Data Structures

October 12 A

Topics

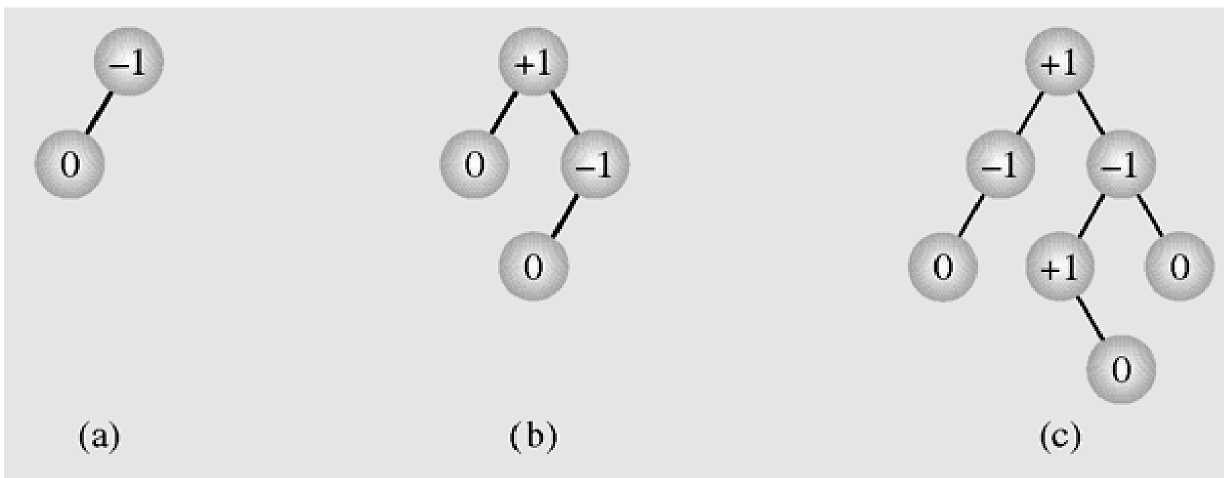
- Global balancing of binary (search) trees
- Local balancing of binary (search) trees (finish up today)
- Self-organizing (binary search) trees
- Heaps
- Postfix expressions

Recall Definitions

- *Height-balanced* (aka *balanced*) binary tree: for any node in the tree the height of the node's subtrees differ by at most 1.
- Page 251 Drozdek: a *perfectly balanced* tree, is a balanced tree such that all leaves are on one level or on two levels. (Exercise: construct a balanced tree such that the leaves are in more than two levels.)
- A binary tree has the *Symmetric Search property*, if
  - Each node contains a key
  - Different nodes will have different keys
  - There is an ordering on the keys
  - For each node in the tree, the key of the node is strictly bigger than *any* key in the left subtree and strictly smaller than *any* key in the right subtree

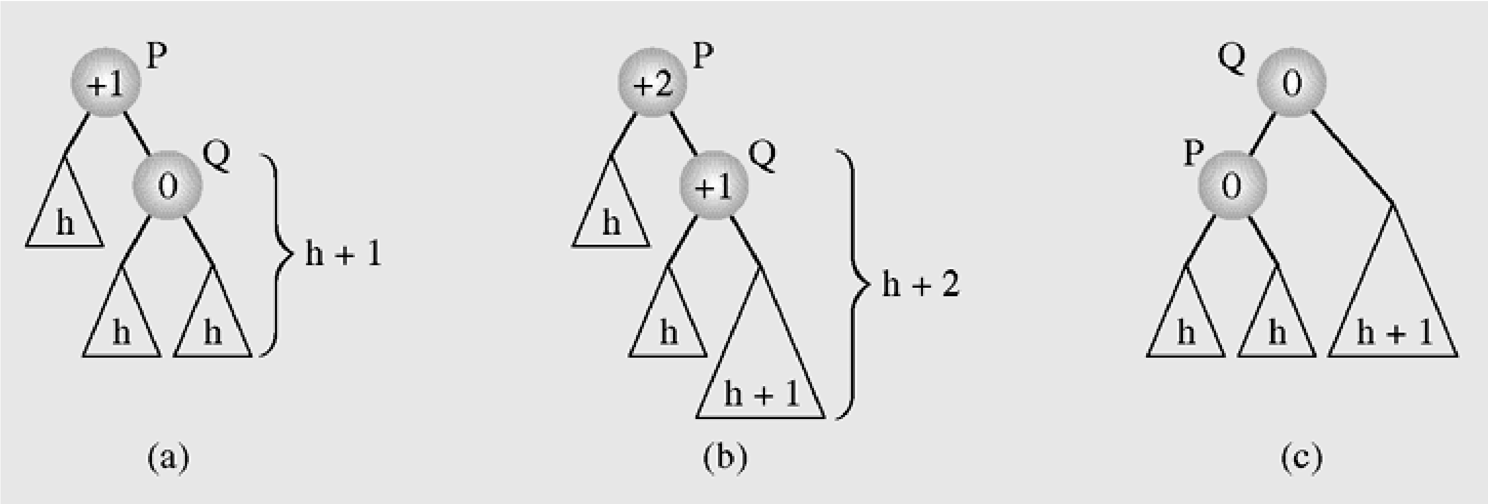
AVL Trees

- An **AVL tree** is one in which the height of the left and right subtrees of every node differ by at most one



