε» το στοιχείο προς τα κάτω συγκρίνοντας με το μικρότερο παιδί και ανταλλάσσοντας (swap) αν χρειάζεται



Δυαδικός σωρός
ελαχίστου

Διαγραφή από δυαδικό σωρό

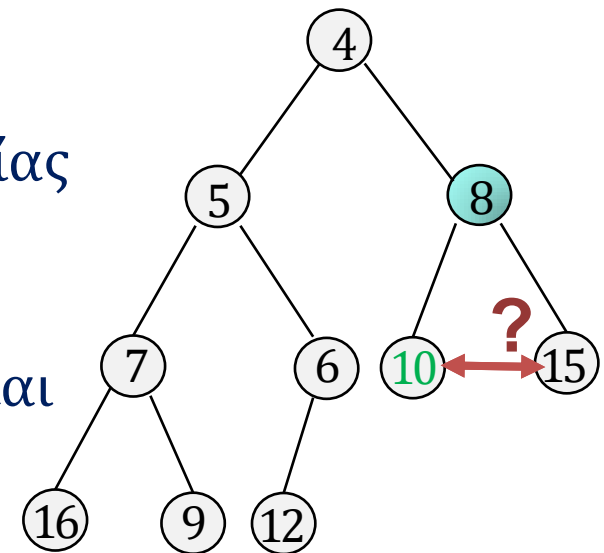
- Έστω ότι θέλουμε να διαγράψουμε το ελάχιστο στοιχείο (ρίζα)
- Η τελευταία θέση «περισσεύει»
- Τοποθετούμε το στοιχείο της τελευταίας θέσης ($A[n]$) στη ρίζα
- «Κυλάμε» το στοιχείο προς τα κάτω συγκρίνοντας με το μικρότερο παιδί και ανταλλάσσοντας (swap) αν χρειάζεται



Δυαδικός σωρός
ελαχίστου

Διαγραφή από δυαδικό σωρό

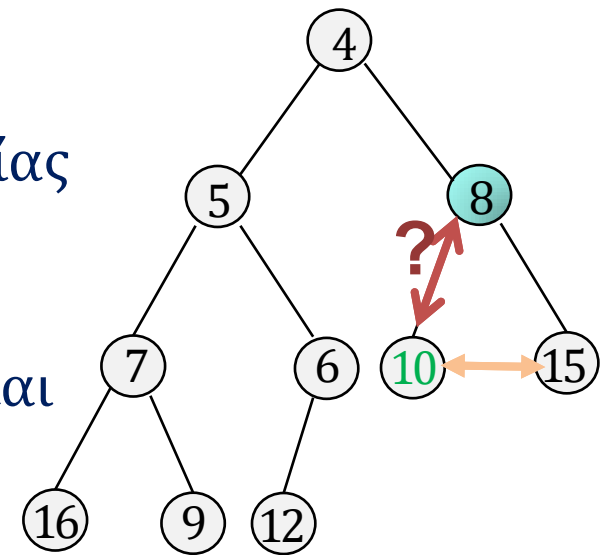
- Έστω ότι θέλουμε να διαγράψουμε το ελάχιστο στοιχείο (ρίζα)
- Η τελευταία θέση «περισσεύει»
- Τοποθετούμε το στοιχείο της τελευταίας θέσης ($A[n]$) στη ρίζα
- «Κυλάμε» το στοιχείο προς τα κάτω συγκρίνοντας με το μικρότερο παιδί και ανταλλάσσοντας (swap) αν χρειάζεται
- Επαναλαμβάνουμε ...



Δυαδικός σωρός
ελαχίστου

Διαγραφή από δυαδικό σωρό

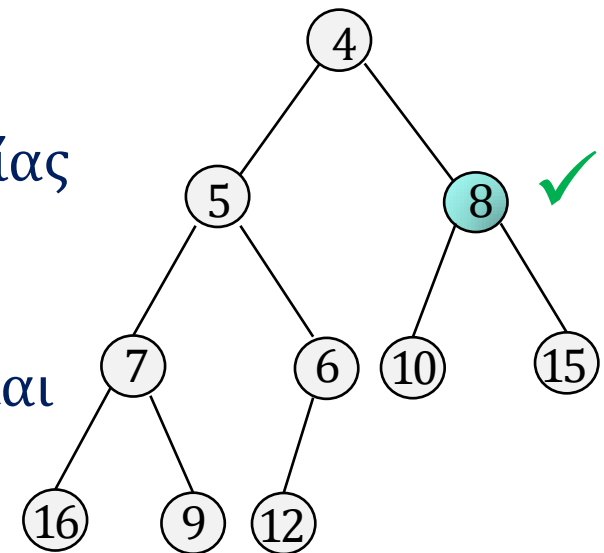
- Έστω ότι θέλουμε να διαγράψουμε το ελάχιστο στοιχείο (ρίζα)
- Η τελευταία θέση «περισσεύει»
- Τοποθετούμε το στοιχείο της τελευταίας θέσης ($A[n]$) στη ρίζα
- «Κυλάμε» το στοιχείο προς τα κάτω συγκρίνοντας με το μικρότερο παιδί και ανταλλάσσοντας (swap) αν χρειάζεται
- Επαναλαμβάνουμε ...



