Δομές Δεδομένων και Αλγόριθμοι

Χρήστος Γκόγκος

ΤΕΙ Ηπείρου

Χειμερινό Εξάμηνο 2014-2015
Παρουσίαση 17. Σωροί (Heaps)

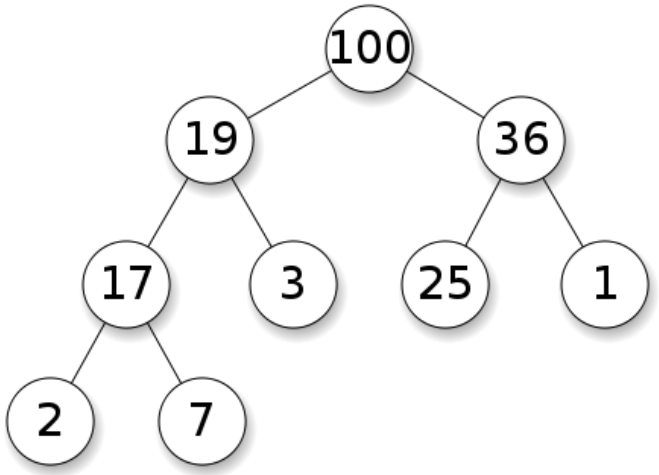
έκδοση 1.0

Heap

Σωρός

Ο σωρός είναι μια μερικά ταξινομημένη δομή δεδομένων που υποστηρίζει τις ακόλουθες λειτουργίες:

- **εύρεση** του στοιχείου με τη μεγαλύτερη τιμή κλειδιού.
- **διαγραφή** του στοιχείου με τη μεγαλύτερη τιμή κλειδιού.
- **προσθήκη** νέου κλειδιού στη δομή.



Ένας σωρός μπορεί να έχει στη κορυφή το μεγαλύτερο στοιχείο (Max-Heap) ή αντίστοιχα να έχει στη κορυφή το μικρότερο στοιχείο (Min-Heap). Στη δεύτερη περίπτωση οι λειτουργίες της εύρεσης και της διαγραφής αφορούν τη μικρότερη τιμή κλειδιού.

Πότε χρησιμοποιείται ο σωρός;

- Όταν η εφαρμογή που πρέπει να υλοποιηθεί πραγματοποιεί επαναλαμβανόμενους υπολογισμούς του μέγιστου στοιχείου.
- Μπορεί να υλοποιηθεί με κατάλληλο τρόπο έτσι ώστε να επιστρέφει αποδοτικά την k-οστή μεγαλύτερη τιμή.

Ο σωρός ως δυαδικό δένδρο

Ένας σωρός μπορεί να οριστεί ως ένα δυαδικό δένδρο για το οποίο ισχύουν οι ακόλουθοι 2 περιορισμοί:

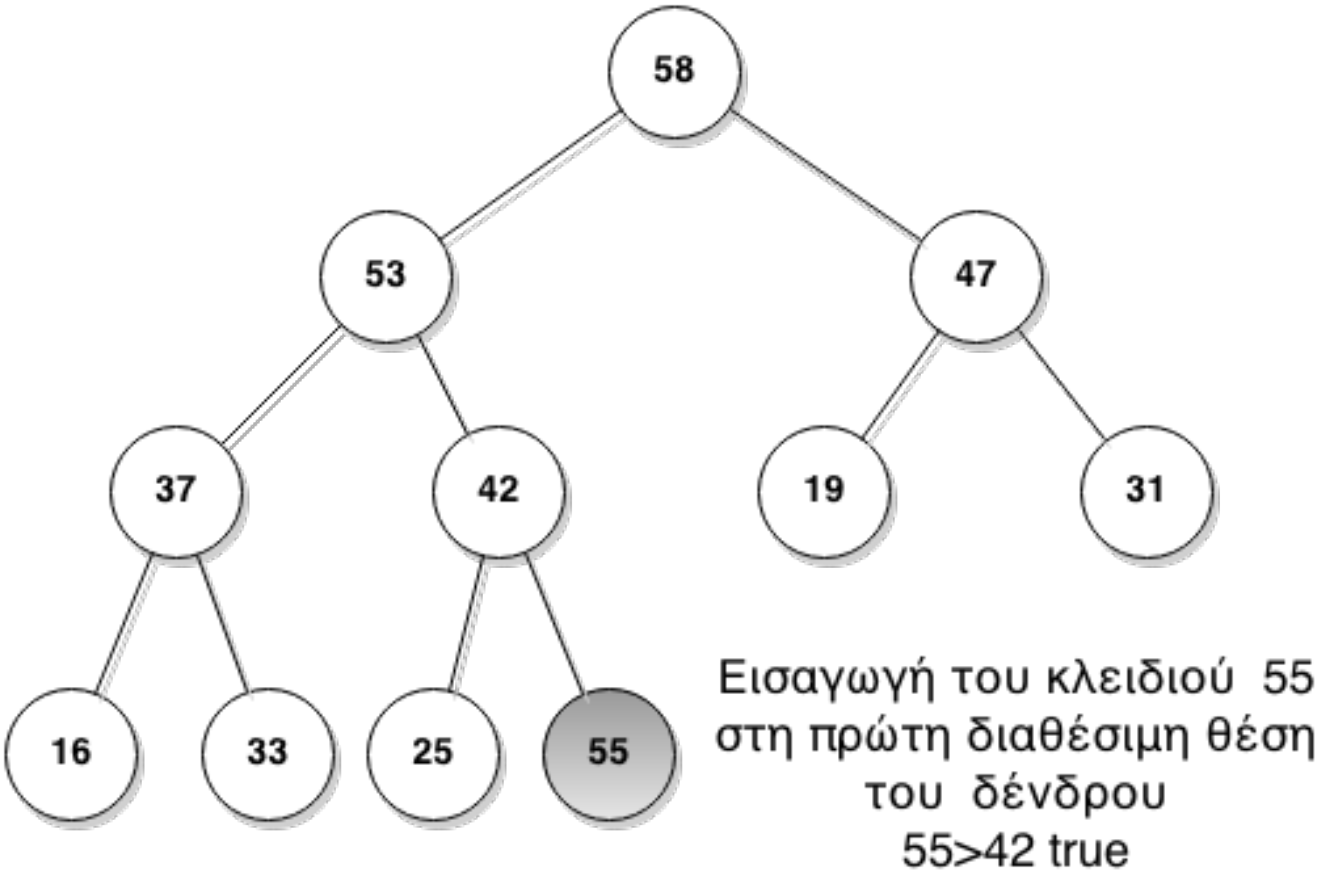
1. **πληρότητα**: το δυαδικό δένδρο είναι συμπληρωμένο, δηλαδή όλα τα επίπεδά του είναι πλήρη εκτός πιθανά από το τελευταίο επίπεδο στο οποίο μπορούν να λείπουν μόνο τα δεξιότερα φύλλα.
2. **κυριαρχία γονέα**: το κλειδί σε κάθε κορυφή είναι μεγαλύτερο ή ίσο από τα κλειδιά των παιδιών (σε Max-Heap).