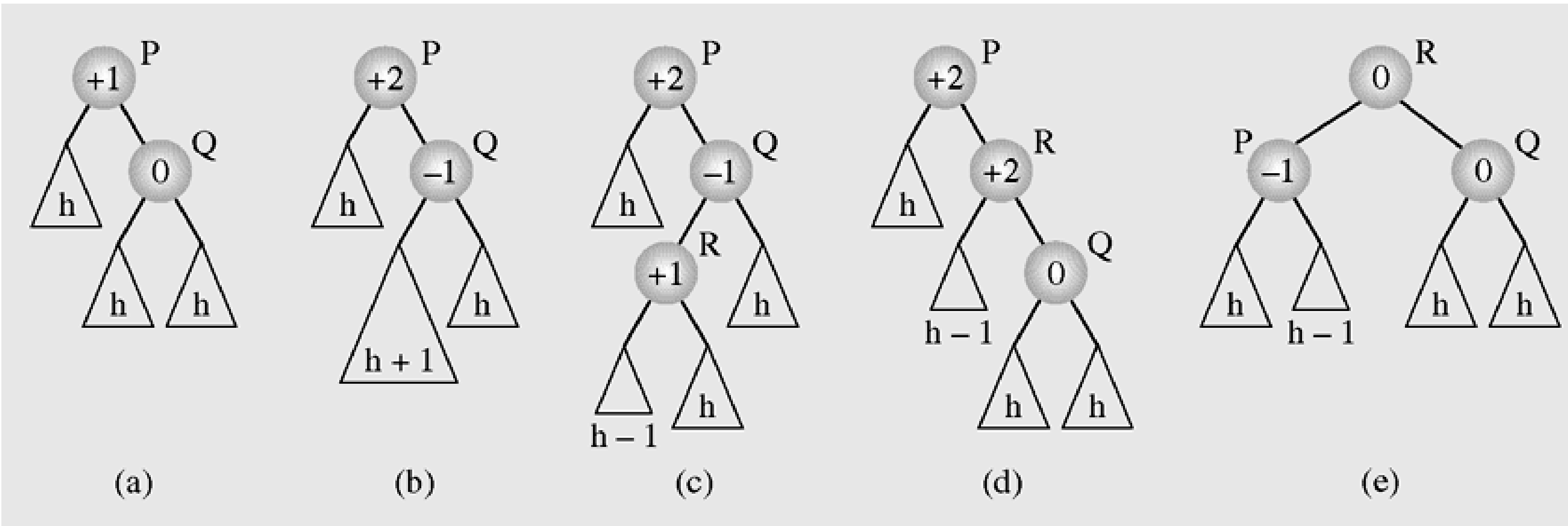
Examples of AVL trees

AVL Trees (continued)



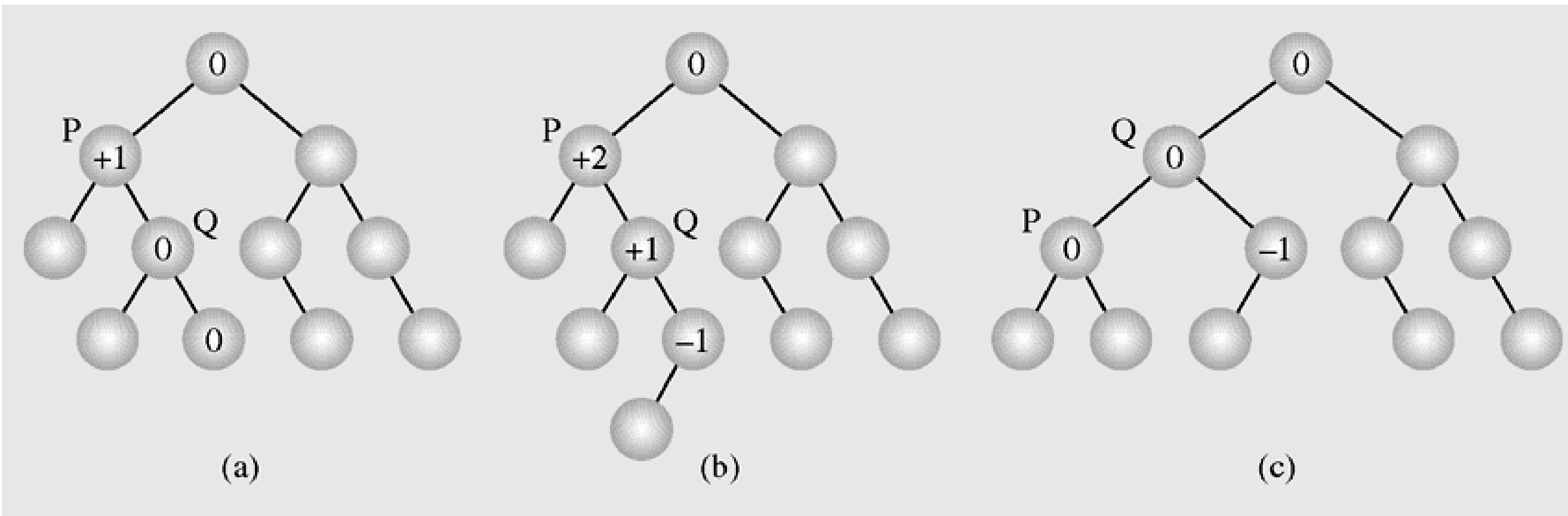
Balancing a tree after insertion of a node in the right subtree of node *Q*