Δυαδικός σωρός
ελαχίστου

Διαγραφή από δυαδικό σωρό

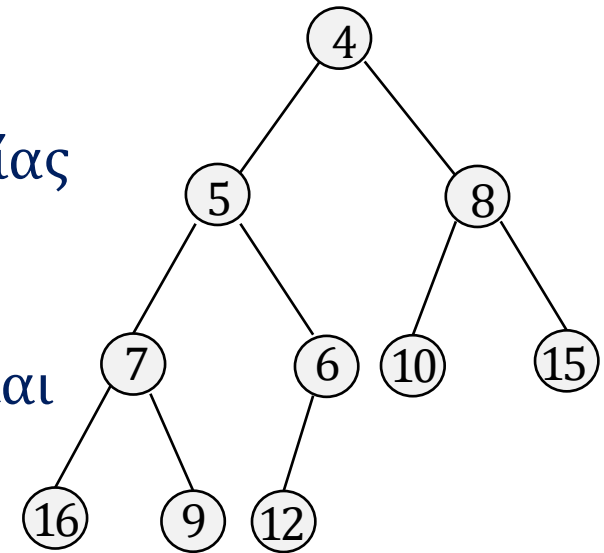
- Έστω ότι θέλουμε να διαγράψουμε το ελάχιστο στοιχείο (ρίζα)
- Η τελευταία θέση «περισσεύει»
- Τοποθετούμε το στοιχείο της τελευταίας θέσης ($A[n]$) στη ρίζα
- «Κυλάμε» το στοιχείο προς τα κάτω συγκρίνοντας με το μικρότερο παιδί και ανταλλάσσοντας (swap) αν χρειάζεται
- Επαναλαμβάνουμε ...



Δυαδικός σωρός
ελαχίστου

Διαγραφή από δυαδικό σωρό

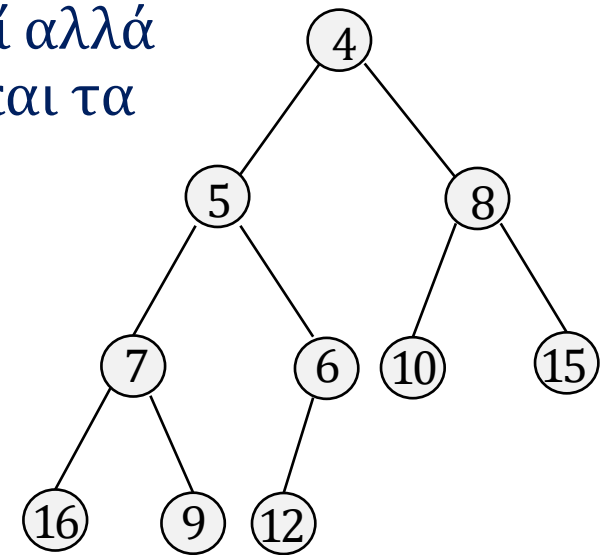
- Έστω ότι θέλουμε να διαγράψουμε το ελάχιστο στοιχείο (ρίζα)
- Η τελευταία θέση «περισσεύει»
- Τοποθετούμε το στοιχείο της τελευταίας θέσης ($A[n]$) στη ρίζα
- «Κυλάμε» το στοιχείο προς τα κάτω συγκρίνοντας με το μικρότερο παιδί και ανταλλάσσοντας (swap) αν χρειάζεται
- Επαναλαμβάνουμε ...
- Χρόνος ανακατασκευής: $O(\log n)$ – γιατί;



Δυαδικός σωρός
ελαχίστου

Διαγραφή από δυαδικό σωρό => μέθοδος (ανα)κατασκευής

- Η διαδικασία διαγραφής ουσιαστικά εμπεριέχει διαδικασία **ανακατασκευής** του σωρού όταν τα υποδένδρα είναι σωροί αλλά η ρίζα είναι μεγαλύτερη από το ένα ή και τα δύο παιδιά της
- Αυτή η διαδικασία λέγεται **Heapify** ή **Combine** στη βιβλιογραφία και είναι χρήσιμη γενικότερα
- Για παράδειγμα, μπορούμε με χρήση της διαδικασίας αυτής να κατασκευάσουμε σωρό με n στοιχεία σε **συνολικό χρόνο $O(n)$** (αντί για **$O(n \log n)$** με χρήση διαδοχικών εισαγωγών) – πώς;