Ιδιότητες του σωρού στη δενδρική του απεικόνιση

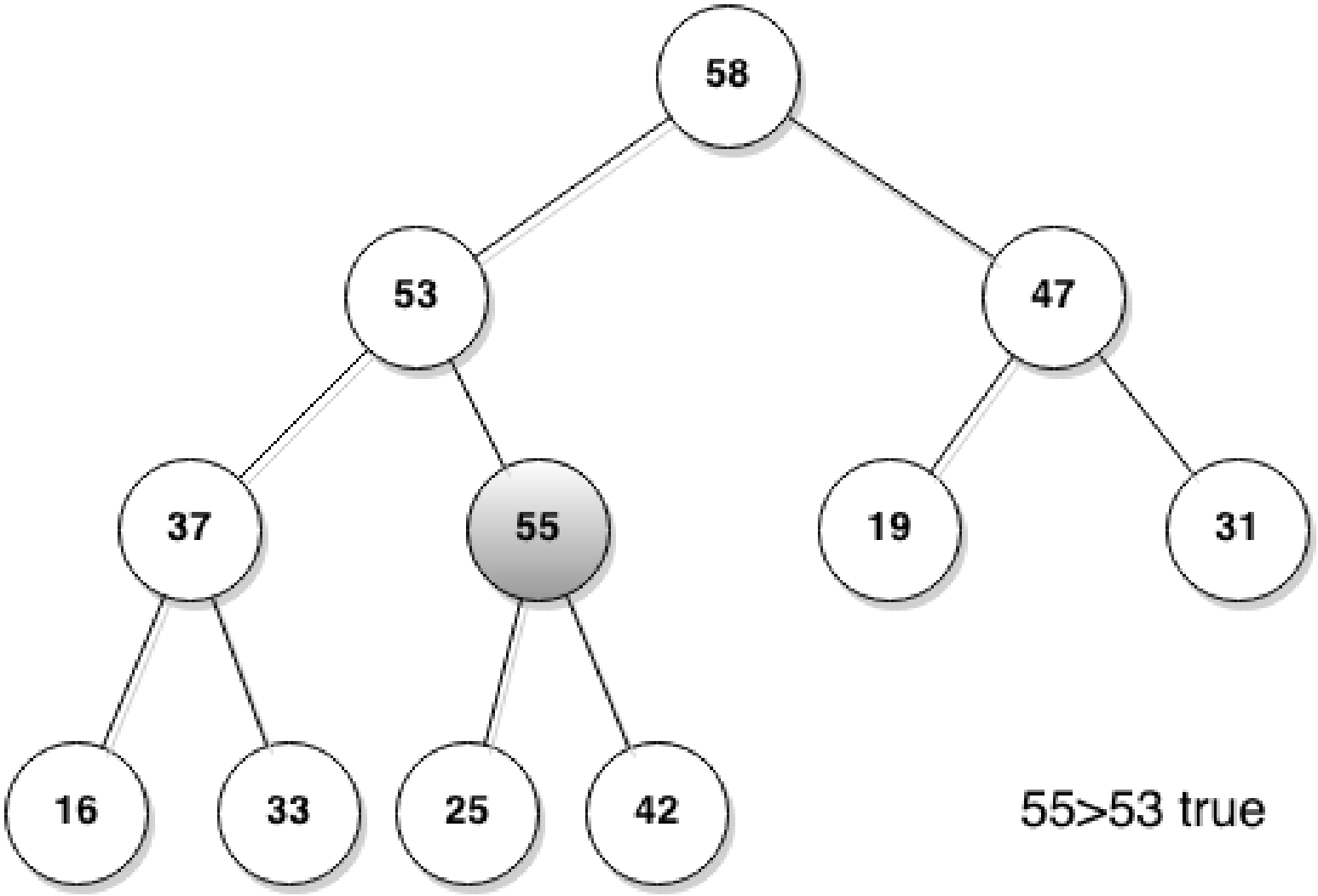
- Η ρίζα του σωρού περιέχει το μεγαλύτερο στοιχείο.
- Κάθε κόμβος του σωρού μαζί με όλους τους απογόνους του είναι και αυτός σωρός.
- Τα κλειδιά είναι ταξινομημένα από πάνω προς τα κάτω δηλαδή τα κλειδιά σε κάθε διαδρομή από την κορυφή προς τα φύλλα είτε μειώνονται είτε παραμένουν στην ίδια τιμή.
- Τα κλειδιά σε κάθε επίπεδο του δένδρου δεν είναι ταξινομημένα.
- Τα παιδιά κάθε κόμβου δεν είναι ταξινομημένα.