AVL Trees (continued)



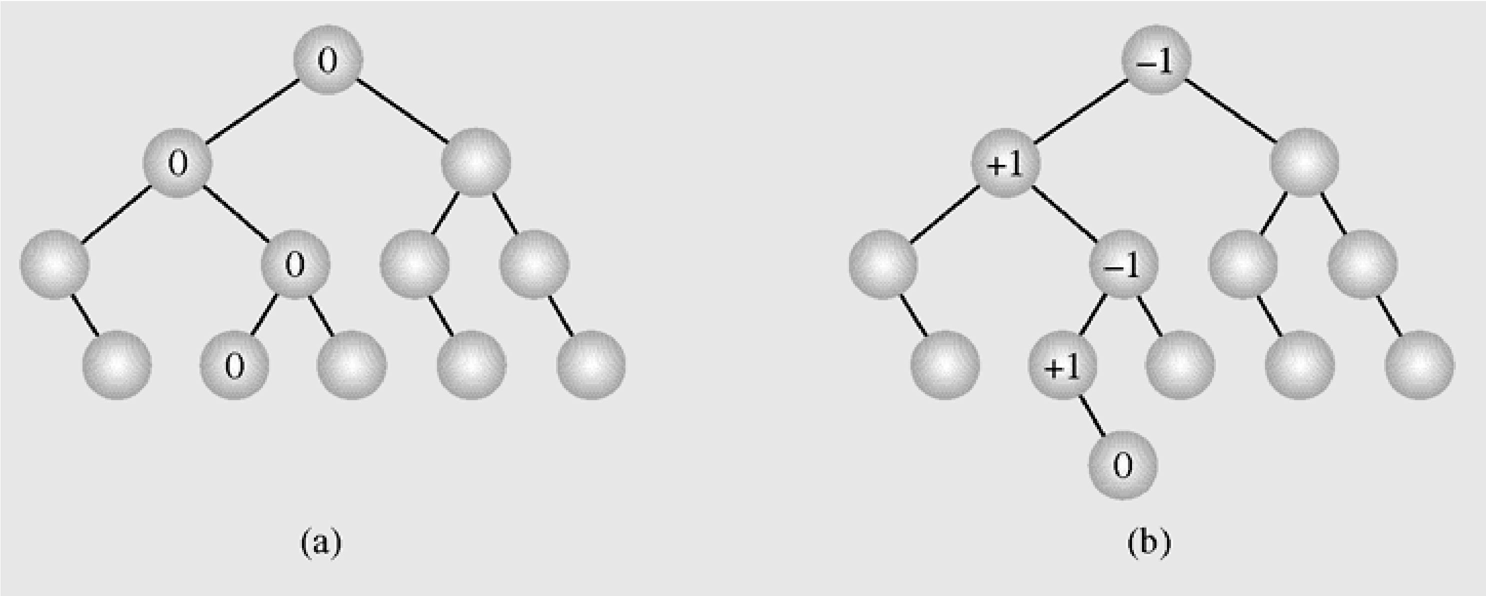
Balancing a tree after insertion of a node in the left subtree of node *Q*

AVL Trees (continued)