Δυαδικός σωρός
ελαχίστου

Ταξινόμηση με σωρό (HeapSort)

- Έστω ότι θέλουμε να ταξινομήσουμε n αριθμούς (ή στοιχεία)
- Τα εισάγουμε σε σωρό με n διαδοχικές εισαγωγές: $O(n \log n)$ [ή χρησιμοποιούμε Heapify, $O(n)$]
- Επαναλαμβάνουμε n φορές:
 - διαγραφή ελαχίστου και εκτύπωσή του
- Συνολικός χρόνος διαγραφών: $O(n \log n)$
- Πολυπλοκότητα HeapSort: $O(n \log n)$
- Εναλλακτικά: **σωρός μεγίστου**, διαγράφουμε διαδοχικά μέγιστο και το εισάγουμε στην τελευταία θέση (ταξινόμηση *επί τόπου* !)

Κατακερματισμός (Hashing)



- Έστω ότι θέλουμε να καταγράψουμε ένα πλήθος δεδομένων (ή κλειδιών / κωδικών) σε ευρετήριο (λεξικό)
- Όστε να μπορούμε να ελέγχουμε αν κάποιο κλειδί έχει εμφανιστεί στα δεδομένα μας
- Πόσο καλά μπορούμε να το κάνουμε;
- Θέλουμε γρήγορη καταχώρηση και ανάκτηση, ιδανικά **$O(1)$**
- Εφικτό με χρήση array – αλλά «σπάταλο» σε μνήμη
- Εφικτό με χρήση **συναρτήσεων κατακερματισμού (hash functions)**

Κατακερματισμός (Hashing)

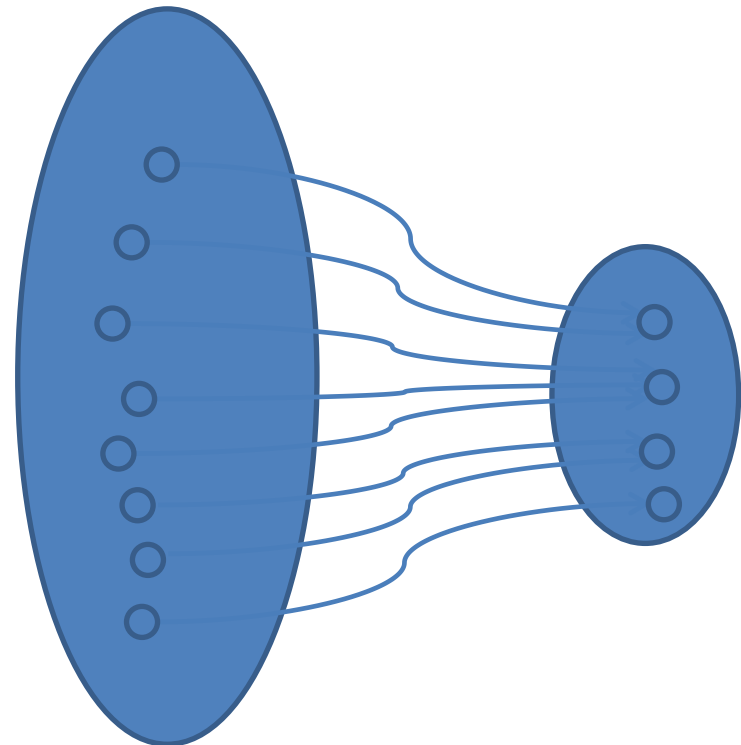


- Άλλες εφαρμογές: κατανομή φοιτητών σε αίθουσες, βιβλίων σε ράφια βιβλιοθήκης, κ.ά.
- Επιπλέον: έστω ότι θέλουμε να ταξινομήσουμε n αριθμούς
- Οι αριθμοί εισόδου μπορεί να είναι πολύ μεγάλοι
- Μπορούμε σε **$O(n)$** ;
- Ιδέες από **bucket sort**

Κατακερματισμός (Hashing)



