Οι διαφάνειες που ακολουθούν είναι προσαρμογή των διαφανειών από τον ιστότοπο https://www.cise.ufl.edu/~sahni/cop3530/ (σελίδα του καθ. Sartaj Sahni για το μάθήμα των Δομών Δεδομένων)

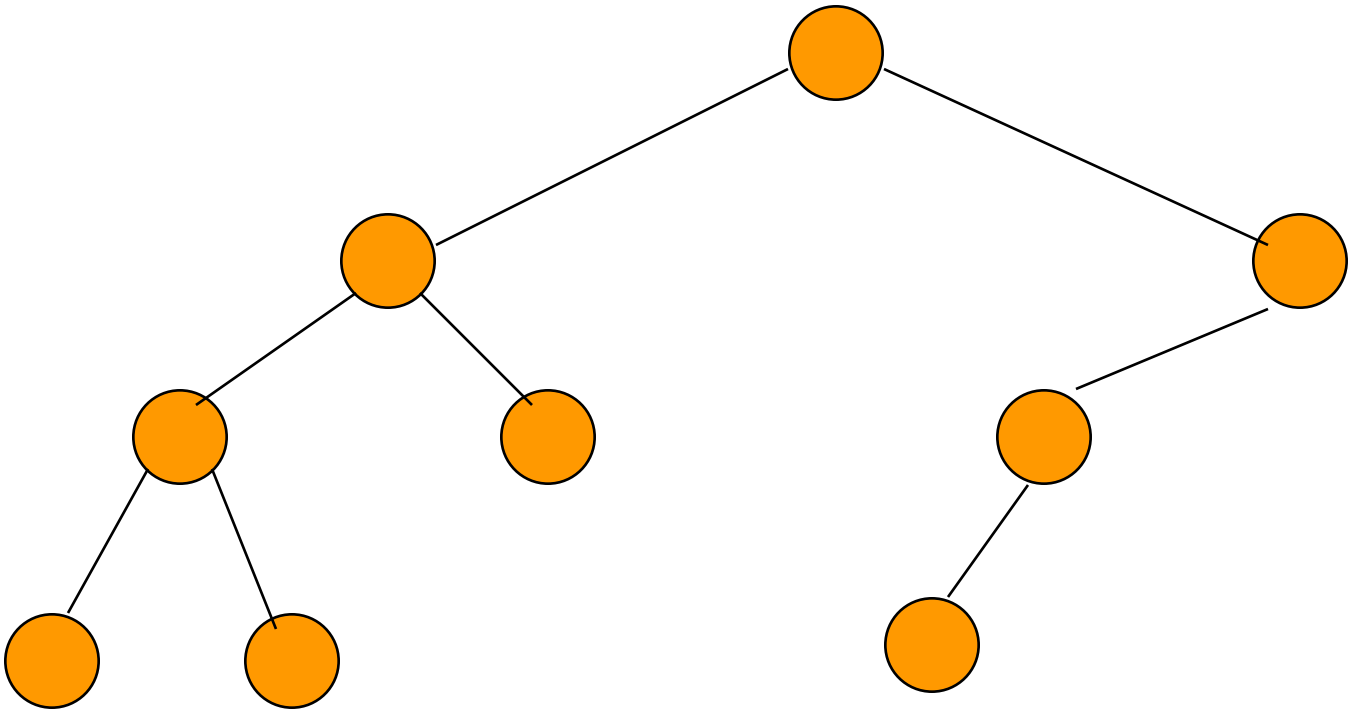
Αριστερικά Δέντρα

- Δυαδικά δέντρα που υλοποιούνται με τη χρήση δεικτών.
- Υποστηρίζουν τις ίδιες λειτουργίες που υποστηρίζει ένας σωρός με την ίδια ασυμπτωτική πολυπλοκότητα.
- Συγχώνευση δύο αριστερικών δέντρων που υλοποιούν ουρές προτεραιότητας σε χρόνο $O(\log n)$.

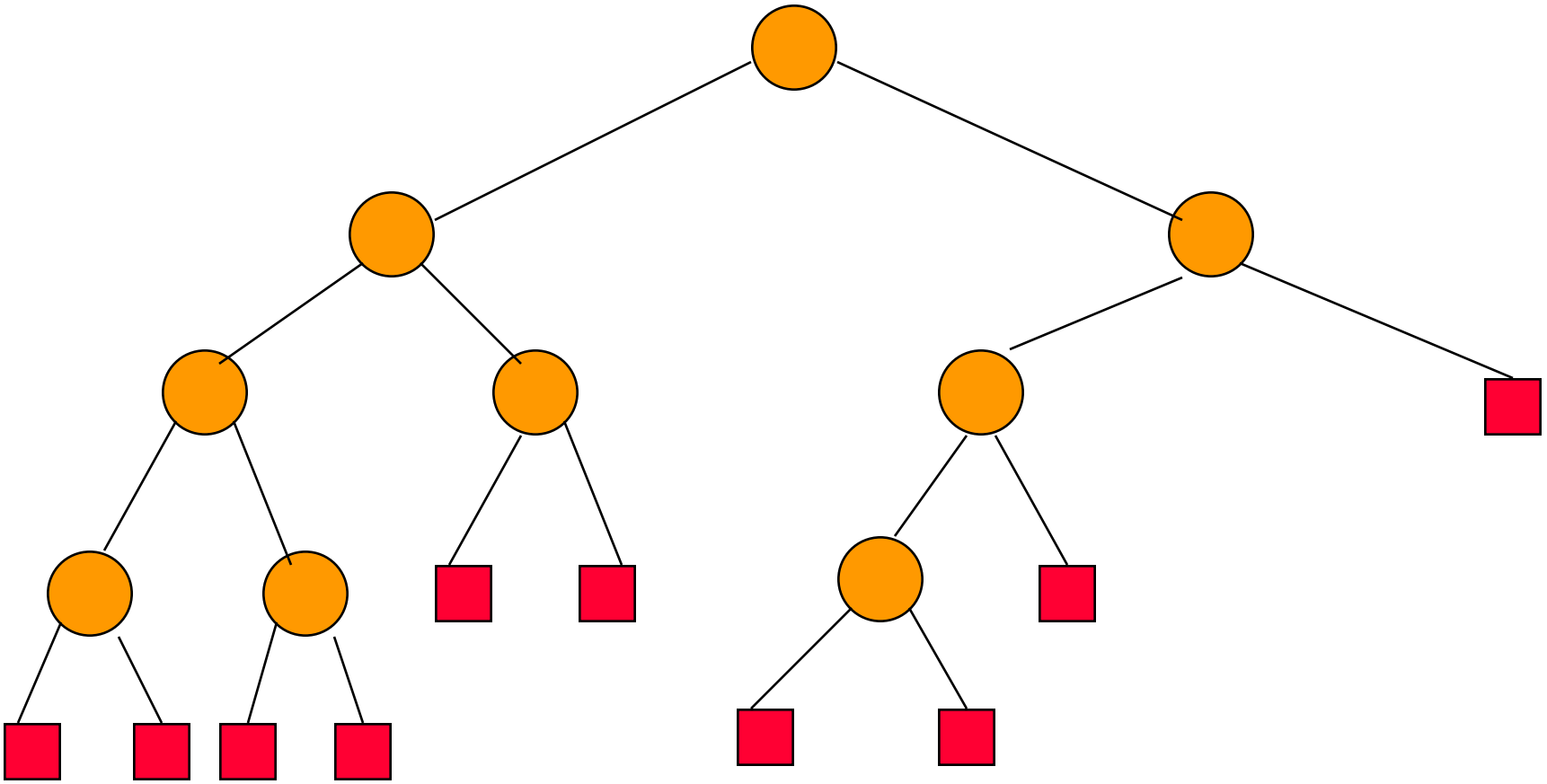
Εκτεταμένο Δυαδικό Δέντρο

- Έναρξη με οποιοδήποτε δυαδικό δέντρο και προσθήκη ενός εξωτερικού κόμβου οποτεδήποτε υπάρχει ένα άδειο υποδέντρο.
- Το αποτέλεσμα είναι ένα επεκταμένο δυαδικό δέντρο.