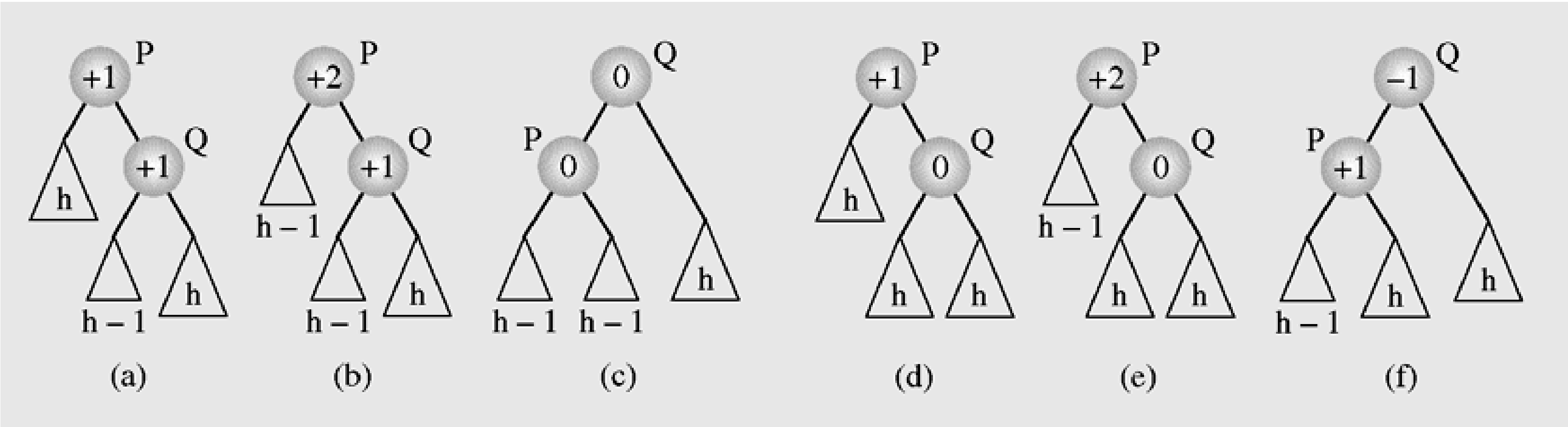
An example of inserting a new node (b) in an AVL tree (a), which requires one rotation (c) to restore the height balance

AVL Trees (continued)



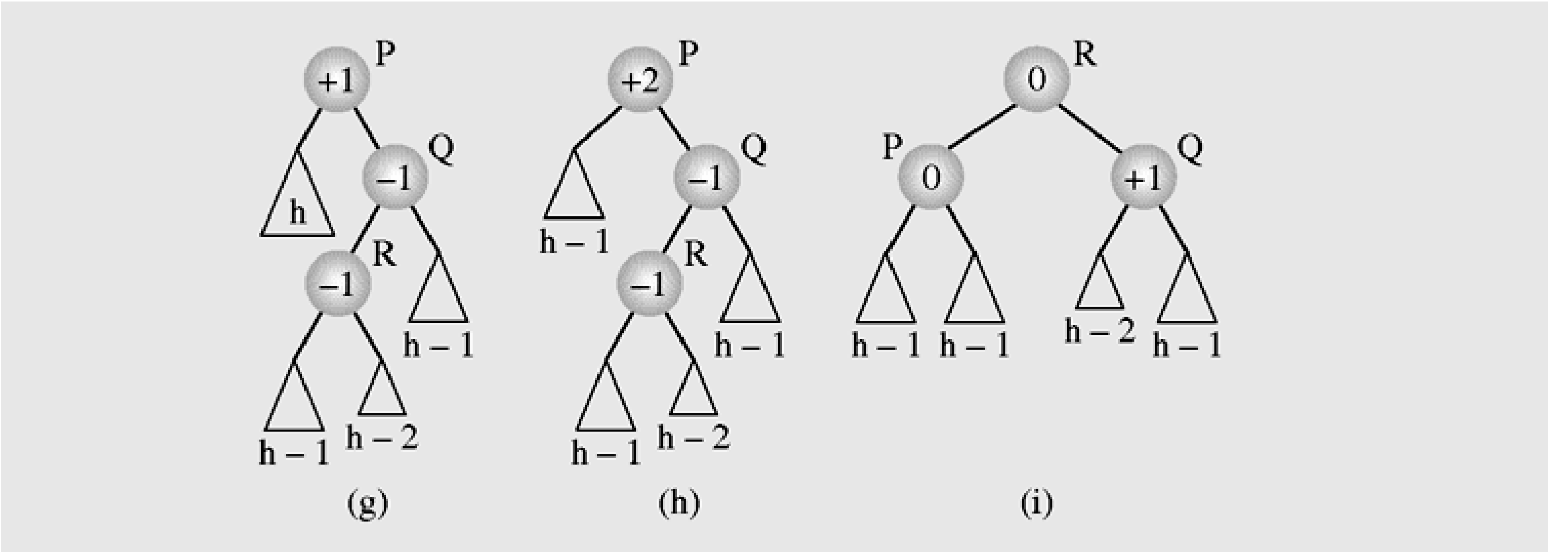
In an AVL tree (a) a new node is inserted (b) requiring no height adjustments

AVL Trees (continued)