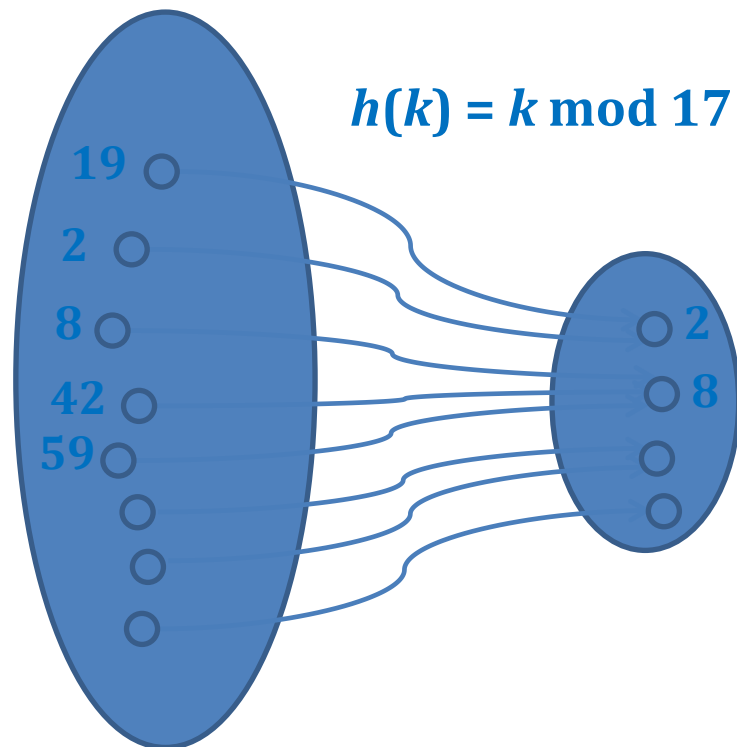
- Απεικόνιση ενός μεγάλου συνόλου σε ένα πολύ μικρότερο
- Όσο το δυνατόν πιο ομοιόμορφα
- Στόχος: ελαχιστοποίηση συγκρούσεων
- (Διαφορά με κρυπτογραφικό hashing: εκεί δεν θέλουμε συγκρούσεις)



Κατακερματισμός (Hashing)

■ Μέθοδος διαίρεσης

- Απλή μορφή: $h(k) = k \bmod m$



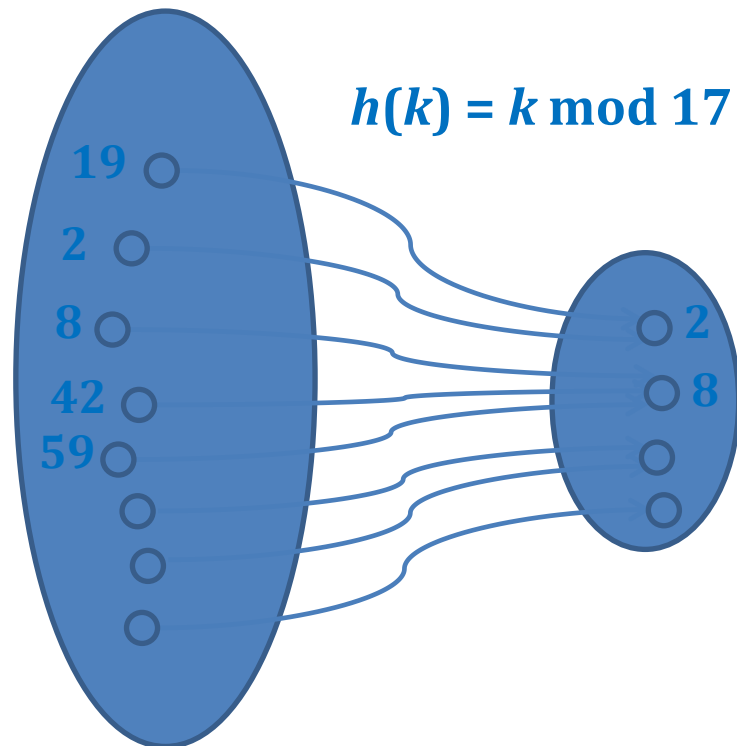
Κατακερματισμός (Hashing)

Μέθοδος διαίρεσης

- Απλή μορφή: $h(k) = k \bmod m$
- Οικογένεια συναρτήσεων:
 $h(k) = ak + b \bmod m$
- Μειονέκτημα: δεν έχει ιδιότητα **καθολικότητας**

Καθολική (universal)
συνάρτηση κατακερ/σμού:

τυχαία επιλογή a, b ,
απεικονίζει διαφορετικά
κλειδιά στην ίδια θέση με
πιθανότητα $\leq 1/m$



Κατακερματισμός (Hashing)