Ένα Δυαδικό Δέντρο



Ένα εκτεταμένο δυαδικό δέντρο

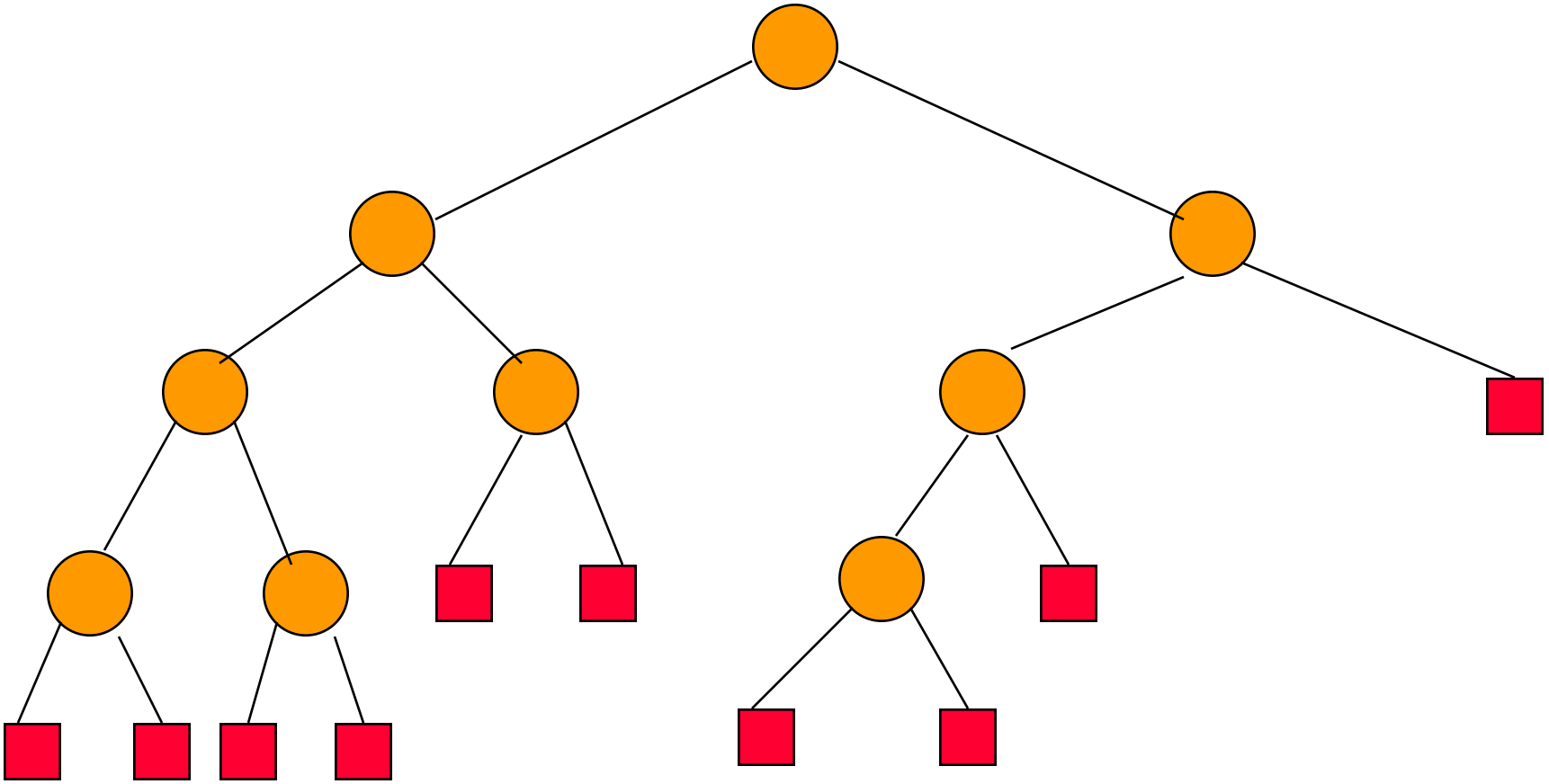


πλήθος εξωτερικών κόμβων: n+1

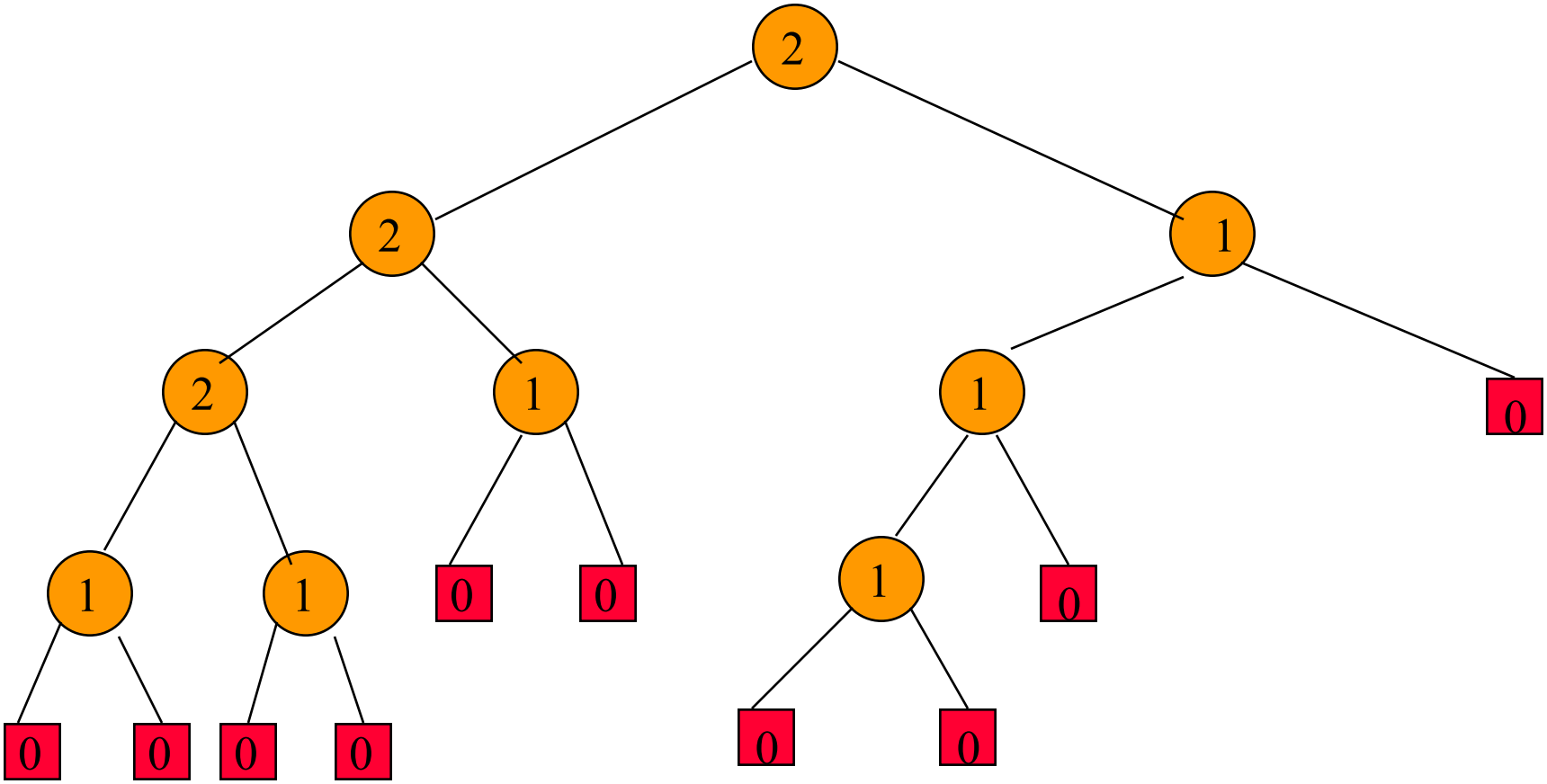
Η συνάρτηση s()

- Για κάθε κόμβο x σε ένα εκτεταμένο δέντρο, έστω s(x) το μήκος του συντομότερου μονοπατιού από τον x σε ένα εξωτερικό κόμβο σε ένα υποδέντρο με ρίζα τον x.

s() Values Example



Παράδειγμα Τιμών s()



Ιδιότητες της s()

Αν x είναι ένας εξωτερικός κόμβος, τότε
$s(x) = 0$.

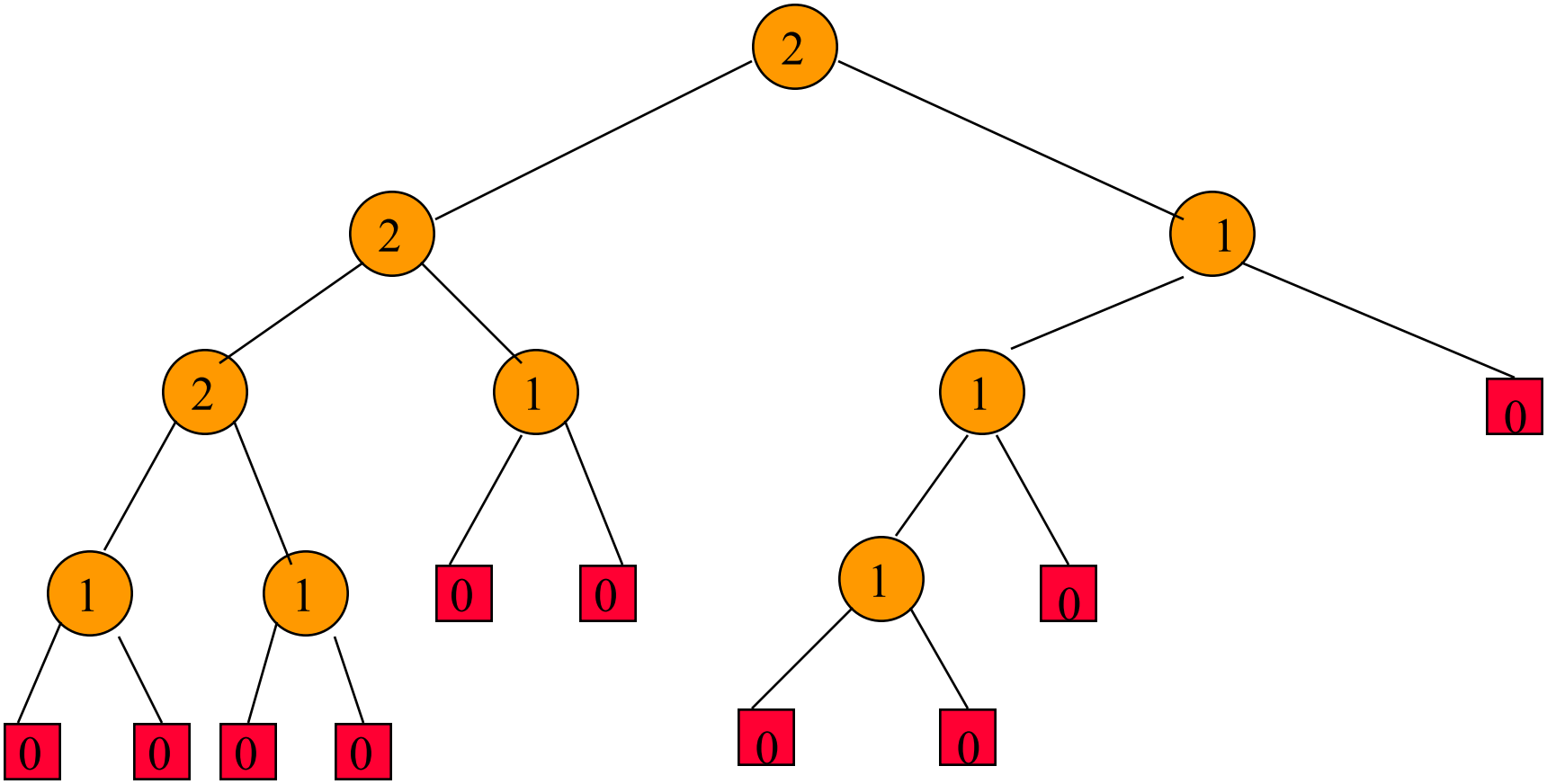
Αλλιώς,

$$s(x) = \min \{s(\text{leftChild}(x)), s(\text{rightChild}(x))\} + 1$$

Αριστερικό Δέντρο επηρεασμένο ως προς το ύψος (Height Biased Leftist Trees) - HBLT

Ένα δυαδικό δέντρο είναι ένα αριστερικό δέντρο ανν για κάθε εσωτερικό κόμβο x, s(leftChild(x)) >= s(rightChild(x))