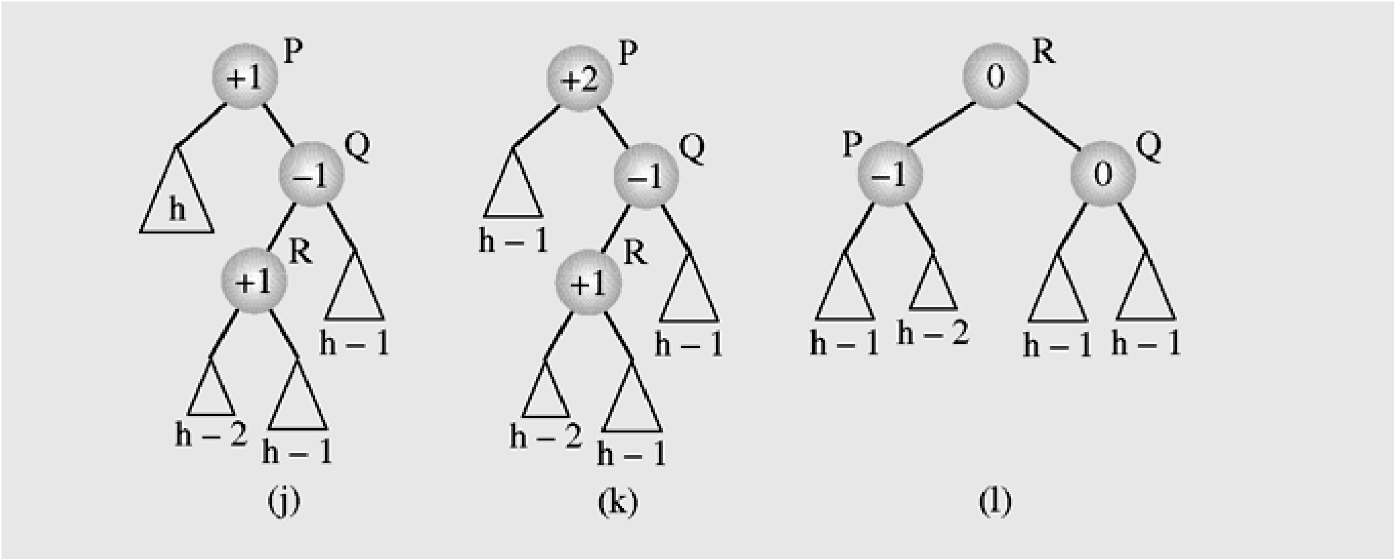
Rebalancing an AVL tree after deleting a node

AVL Trees (continued)



Rebalancing an AVL tree after deleting a node (continued)

AVL Trees (continued)



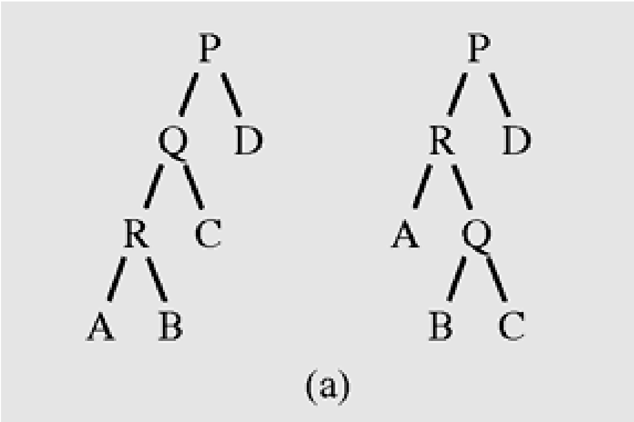
Rebalancing an AVL tree after deleting a node (continued)

Self-Restructuring Trees

- Aka: self-adjusting trees
- Support the same operations as BSTs or balanced BSTs:
  - the main ones: find key, insert key, delete key, find for a natural number i, the i-th key (in size)
- The strategy in self-adjusting trees is to restructure trees by moving up the tree with only those elements that are used more often, and creating a **priority tree**

Self-Restructuring Trees

- **Single rotation** – Rotate a child about its parent if an element in a child is accessed, unless it is the root