Εισαγωγή νέου κλειδιού στο σωρό

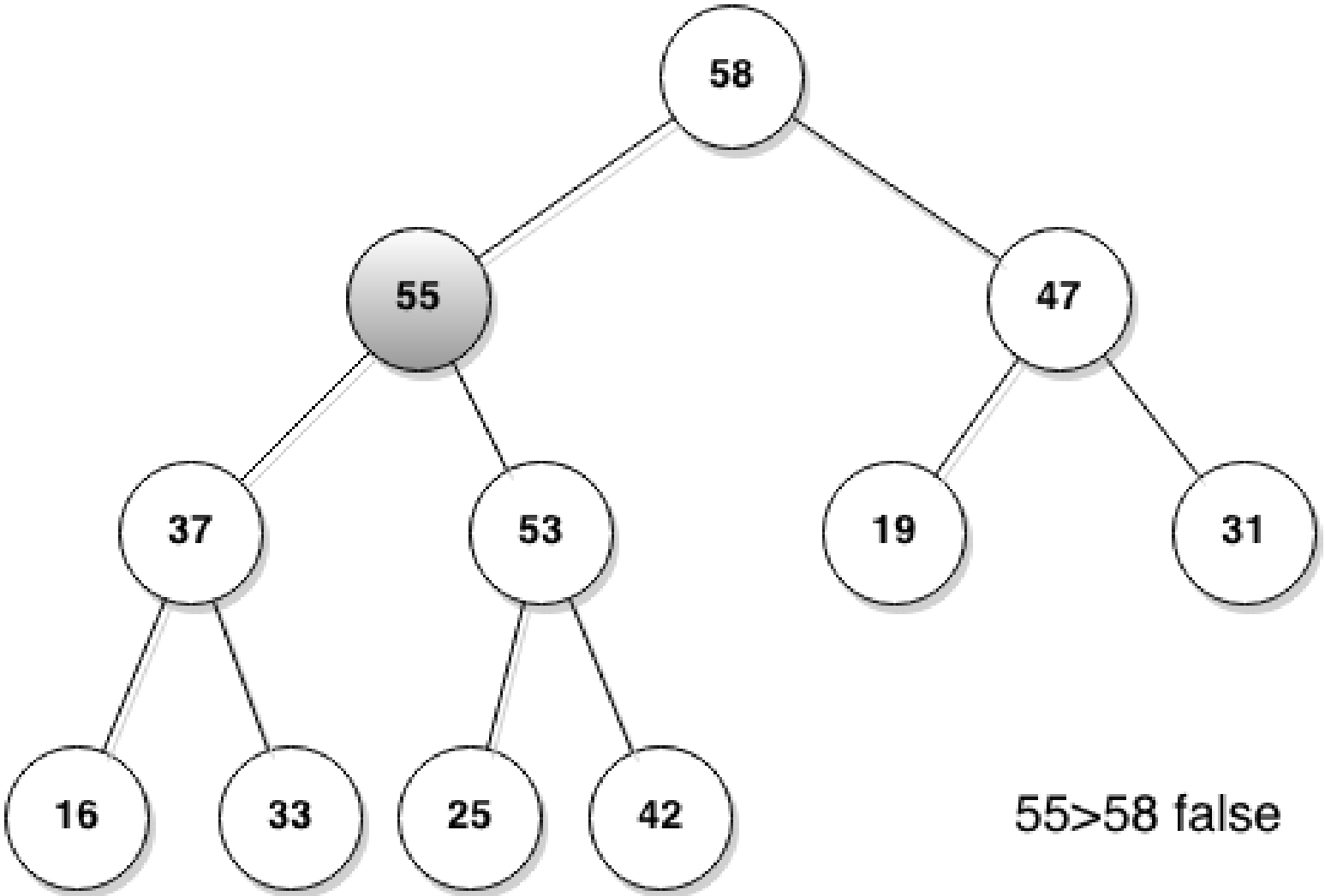
- Το νέο κλειδί εισάγεται ως φύλλο στο δένδρο όσο πιο αριστερά γίνεται.
- Το νέο κλειδί επαναληπτικά συγκρίνεται με τον γονέα του και αν είναι μεγαλύτερο τον αντικαθιστά.