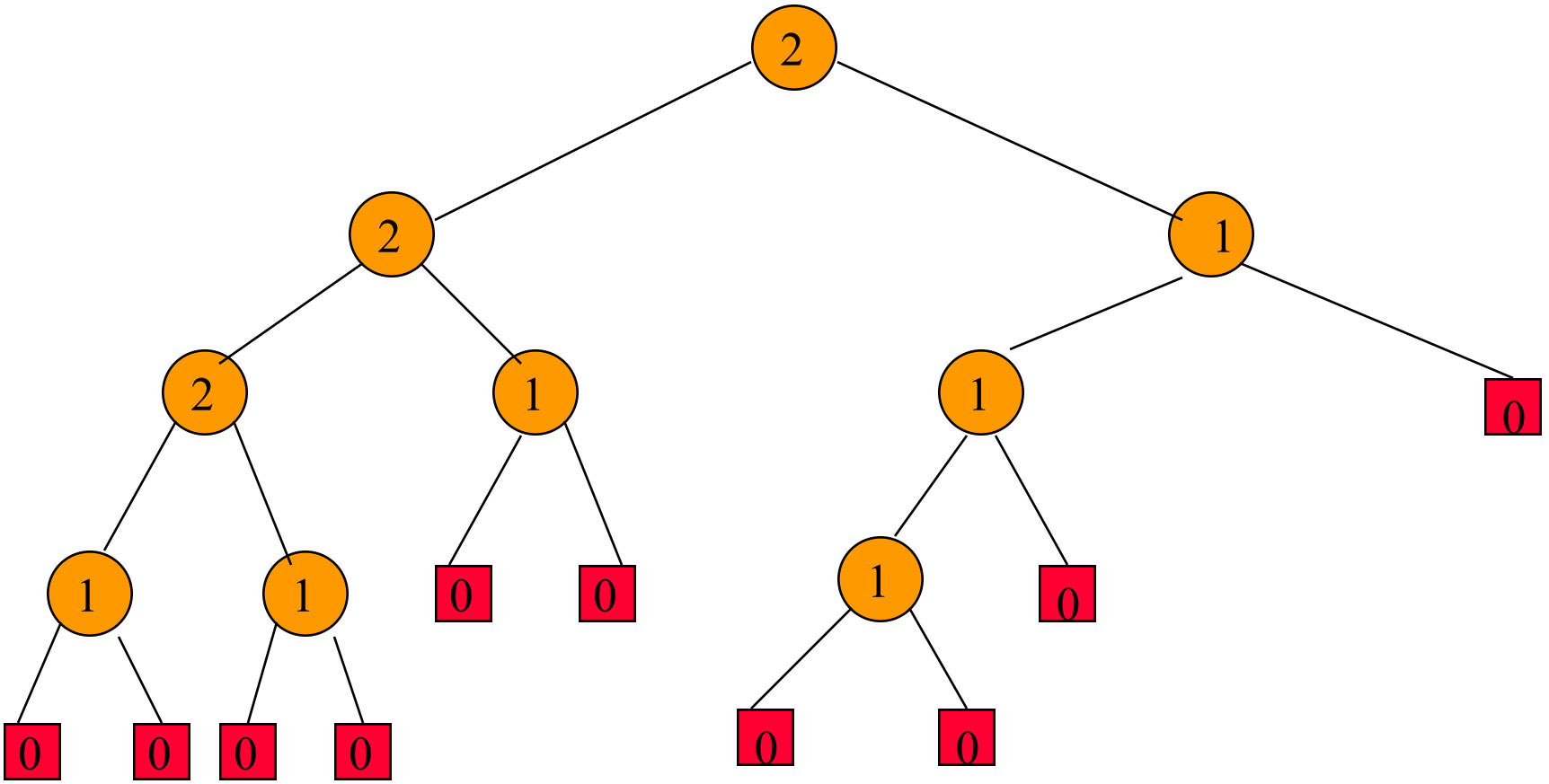
Ένα αριστερικό δέντρο



Αριστερικά Δέντρα--Ιδιότητα 1

- Σε ένα αριστερικό δέντρο, το πιο δεξιό μονοπάτι είναι το συντομότερο από τη ρίζα σε εξωτερικό κόμβο μονοπάτι και το μήκος του μονοπατιού είναι ίσο με s(root).

Ένα Αριστερικό Δέντρο



Μήκος του πιο δεξιού μονοπατιού είναι 2.

Αριστερικά Δέντρα--Ιδιότητα 2

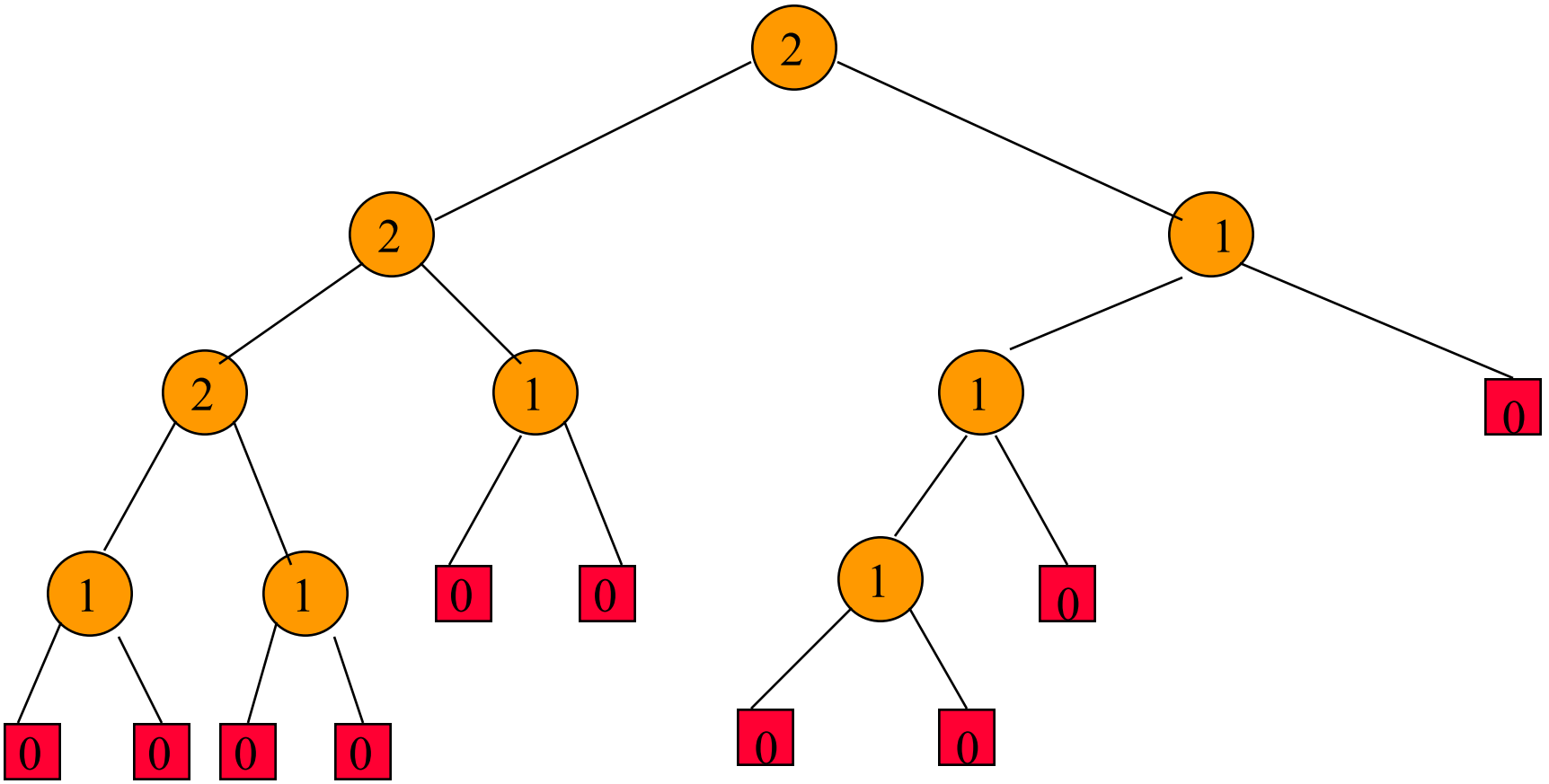
Το πλήθος των εσωτερικών κόμβων είναι τουλάχιστον

$2^{s(root)} - 1$

Επειδή τα επίπεδα 1 μέχρι s(root) δεν έχουν εξωτερικούς κόμβους.

Έτσι, $s(root) <= \log(n+1)$

Ένα Αριστερικό Δέντρο



Τα επίπεδα 1 και 2 δεν έχουν εξωτερικούς κόμβους.

Αριστερικά Δέντρα--Ιδιότητα 3

- Μήκος του πιο δεξιού μονοπατιού είναι O(log n), όπου n είναι το πλήθος των κόμβων σε ένα αριστερικό δέντρο..

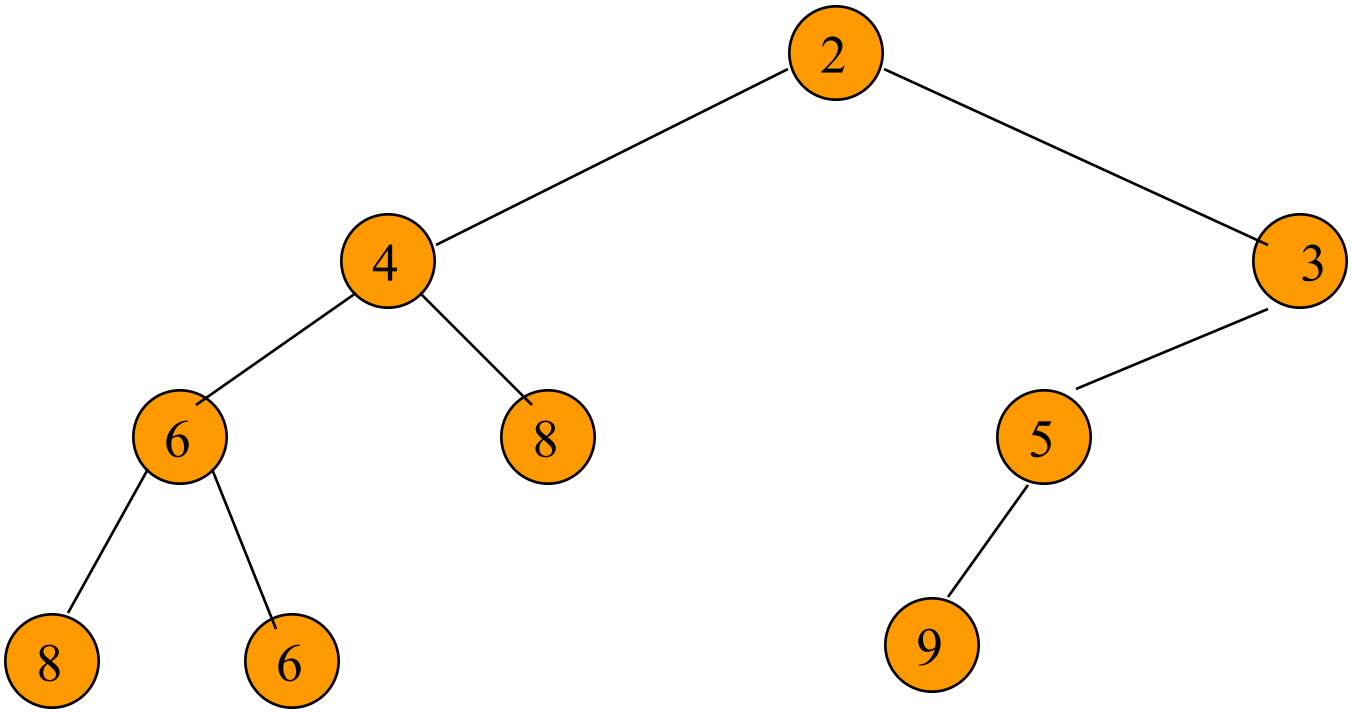
- Έπεται από τις ιδιότητες 1 και 2.

Αριστερικά Δέντρα ως Ουρές Προτεραιότητας

Αριστερικό δέντρο ελαχίστων: Ένα αριστερικό δέντρο το οποίο είναι και δέντρο ελαχίστων.

Αριστερικό δέντρο μεγίστων: Ένα αριστερικό δέντρο το οποίο είναι και δέντρο μεγίστων.