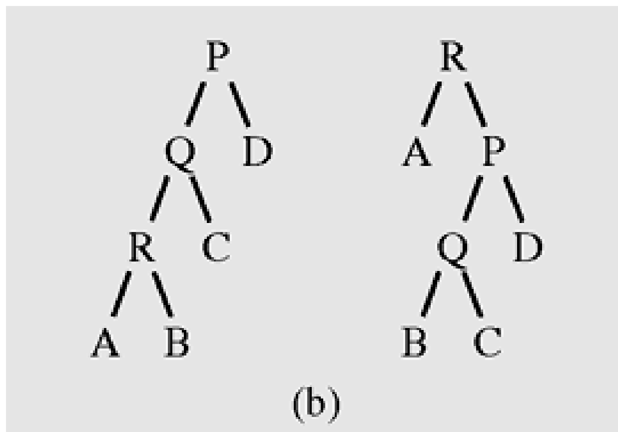
Restructuring a tree by using

(a) a single rotation or

(b) (b) moving to the root when accessing node R

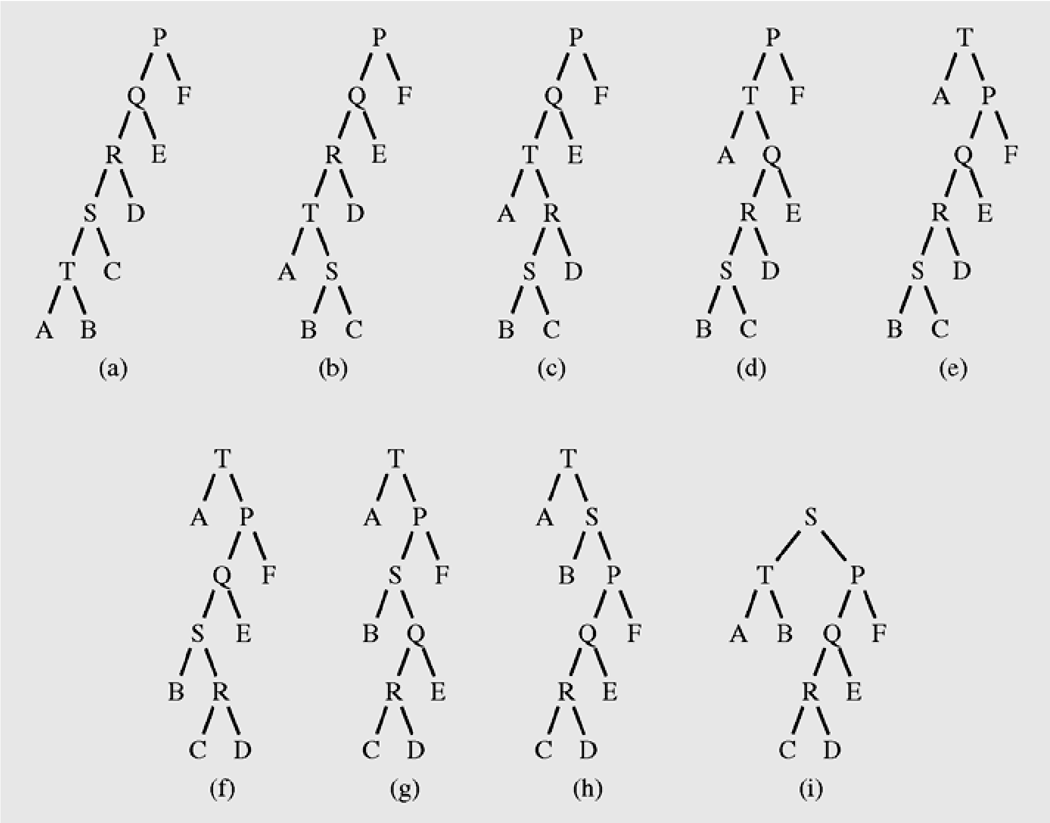
Self-Restructuring Trees (continued)

- **Moving to the root –** Repeat the child–parent rotation until the element being accessed is in the root



Restructuring a tree by using (a) a single rotation or (b) moving to the root when accessing node R (continued)

Self-Restructuring Trees (continued)