Μέθοδος πολλαπλασιασμού

- Απλή μορφή: $h(k) = (ak \bmod 2^w) \div 2^{w-r}$ $[m = 2^r]$

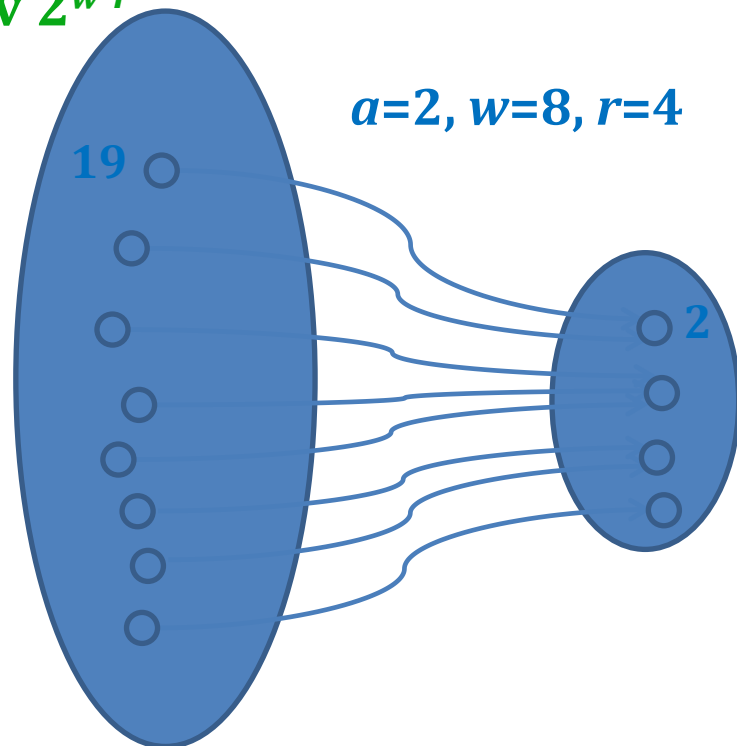
- Οικογένεια συναρτήσεων:

$$h(k) = (ak + b \bmod 2^w) \div 2^{w-r}$$

- Πλεονεκτήματα:

- εύκολη υλοποίηση
- ιδιότητα **2-καθολικότητας**

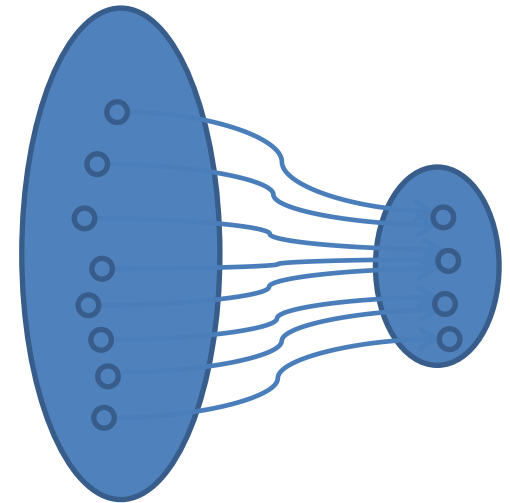
τυχαία επιλογή a, b ,
απεικονίζει διαφορετικά
κλειδιά στην ίδια θέση με
πιθανότητα $\leq 2/m$



Κατακερματισμός (Hashing)

• Άμεση / κλειστή διευθυνσιοδότηση (direct / closed addressing)

- Κάθε κλειδί απεικονίζεται στην ίδια πάντα θέση
- Συγκρούσεις αντιμετωπίζονται με αλυσίδωση
- Μέθοδος διαίρεσης, μέθοδος πολλαπλασιασμού
- Σημαντικό μέγεθος: παράγοντας φόρτου
$$a = n / m > 1$$
- Πολυπλοκότητα εισαγωγής, αναζήτησης, διαγραφής:
 $O(1+a)$ [αναμεν/νος # βημάτων]



Κατακερματισμός (Hashing)

• **Ανοιχτή διευθυνσιοδότηση (open addressing)**

- Αποφυγή συγκρούσεων με κατάλληλη αλλαγή θέσης
- Γραμμική και τετραγωνική βολιδοσκόπηση, διπλός κατακερματισμός, μέθοδος «κούκου»
- Σημαντικό μέγεθος: **παράγοντας φόρτου**
$$a = n / m < 1$$
- Πολυπλοκότητα εισαγωγής, αναζήτησης, διαγραφής:
 $O(1/(1-a))$ [αναμ/νος # βολιδοσκ/σεων για ομοιόμορφα τυχαία βολιδοσκόπηση (διπλός κατακερματισμός την προσεγγίζει καλά)]