Αριστερικό δέντρο ελαχίστων



Λειτουργίες Αριστερικών Δέντρων Ελαχίστων

Insert()

Delete_Min()

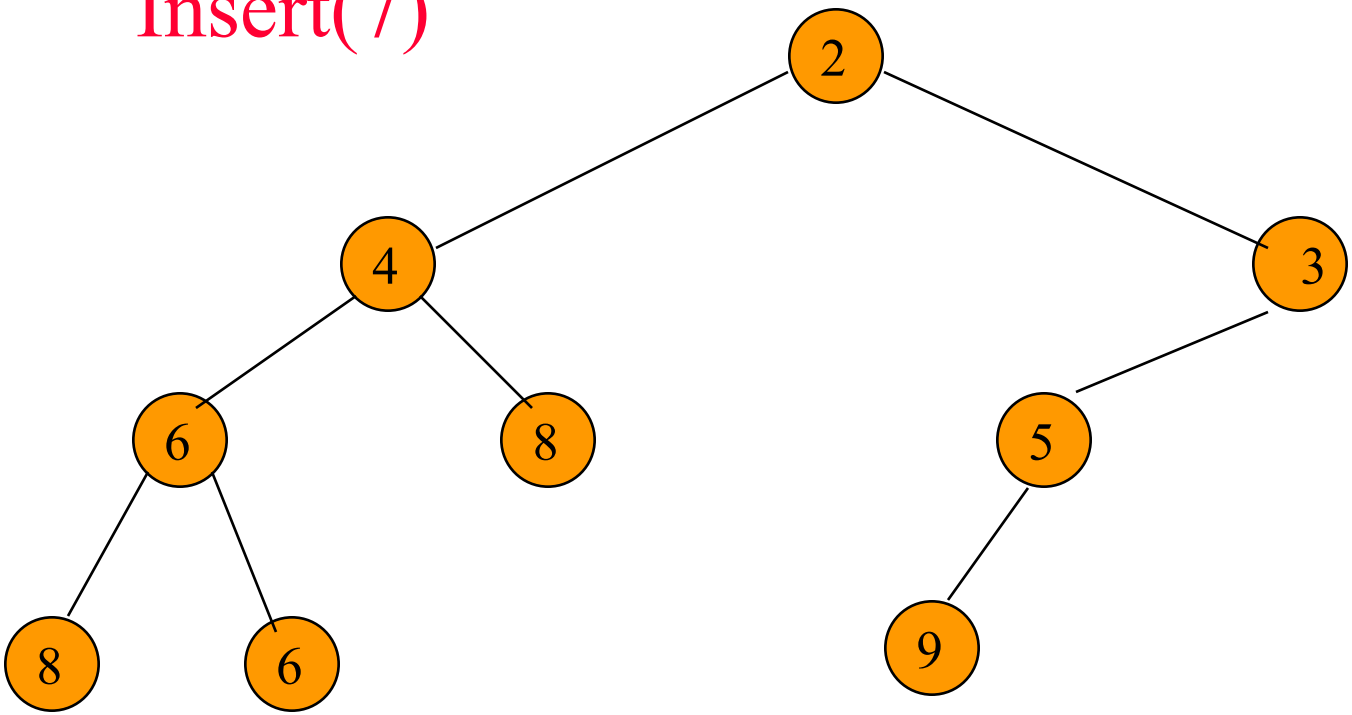
Meld()

Initialize()

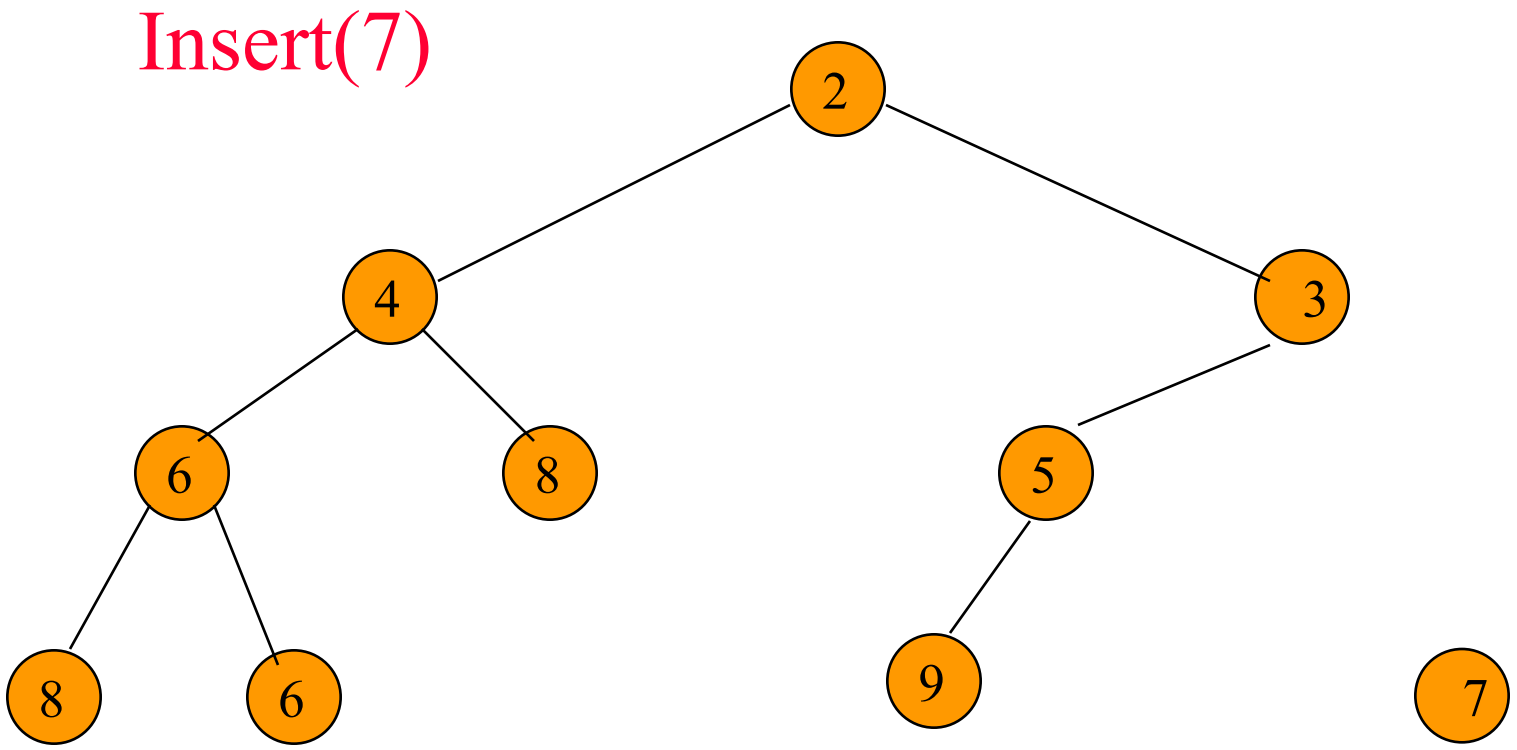
Η Insert() και Delete() χρησιμοποιούν τη Meld().

Λειτουργία Insert

Insert(7)

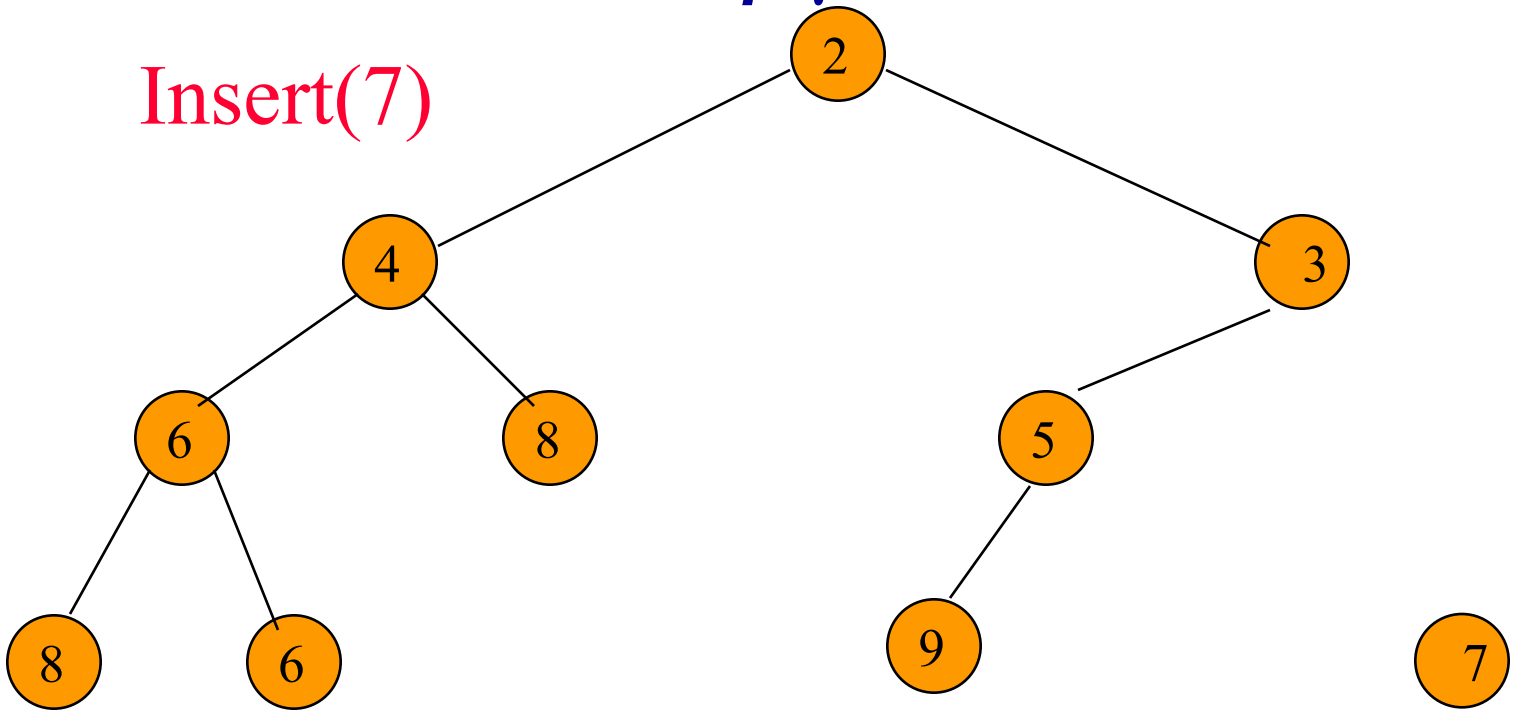


Λειτουργία Insert



Δημιουργούμε ένα δέντρο που περιέχει ως μοναδικό κόμβο το στοιχείο 7.