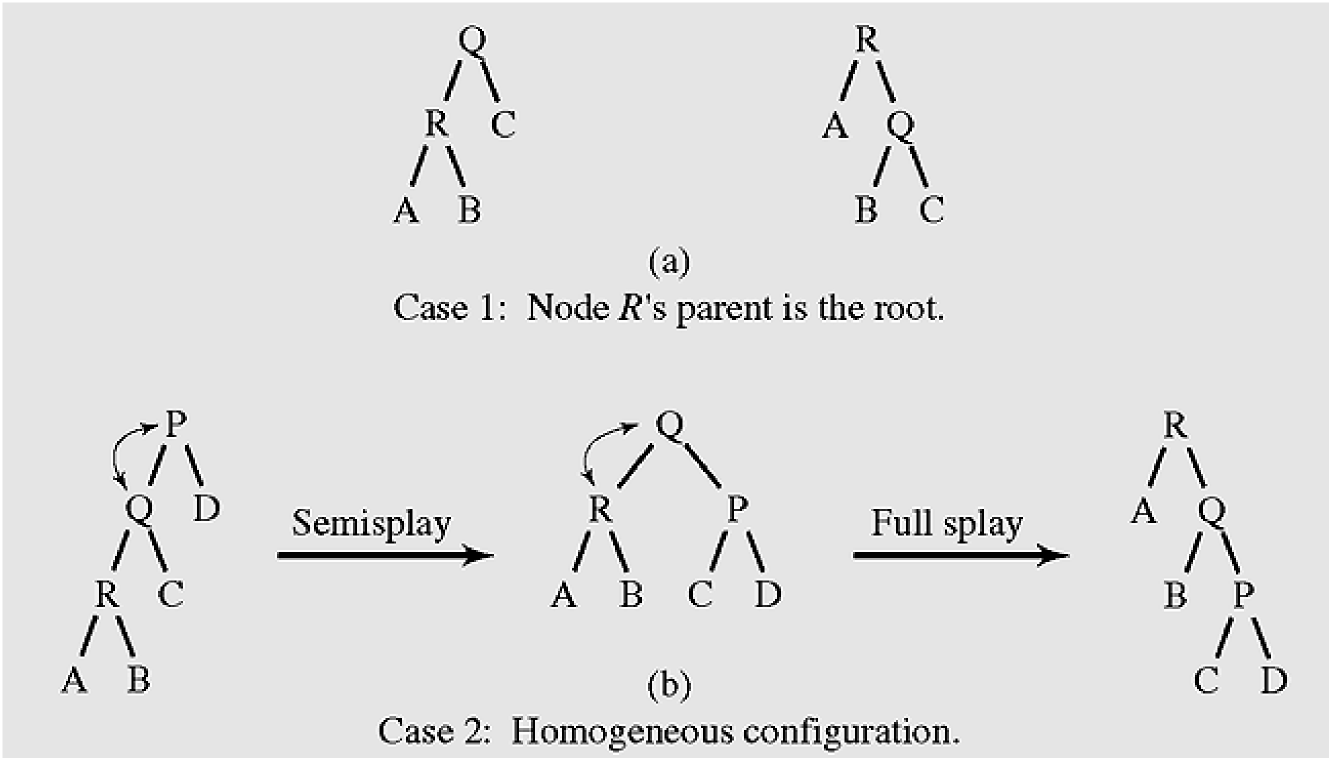


(a–e) Moving element *T* to the root and then (e–i) moving element *S* to the root

Splaying

- A modification of the move-to-the-root strategy is called **splaying**
- Splaying applies single rotations in pairs in an order depending on the links between the child, parent, and grandparent
- **Semisplaying** requires only one rotation for a homogeneous splay and continues splaying with the parent of the accessed node, not with the node itself

Splaying (continued)

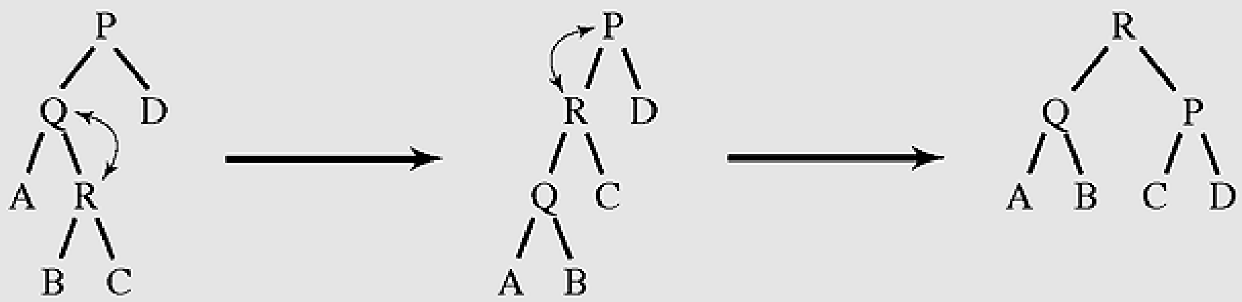


(a)
Case 1: Node *R*'s parent is the root.

(b)
Case 2: Homogeneous configuration.

Examples of splaying

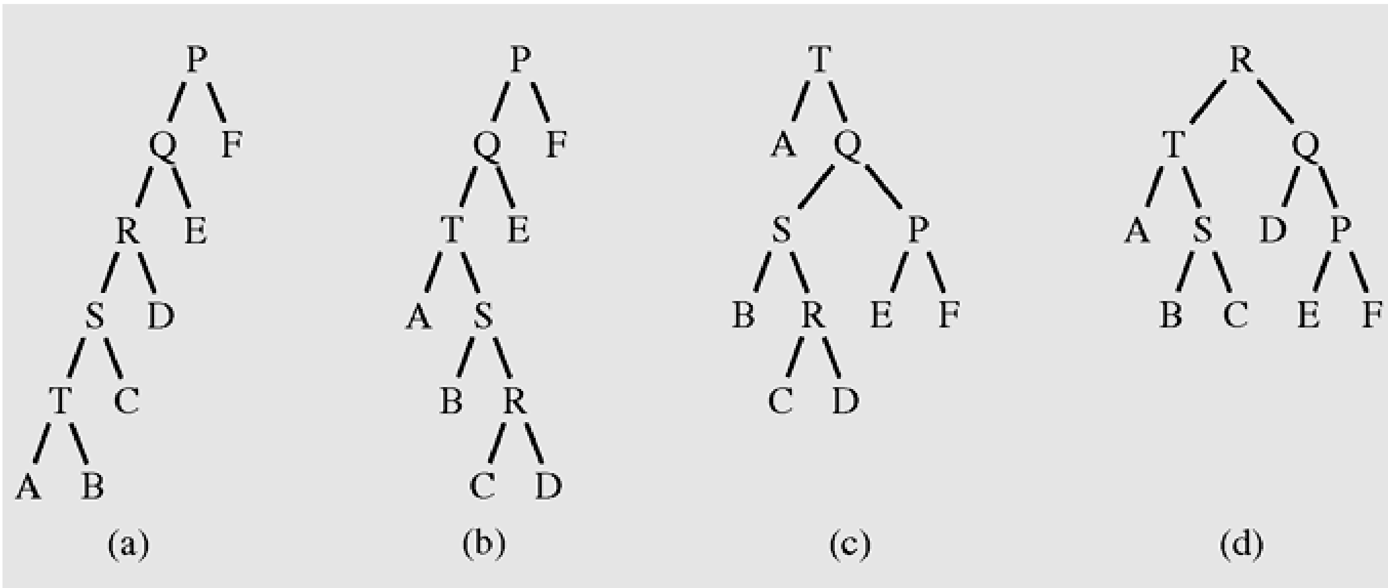
Splaying (continued)