Το νέο κλειδί “ανεβαίνει” προς τη σωστή θέση

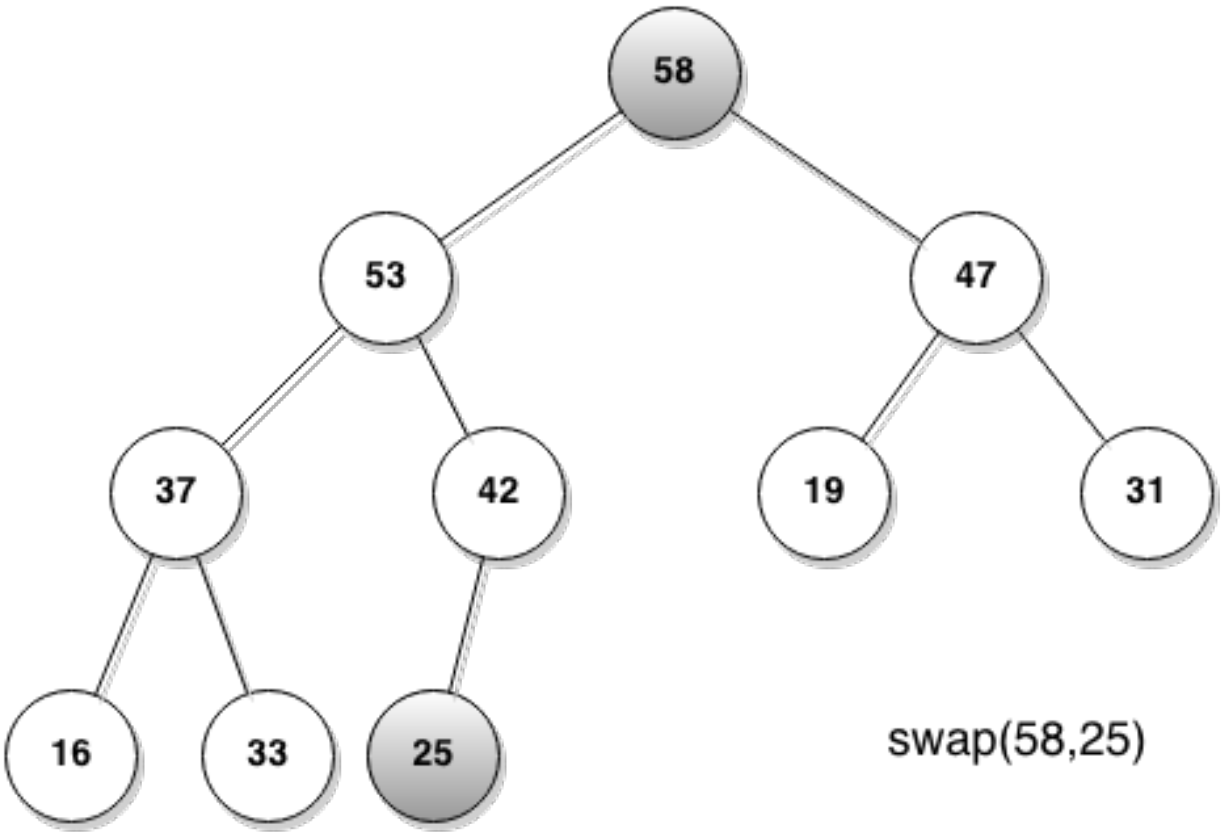


Το νέο κλειδί “ανεβαίνει” προς τη σωστή θέση

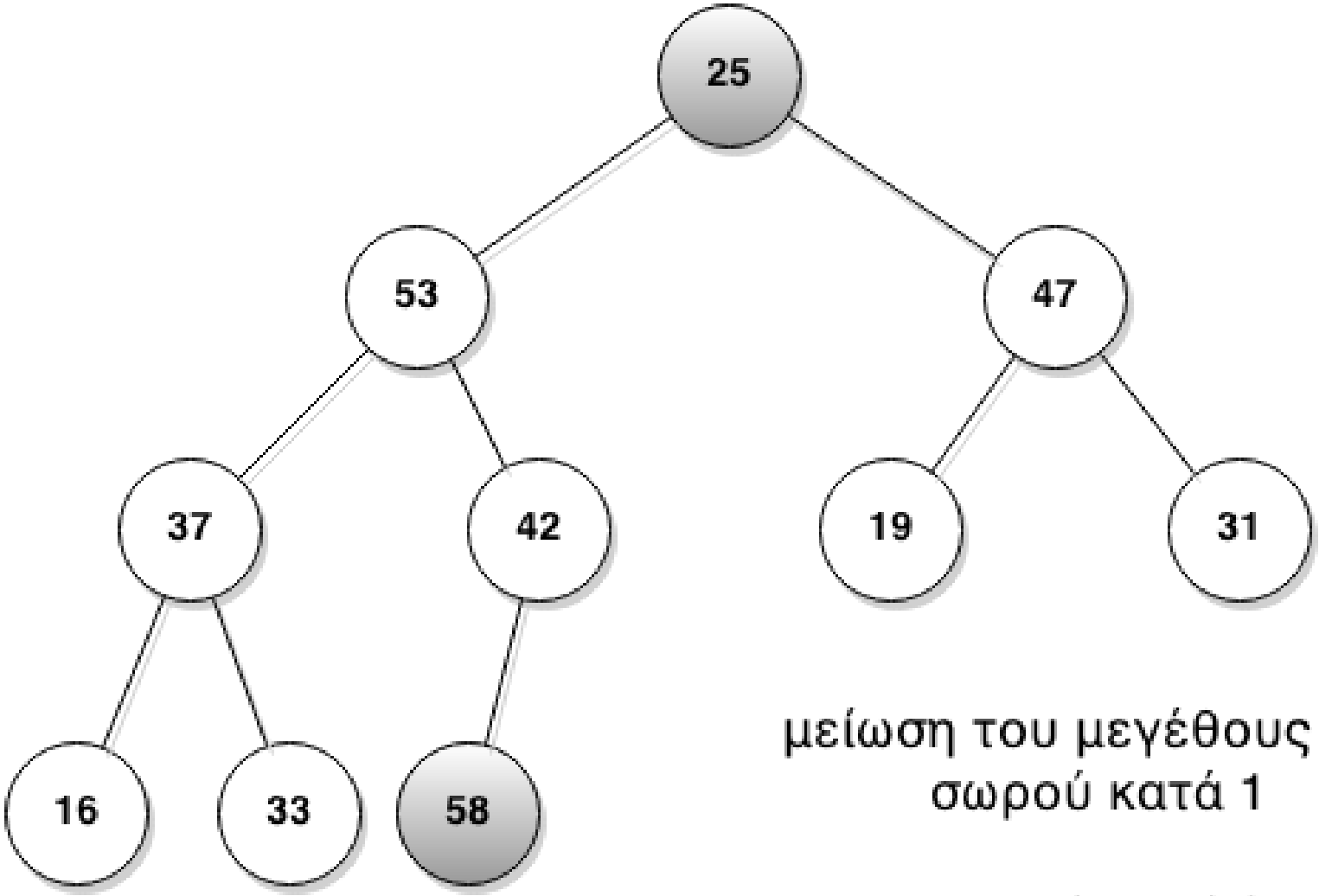


Διαγραφή της ρίζας του σωρού

- Το κλειδί που αντιμετατέθηκε με τη ρίζα συγκρίνεται με τα παιδιά του και σε περίπτωση που είναι μικρότερο αντιμετατίθεται με το μεγαλύτερο από τα παιδιά του.
- Η διαδικασία επαναλαμβάνεται μέχρι το κλειδί που αντιμετατέθηκε με τη ρίζα να είναι μεγαλύτερο ή ίσο από τα παιδιά του.