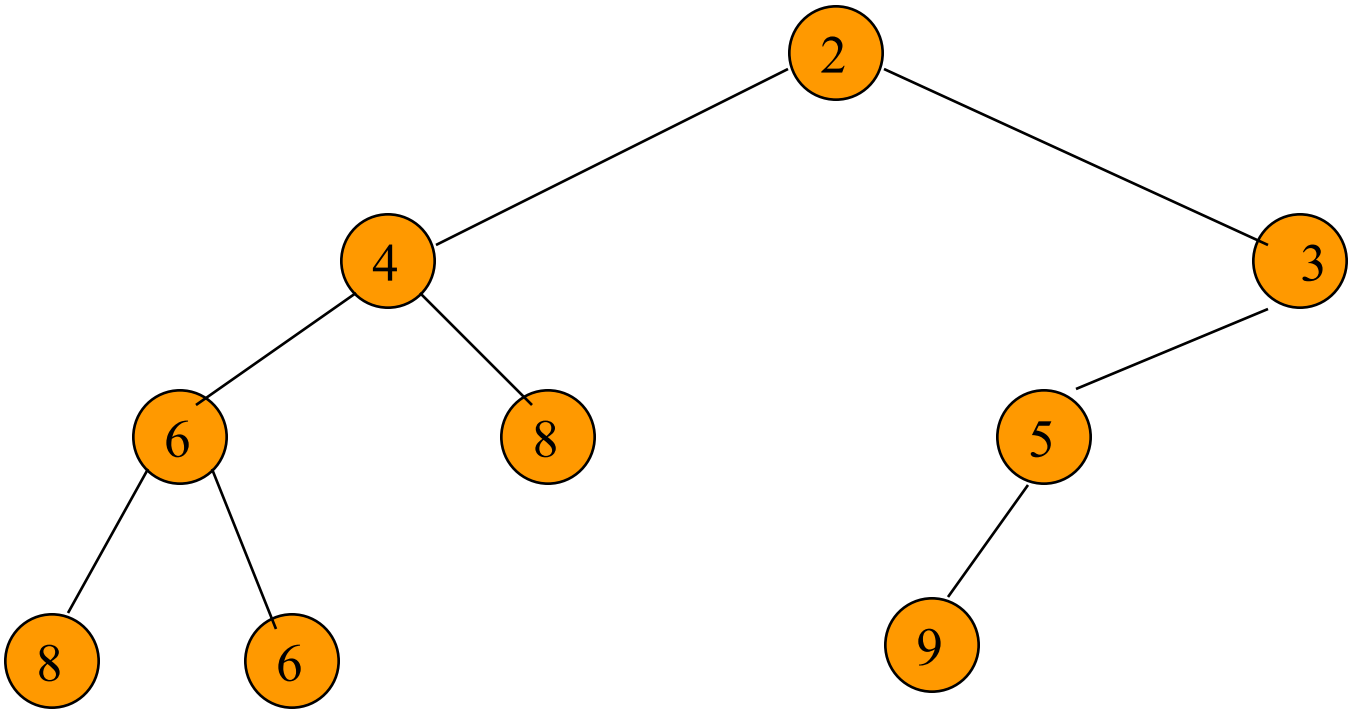
Λειτουργία Insert



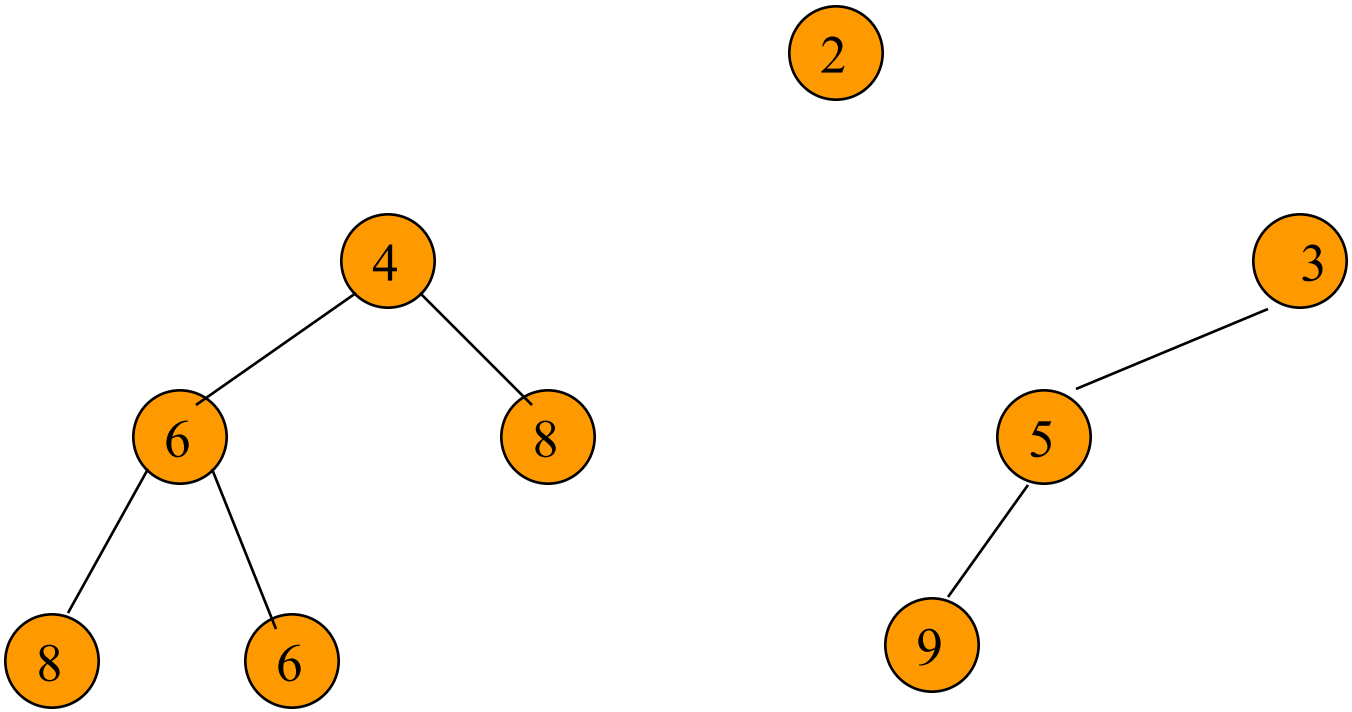
Δημιουργούμε ένα δέντρο που περιέχει ως μοναδικό κόμβο το στοιχείο 7.

Συγχωνεύουμε τα δύο δέντρα.

Αφαίρεση Ελαχίστου

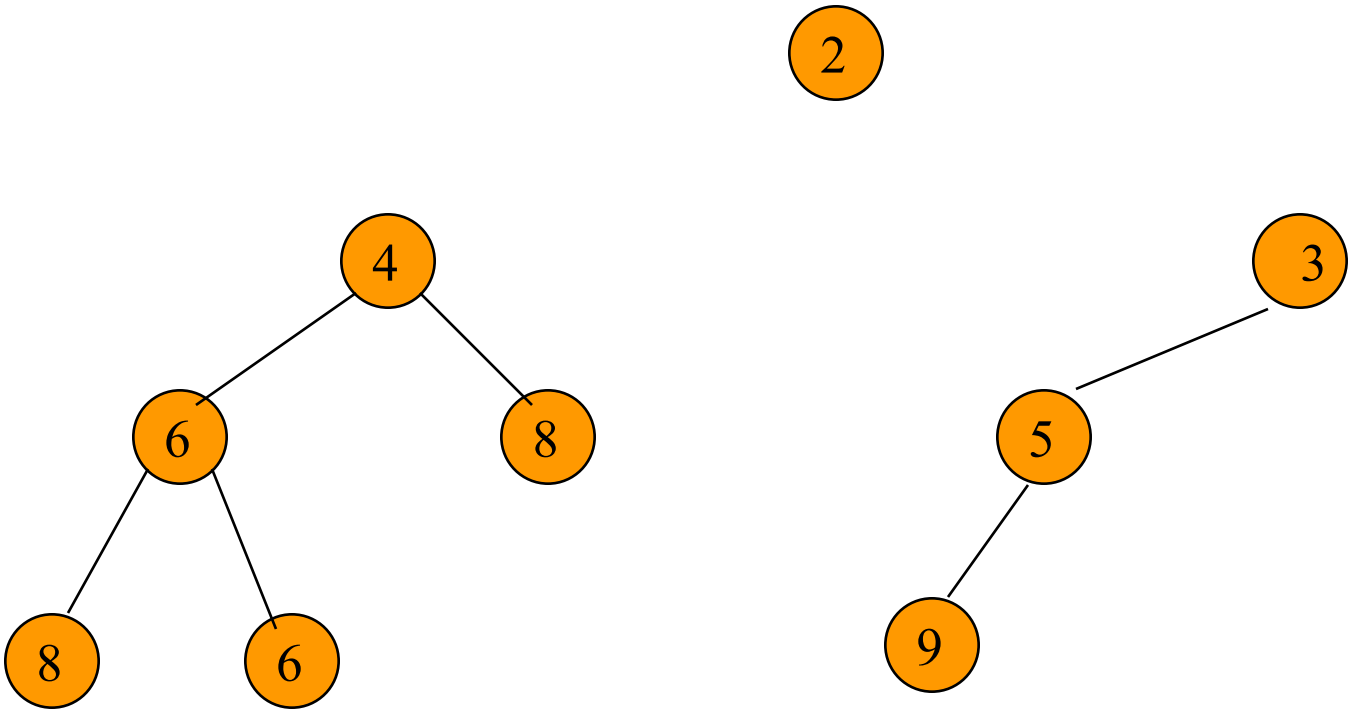


Διαγραφή Ελαχίστου



Αφαίρεση της ρίζας.

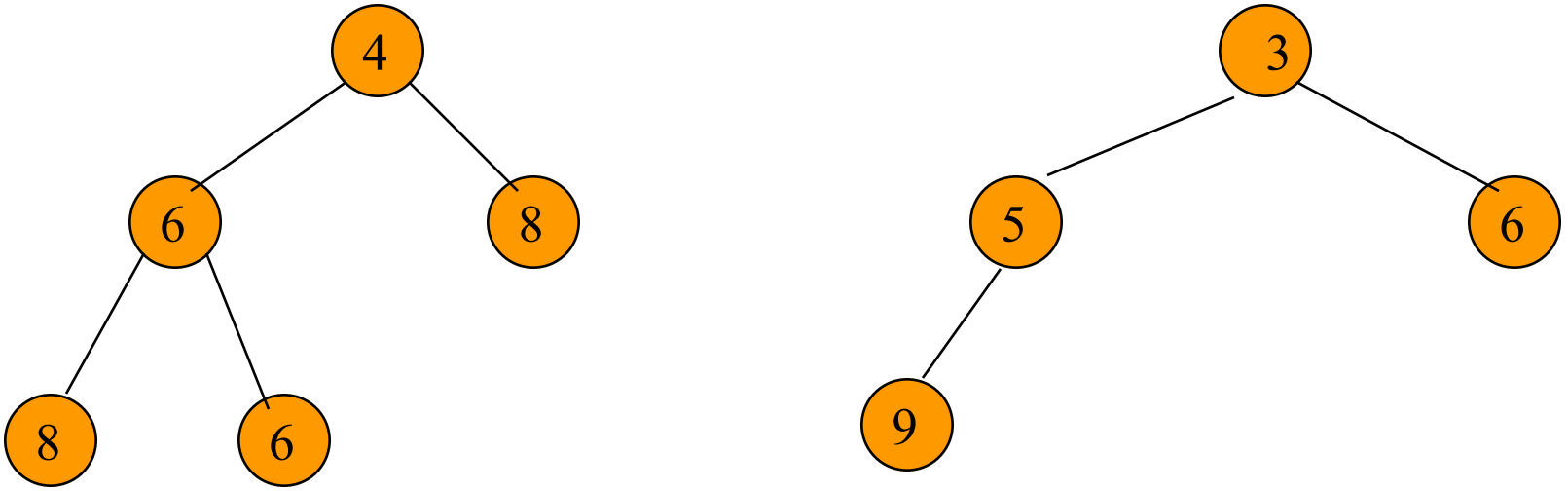
Διαγραφή του ελαχίστου



Αφαίρεση της ρίζας.

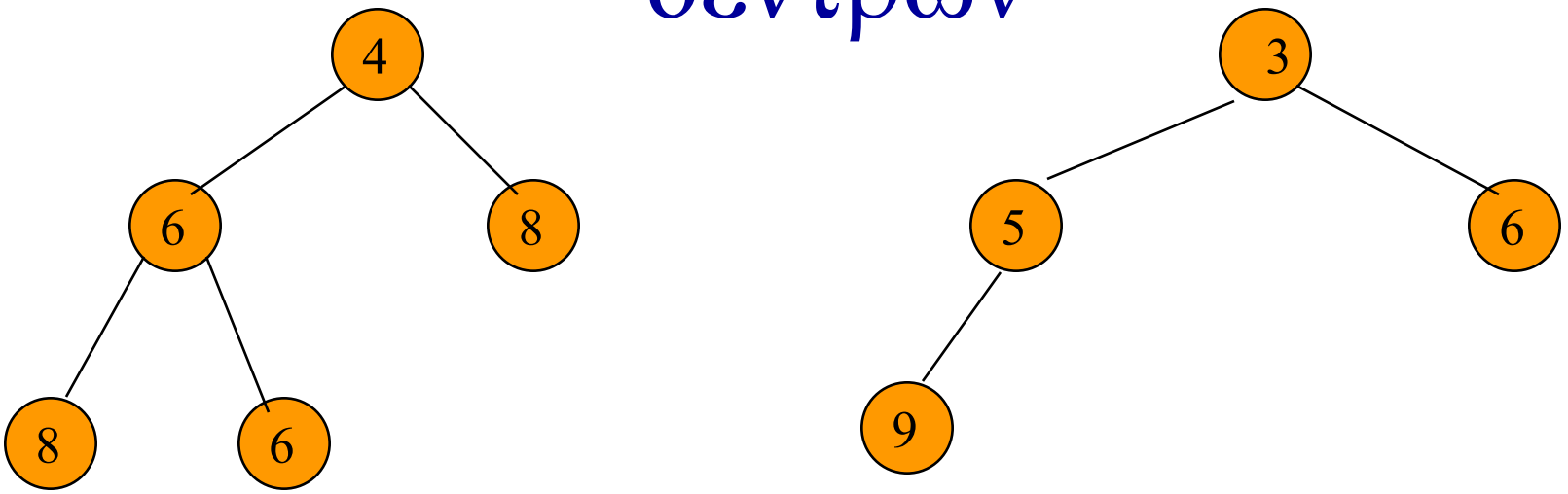
Συγχώνευση υποδέντρων.

Συγχώνευση δύο αριστερικών δέντρων



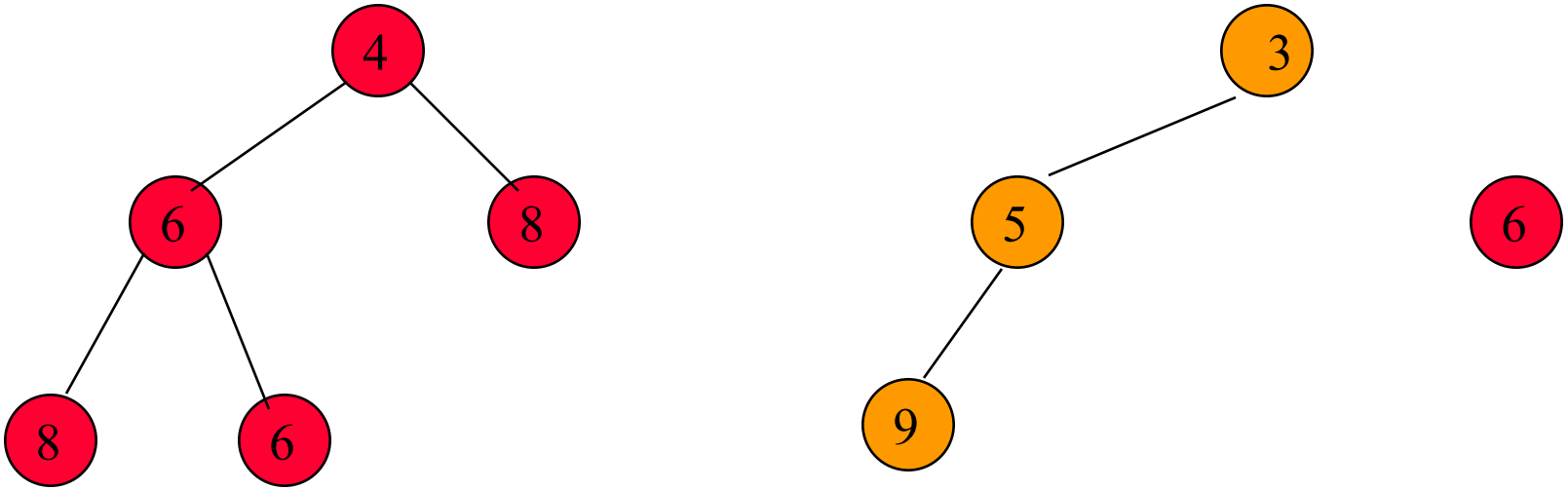
Διάσχιση μόνο των πιο δεξιών μονοπατιών προκειμένου να πετύχουμε λογαριθμική πολυπλοκότητα.

Συγχώνευση δύο αριστερικών δέντρων



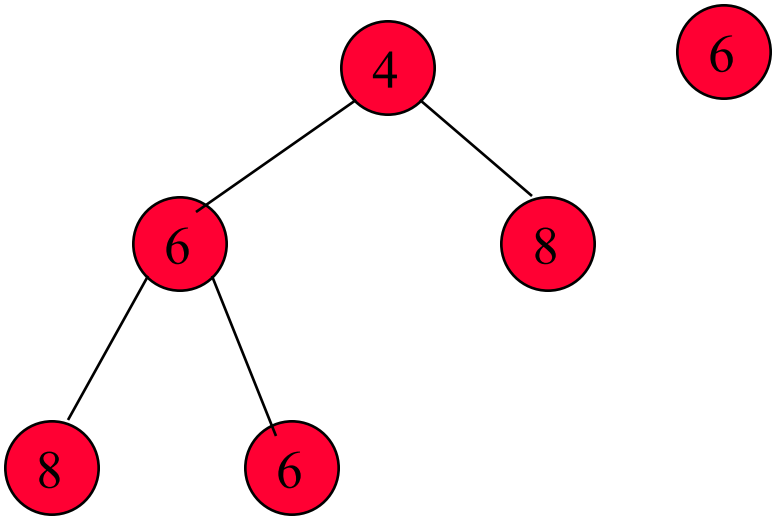
Συγχώνευση του δεξιού υποδέντρου του δέντρου με τη μικρότερη ρίζα με ολόκληρο το άλλο δέντρο.

Συγχώνευση δύο αριστερικών δέντρων



Συγχώνευση του δεξιού υποδέντρου του δέντρου με τη μικρότερη ρίζα με ολόκληρο το άλλο δέντρο.

Συγχώνευση δύο αριστερικών δέντρων



Συγχώνευση του δεξιού υποδέντρου του δέντρου με τη μικρότερη ρίζα με ολόκληρο το άλλο δέντρο

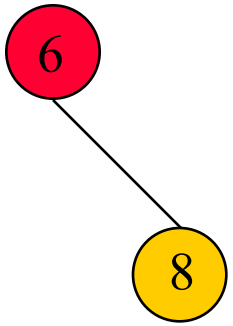
Συγχώνευση δύο αριστερικών δέντρων

Συγχώνευση του δεξιού υποδέντρου του δέντρου με τη μικρότερη ρίζα με ολόκληρο το άλλο δέντρο

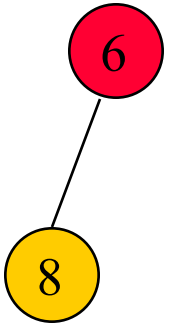
Το δεξιό υποδέντρο του 6 είναι άδειο. Έτσι, το αποτέλεσμα της συγχώνευσης του δεξιού υποδέντρου του δέντρου με τη μικρότερη ρίζα με το άλλο δέντρο είναι το άλλο δέντρο..

Συγχώνευση δύο αριστερικών δέντρων

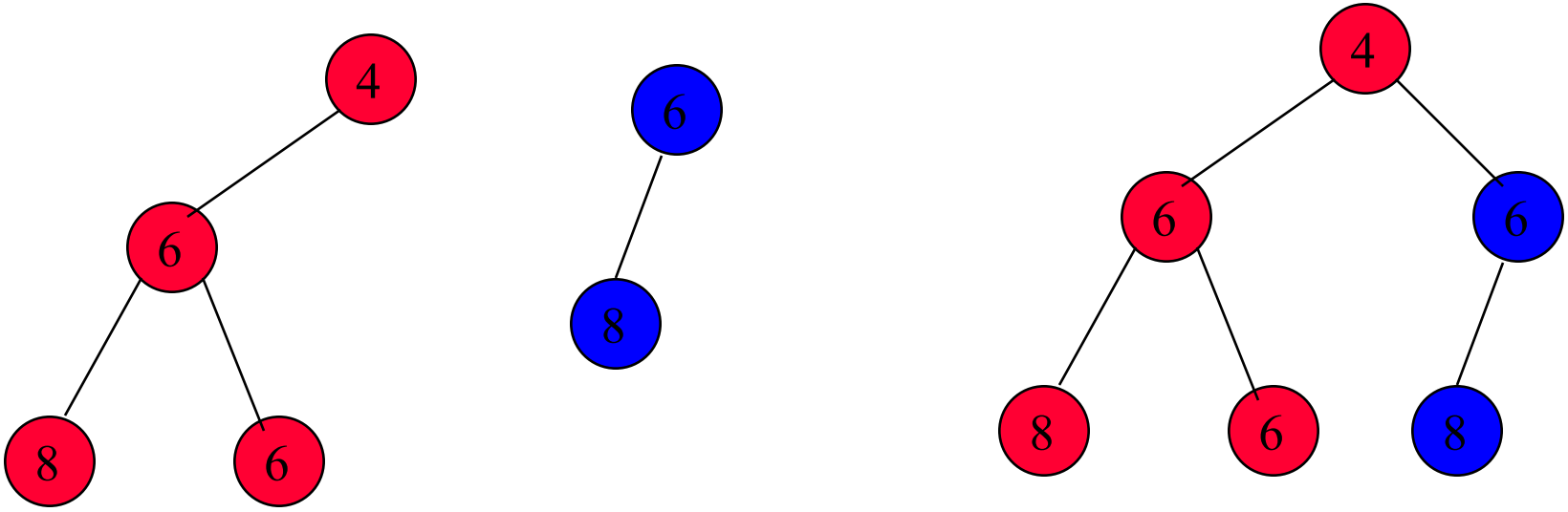
Συγχώνευση του δεξιού υποδέντρου του δέντρου με τη μικρότερη ρίζα με ολόκληρο το άλλο δέντρο



Ενάλλαξε το αριστερό και δεξιό υποδέντρο αν s(left) < s(right).