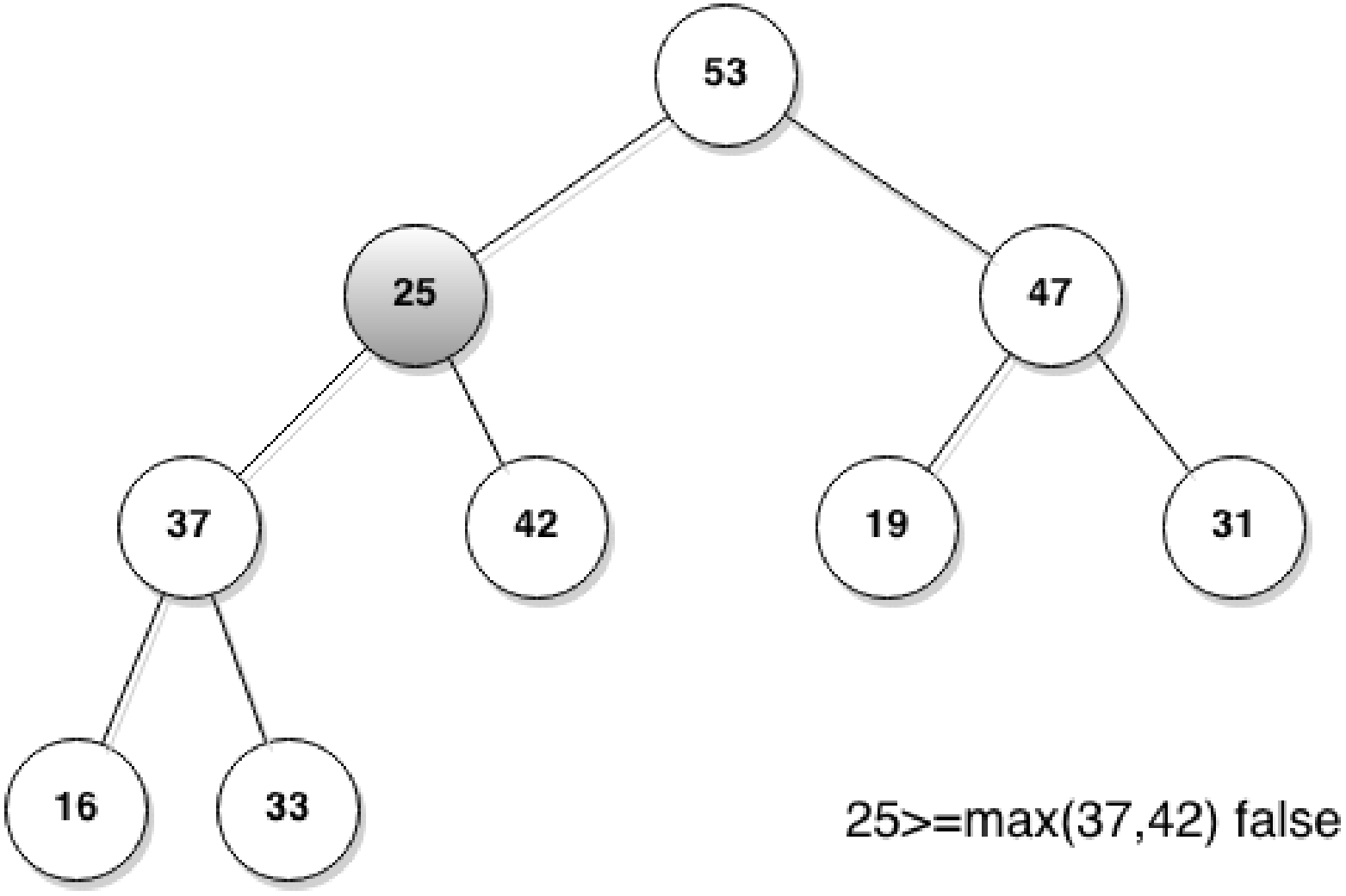
Αντιμετάθεση της ρίζας με το τελευταίο φύλλο του δένδρου



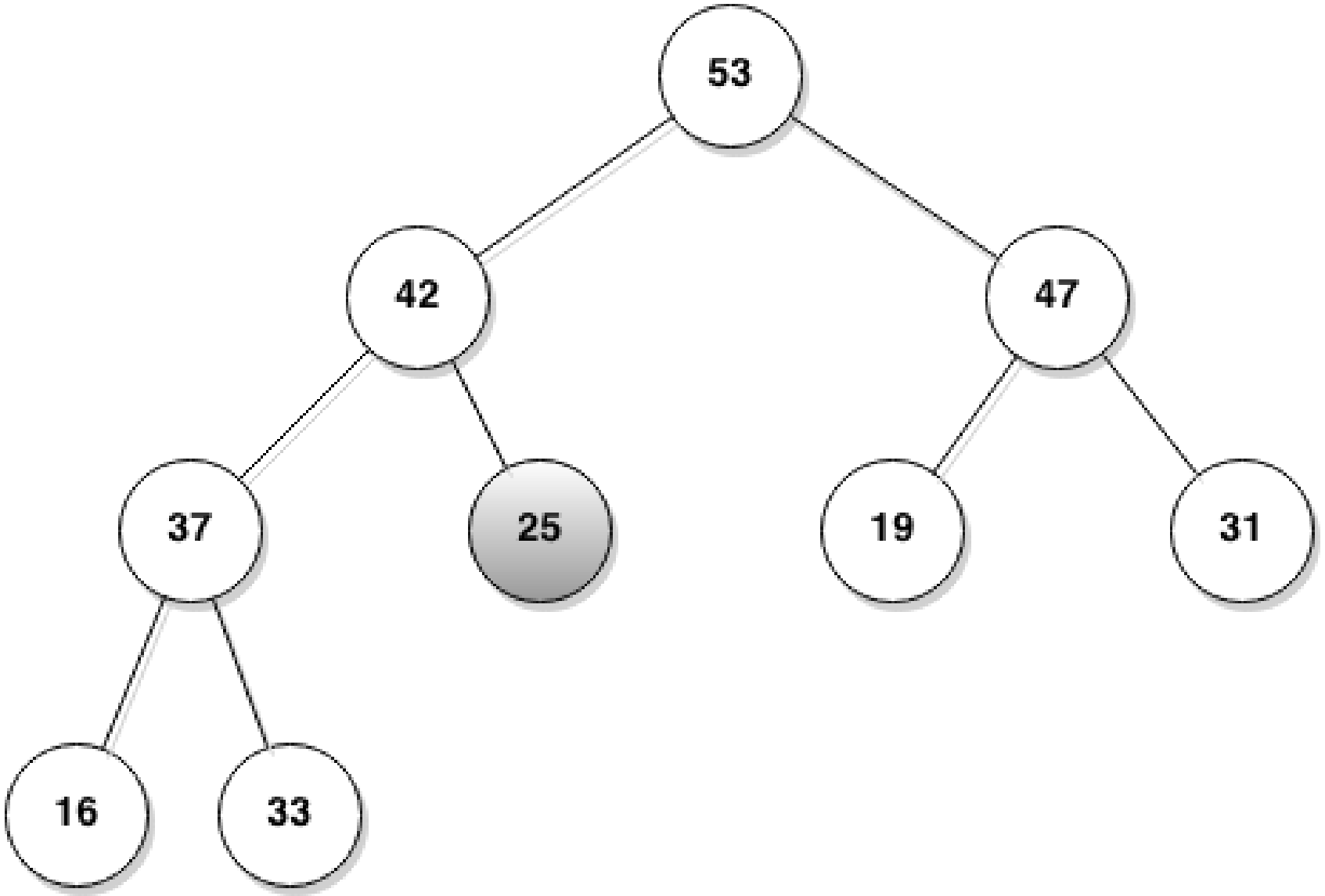
μείωση του μεγέθους του σωρού κατά 1

25>=max(53,47) false

Το κλειδί που αντικατέστησε τη ρίζα “κατεβαίνει” προς τη σωστή θέση



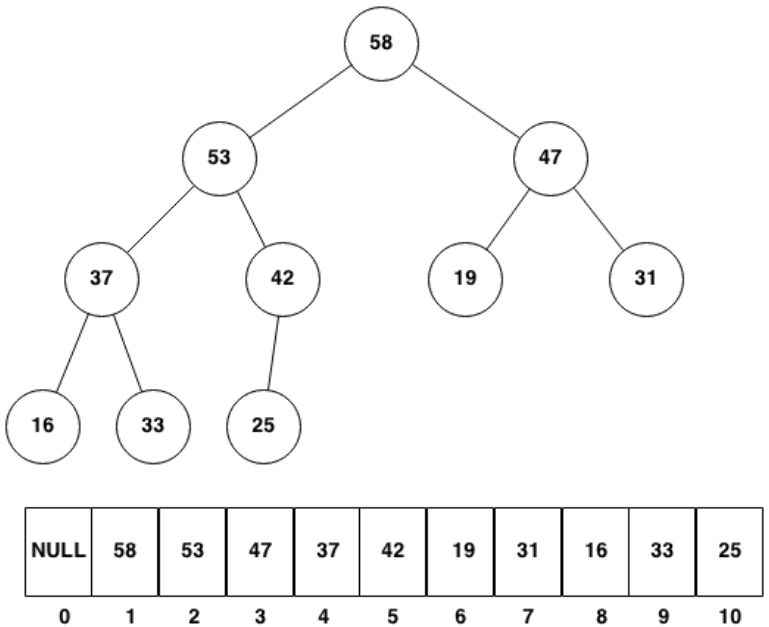
Το κλειδί που αντικατέστησε τη ρίζα “κατεβαίνει” προς τη σωστή θέση



Η δενδρική δομή του σωρού ως πίνακας

Ένας σωρός μπορεί να υλοποιηθεί με ένα πίνακα καταγράφοντας τα στοιχεία από πάνω προς τα κάτω και από αριστερά προς τα δεξιά.

- Τα κελιά γονείς βρίσκονται στις πρώτες $\lfloor \frac{n}{2} \rfloor$ ενώ τα φύλλα καταλαμβάνουν τις τελευταίες $\lceil \frac{n}{2} \rceil$ θέσεις.
- Τα παιδιά για κάθε κλειδί στις θέσεις i από 1 μέχρι και $\lfloor \frac{n}{2} \rfloor$ βρίσκονται στις θέσεις $2i$ και $2i + 1$.
- Ο γονέας για κάθε κλειδί στις θέσεις i από 2 μέχρι και n βρίσκεται στη θέση $\lfloor \frac{i}{2} \rfloor$.



Σχέση στοιχείων πίνακα

Ένας σωρός (Max-Heap) με n στοιχεία μπορεί να οριστεί ως ένας πίνακας $H[1 \cdots n]$ για τον οποίο ισχύει ότι κάθε στοιχείο στη θέση i στο πρώτο μισό του πίνακα είναι μεγαλύτερο ή ίσο από τα στοιχεία στις θέσεις $2i$ και $2i+1$.

$$H[i] \geq max\{H[2i], H[2i+1]\} \forall i = 1, \cdots, \lfloor \frac{n}{2} \rfloor \quad (1)$$

Δημιουργία σωρού από κάτω προς τα πάνω (bottom up)