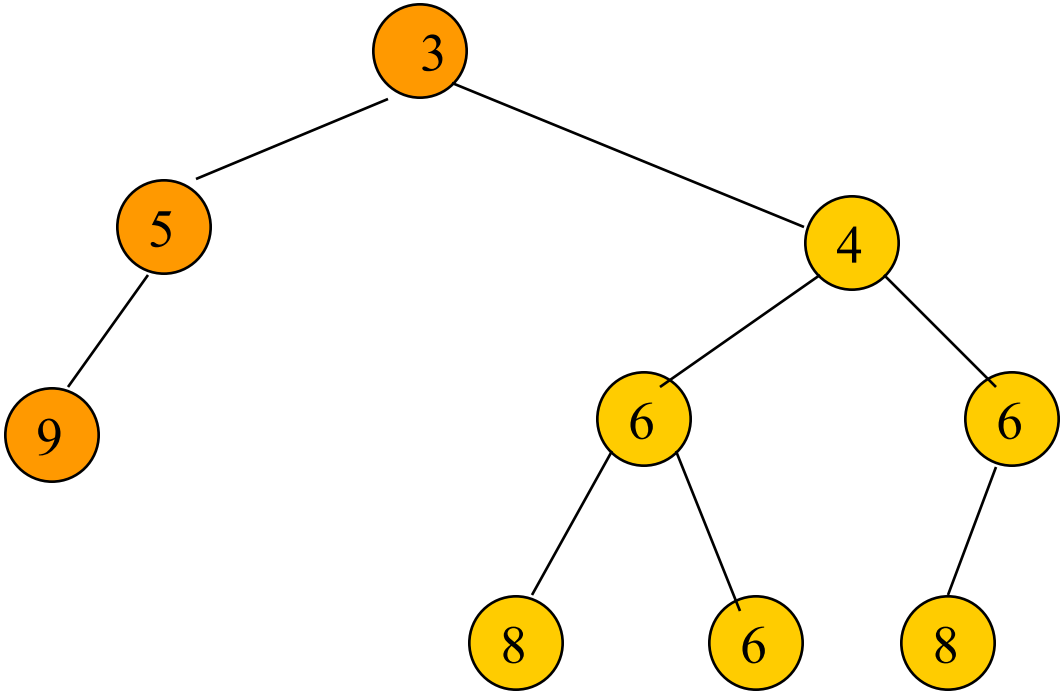
Συγχώνευση δύο αριστερικών δέντρων



Συγχώνευση του δεξιού υποδέντρου του δέντρου με τη μικρότερη ρίζα με ολόκληρο το άλλο δέντρο

Ενάλλαξε το αριστερό και δεξιό υποδέντρο αν s(left) < s(right).

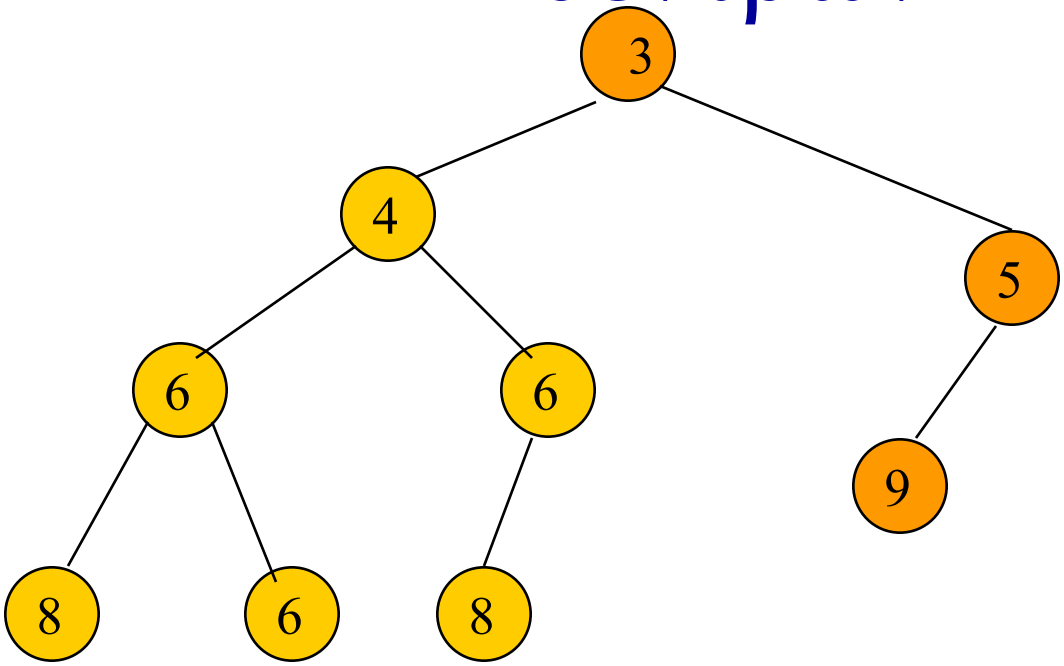
Συγχώνευση δύο αριστερικών δέντρων

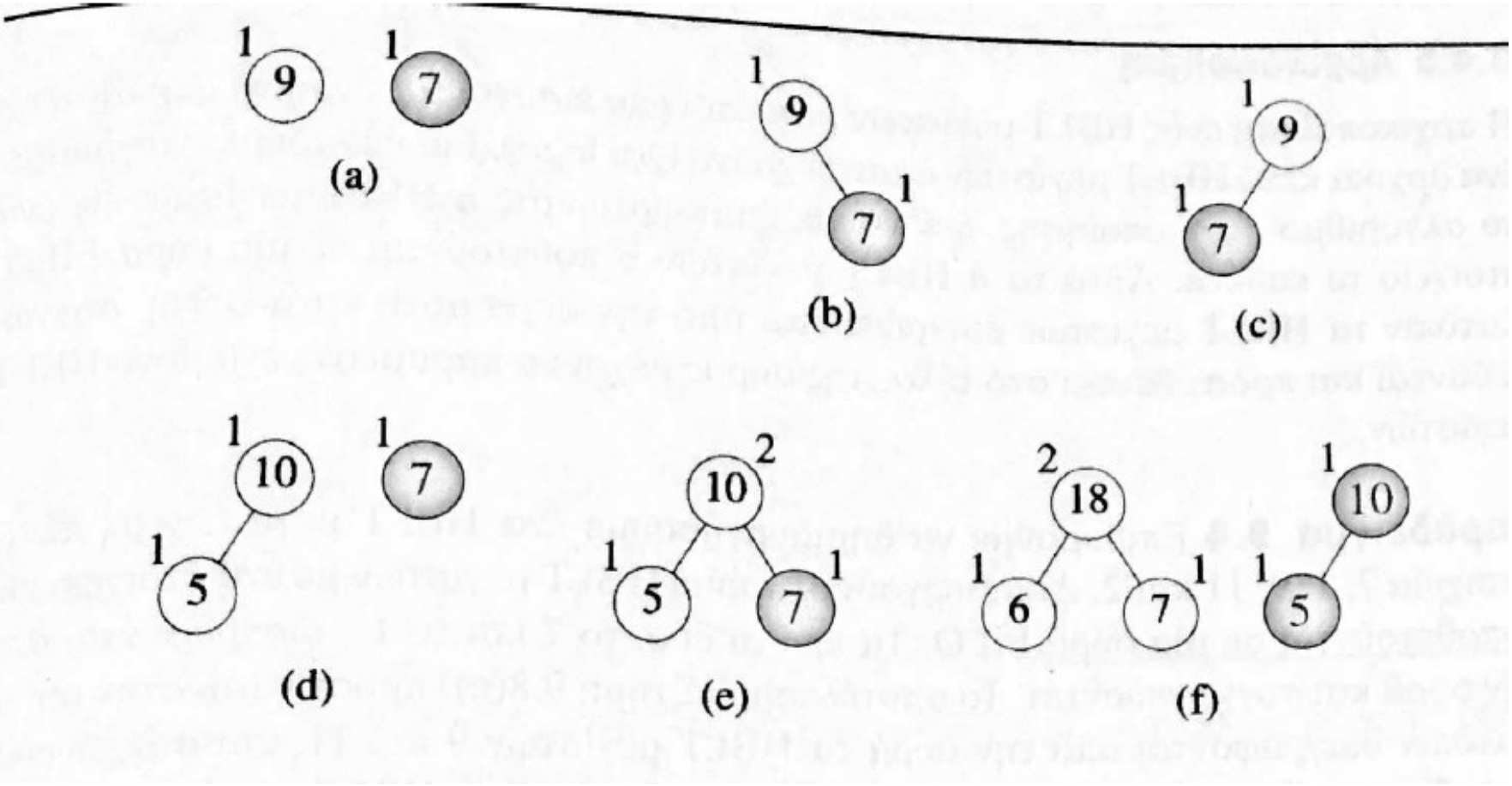


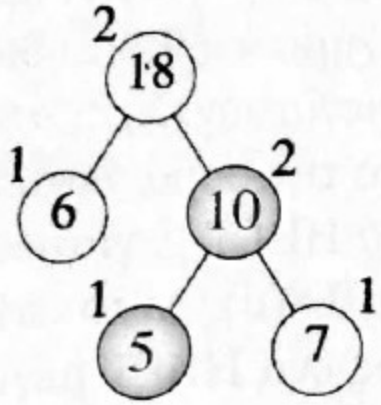
Συγχώνευση του δεξιού υποδέντρου του δέντρου με τη μικρότερη ρίζα με ολόκληρο το άλλο δέντρο

Ενάλλαξε το αριστερό και δεξιό υποδέντρο αν s(left) < s(right).