Ξεκινώντας από τον τελευταίο στη σειρά κόμβο γονέα ο αλγόριθμος ελέγχει αν ικανοποιείται η κυριαρχία γονέα για τη τιμή κλειδιού K που διαθέτει. Αν δεν ισχύει τότε η τιμή κλειδιού K ανταλλάσσεται με την τιμή κλειδιού του μεγαλύτερου από τα κλειδιά των παιδιών και ελέγχεται αν η κυριαρχία γονέα ισχύει για το K στη νέα του θέση. Η διαδικασία συνεχίζεται μέχρι να ικανοποιηθεί για το K η κυριαρχία γονέα. Στη συνέχεια η ίδια διαδικασία ακολουθείται για τον επόμενο κόμβο γονέα και μέχρι να έρθει η σειρά της ρίζας.

```
const int static MAX_HEAP_SIZE = 1000;
int heap_size = 0;

template<class T> T* heap_bottom_up(T* a, int N) {
    T* heap = new T[MAX_HEAP_SIZE]();
    heap_size = N;
    heap[0] = NULL;
    for (int i = 0; i < N; i++)
        heap[i + 1] = a[i];
    for (int i = heap_size / 2; i >= 1; i--)
        heapify(heap, i);
    return heap;
}
```

```
template<class T> void heapify(T* heap, int k) {
    int v = heap[k];
    bool flag = false;
    while (!flag && 2 * k <= heap_size) {
        int j = 2 * k;
        if (j < heap_size)
            if (heap[j] < heap[j + 1])
                j++;
        if (v >= heap[j])
            flag = true;
        else {
            heap[k] = heap[j];
            k = j;
        }
    }
    heap[k] = v;
}
```

Εισαγωγή νέου κλειδιού στο σωρό

Εισαγωγή κόμβου με κλειδί *K* στο σωρό

Αρχικά ο κόμβος προσαρτάται στο τέλος του σωρού (δηλαδή μετά το τελευταίο φύλλο του δένδρου). Στη συνέχεια το κλειδί ανεβαίνει στη κατάλληλη θέση ως εξής: Το *K* συγκρίνεται με το κλειδί του γονέα του και αν το κλειδί του γονέα είναι μεγαλύτερο η διαδικασία τερματίζει αλλιώς τα 2 κλειδιά αντιμετατίθενται και το κλειδί *K* συγκρίνεται με το κλειδί του νέου γονέα του. Η διαδικασία συνεχίζεται μέχρι το *K* να είναι μικρότερο ή ίσο με τον γονέα του ή να έχει φτάσει στη ρίζα του δένδρου (δλδ στο στοιχείο 1 του πίνακα).

```
template<class T> void insert_key(T* h, T key) {
    heap_size++;
    h[heap_size] = key;
     int  pos = heap_size;
    while (pos != 1 && h[pos / 2] < h[pos]) {
        swap(h[pos / 2], h[pos]);
        pos = pos / 2;
    }
}
```

Διαγραφή μέγιστου κλειδιού από το σωρό

Διαγραφή του στοιχείου που βρίσκεται στη ρίζα του δένδρου

Το κλειδί που βρίσκεται στη ρίζα αντιμετατίθεται με το τελευταίο στοιχείο του σωρού. Το μέγεθος του σωρού μειώνεται κατά 1 και εκτελείται η διαδικασία heapify για το νέο πρώτο στοιχείο του σωρού.

```
template<class T> void maximum_key_deletion(T*
        heap) {
    swap(heap[1], heap[heap_size]);
    heap_size−−;
    heapify(heap, 1);
}
```

Heapsort

Ο αλγόριθμος Heapsort προτάθηκε από τον J.W.J.Williams το 1964 και αποτελείται από 2 στάδια:

1. Δημιουργία σωρού με τα n στοιχεία ενός πίνακα τα στοιχεία του οποίου ζητείται να ταξινομηθούν.
2. Εφαρμογή της διαγραφής της ρίζας $n-1$ φορές

Το αποτέλεσμα είναι ότι τα στοιχεία αφαιρούνται από το σωρό σε φθίνουσα σειρά. Καθώς κατά την αφαίρεσή του κάθε στοιχείο τοποθετείται στο τέλος του σωρού τελικά ο σωρός περιέχει τα αρχικά δεδομένα σε αύξουσα σειρά.

```
template<class T> T* heap_sort(T* a, int N) {
    T* heap = heap_bottom_up(a, N);
    for (int i = 1; i < N; i++)
        maximum_key_deletion(heap);
    return heap;
}
```

Απόδοση του αλγορίθμου Heapsort

Το στάδιο δημιουργίας του σωρού (heap_bottom_up) έχει πολυπλοκότητα $O(n)$ ενώ το στάδιο των διαδοχικών διαγραφών της ρίζας (maximum_key_deletion) έχει πολυπλοκότητα $O(n \log n)$. Συνεπώς η πολυπλοκότητά του Heapsort είναι $O(n) + O(n \log n) = O(n \log n)$.
Ο αλγόριθμος Heapsort είναι in place (σε θέση) δηλαδή μπορεί να υλοποιηθεί χωρίς να απαιτεί επιπλέον χώρο.
Στην πράξη έχει συγκρίσιμη ταχύτητα εκτέλεσης με το MergeSort αλλά είναι λιγότερο γρήγορος από το Quicksort.