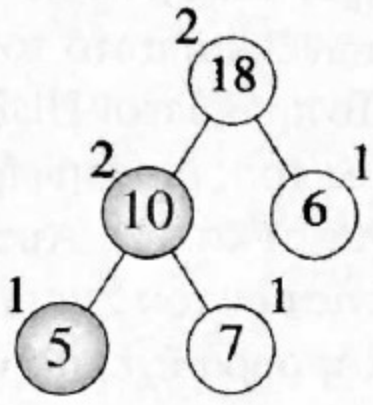
Συγχώνευση δύο αριστερικών δέντρων



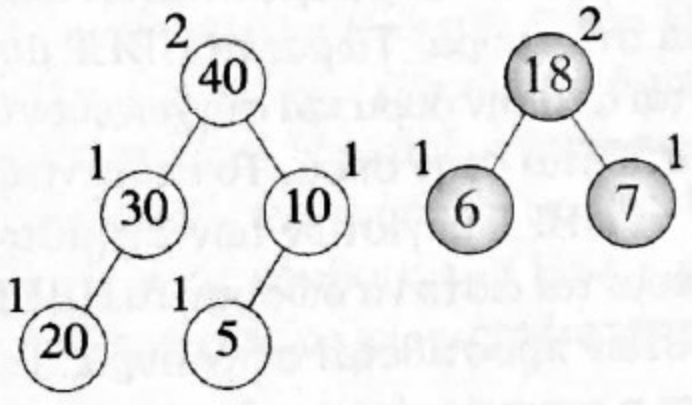

1
9
1
7
(a)
1
9
1
7
(b)
1
9
1
7
(c)
1
10
1
7
1
5
(d)
2
10
1
5
7
1
(e)
2
18
1
6
7
1
1
10
1
5
(f)


2
18
1
6
2
10
1
5
1
7

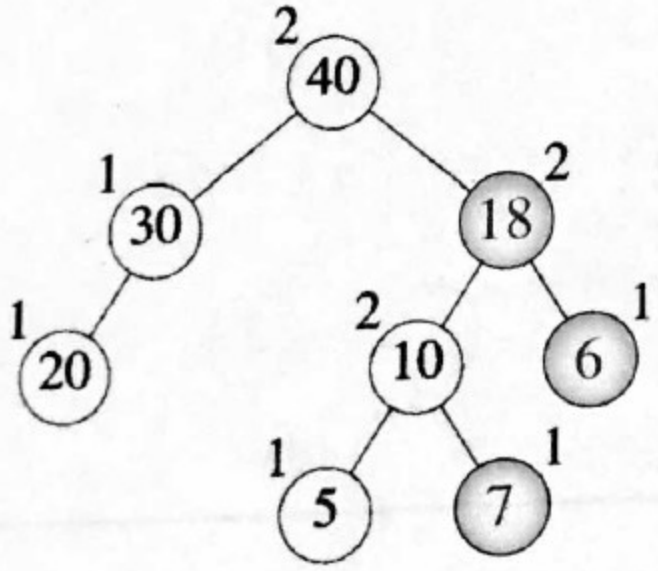
(g)


2
18
2
10
1
6
1
5
1
7

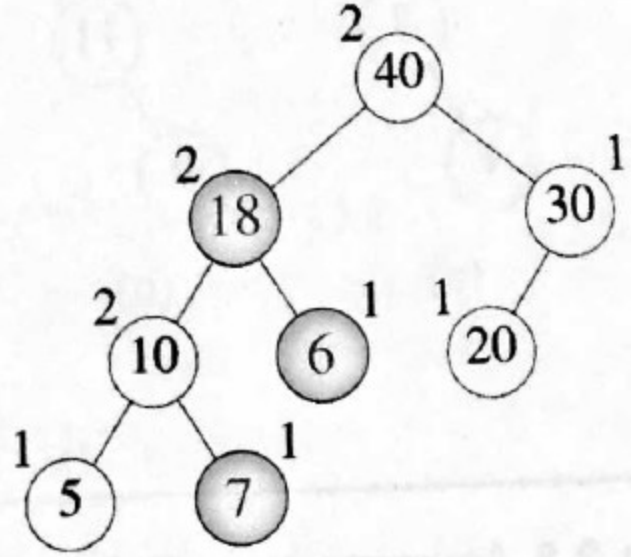
(h)


2
40
1
30
1
10
1
20
1
5
2
18
1
6
1
7

(i)


2
40
1
30
2
18
1
20
2
10
1
6
1
5
1
7

(j)

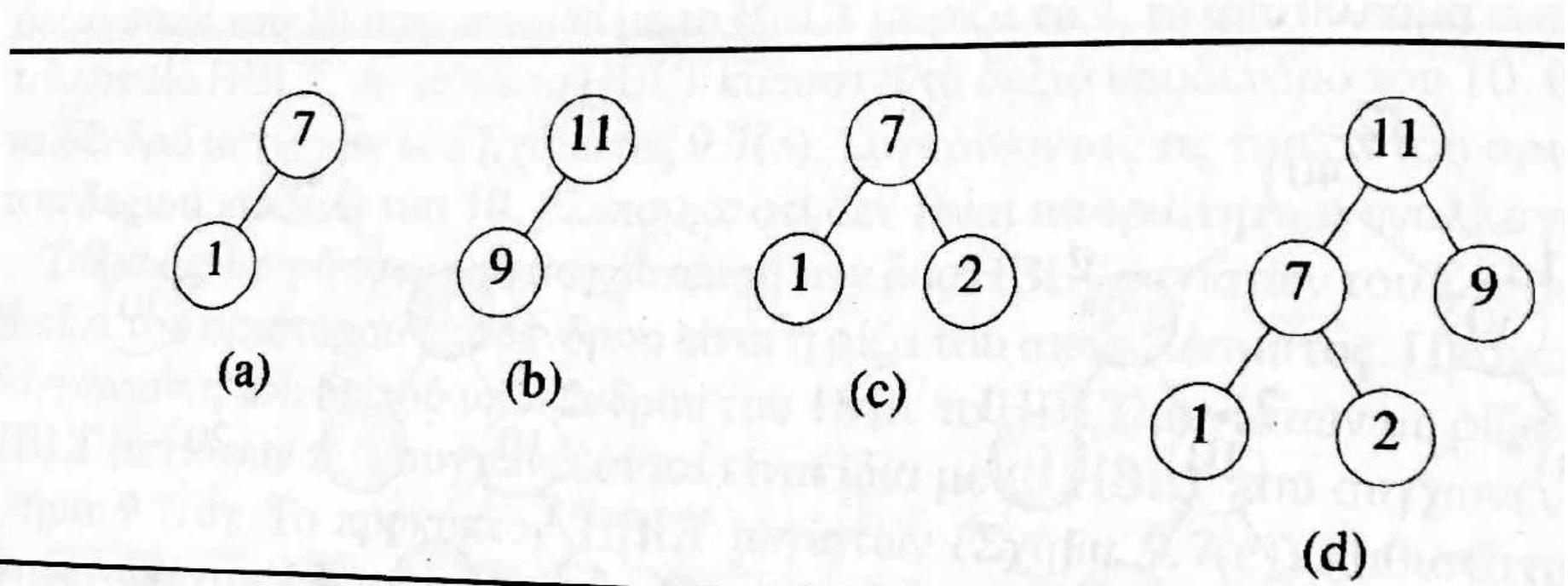

2
40
2
18
1
30
2
10
1
6
1
20
1
5
1
7

(k)

Αρχικοποίηση σε χρόνο O(n)

- Δημιούργησε n δέντρα ενός κόμβου και τοποθέτησε τα σε μία ουρά FIFO
- Επαναληπτικά αφαίρεσε δύο αριστερικά δέντρα από την ουρά, συγχώνευσε αυτά τα δέντρα, και τοποθέτησε το νέο δέντρο πίσω στην ουρά
- Η διαδικασία τερματίζει όταν μόνο ένα αριστερικό δέντρο μένει στην ουρά

Παράδειγμα αρχικοποίησης: 7,1,9,11,2



Σχήμα 9.8 Αρχικοποίηση HBLT μεγίστων

```
template <class T>
class HBLTNode {
   friend MaxHBLT<T>;
   public:
      HBLTNode(const T& e, const int sh)
       {data = e;
        s = sh;
        LeftChild = RightChild = 0;}
   private:
      int s; // s value of node
      T data;
      HBLTNode<T> *LeftChild, *RightChild;
};
```

Πρόγραμμα 9.6 Κλάση κόμβου αριστερίστικου δένδρου επηρεασμένου ως προς το ύψος

```
template<class T>
class MaxHBLT {
   public:
      MaxHBLT() {root = 0;}
      ~MaxHBLT() {Free(root);}
      T Max() {if (!root) throw OutOfBounds();
               return root->data;}
      MaxHBLT<T>& Insert(const T& x);
      MaxHBLT<T>& DeleteMax(T& x);
      MaxHBLT<T>& Meld(MaxHBLT<T>& x) {
                  Meld(root,x.root);
                  x.root = 0;
                  return *this;}
      void Initialize(T a[], int n);
   private:
      void Free(HBLTNode<T> *t);
      void Meld(HBLTNode<T>* &x, HBLTNode<T>* y);
      HBLTNode<T> *root;  // pointer to tree root
};
```

Πρόγραμμα 9.7 Κλάση MaxHBLT

```
template<class T>
void MaxHBLT<T>::Meld(HBLTNode<T>* &x, HBLTNode<T>* y)
{// Meld leftist trees with roots *x and *y.
 // Set x to point to new root.
   if (!y) return; // y is empty
   if (!x) // x is empty
      {x = y;
       return;}

   // neither is empty
   if (x->data < y->data) Swap(x,y);
   // now x->data >= y->data
  Meld(x->RightChild,y);
  if (!x->LeftChild) {// left subtree empty
        // swap subtrees
        x->LeftChild = x->RightChild;
        x->RightChild = 0;
        x->s = 1;}
  else {// see if subtrees to be swapped
        if (x->LeftChild->s < x->RightChild->s)
           Swap(x->LeftChild,x->RightChild);
        x->s = x->RightChild->s + 1;}
}
```

Πρόγραμμα 9.8 Συγχώνευση δύο αριστερίστικων δένδρων

```
template<class T>
MaxHBLT<T>& MaxHBLT<T>::Insert(const T& x)
{// Insert x into the leftist tree.
 // Create tree with one node.
   HBLTNode<T> *q = new HBLTNode<T> (x,1);
   // meld q and original tree
   Meld(root,q);
   return *this;
}
```

Πρόγραμμα 9.9 Εισαγωγή σε HBLT μεγίστων

```
template<class T>
MaxHBLT<T>& MaxHBLT<T>::DeleteMax(T& x)
{// Delete max element and put it in x.
   if (!root) throw OutOfBounds();

   // tree not empty
   x = root->data;   // max element
   HBLTNode<T> *L = root->LeftChild;
   HBLTNode<T> *R = root->RightChild;
   delete root;
   root = L;
   Meld(root,R);
   return *this;
}
```

Πρόγραμμα 9.10 Διαγραφή μέγιστου στοιχείου από HBLT μεγίστων

```
template<class T>
void MaxHBLT<T>::Initialize(T a[], int n)
{// Initialize hblt with n elements.
   Queue<HBLTNode<T> *> Q(n);
   Free(root);  // delete old nodes
   // initialize queue of trees
   for (int i = 1; i <= n; i++) {
      // create trees with one node each
      HBLTNode<T> *q = new HBLTNode<T> (a[i],1);
      Q.Add(q);
      }

   // repeatedly meld from queue
   HBLTNode<T> *b, *c;
   for (int i = 1; i <= n - 1; i++) {
      // delete and meld two trees
      Q.Delete(b).Delete(c);
      Meld(b,c);
      // put melded tree on queue
      Q.Add(b);
      }

   if (n) Q.Delete(root);
}
```

Πρόγραμμα 9.11 Αρχικοποίηση HBLT μεγίστων