(c)
Case 3: Heterogeneous configuration.

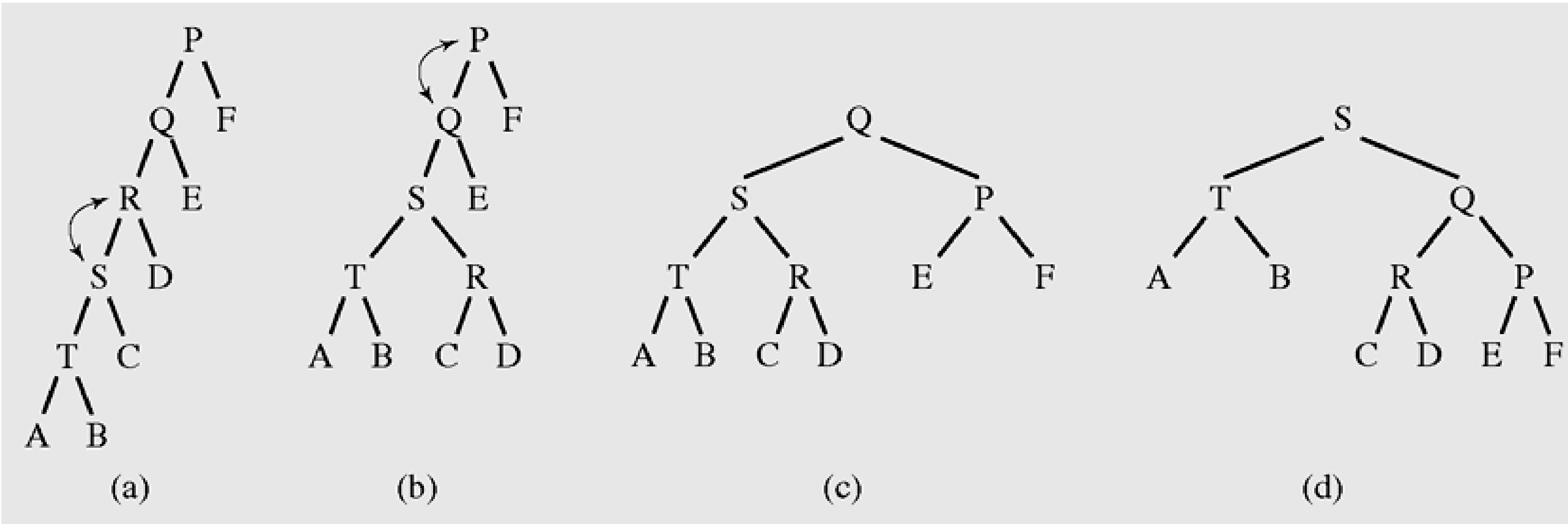
Examples of splaying (continued)

Splaying (continued)



Restructuring a tree with splaying (a–c) after accessing *T* and (c–d) then *R*

Splaying (continued)



(a–c) Accessing *T* and restructuring the tree with semisplaying; (c–d) accessing *T* again

Splay Trees

- Tarjan and Sleator
  - Article (by Sleator and Tarjan, 1985) or gem of a book by R. E. Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics (1983) available upon request
- Support the same operations as BSTs or balanced BSTs or Self-structuring trees: the main ones: find key, insert key, delete key, find for a natural number i, the i-th key (in size)
- A modification of the move-to-the-root strategy is called **splaying**
- Splaying applies single rotations in pairs in an order depending on the links between the child, parent, and grandparent
- **Semisplaying** requires only one rotation for a homogeneous splay and continues splaying with the parent of the accessed node, not with the node itself

Costs?

- Any operation could still require $\Theta(n)$ time, this degenerate behavior cannot occur repeatedly for splay trees, and Sleator and Tarjan proved that any sequence of m operations takes $O(m \log(n))$ worst case time total (n the number of nodes):
- In the long run this data structure behaves as though each operation takes $O(\log(n))$ – this called the **amortized time bound**
- See the aforementioned article or book by Tarjan

Heaps

- A particular kind of binary tree, called a **heap**, has two properties:
  - The value of each node is greater than or equal to the values stored in each of its children
  - The tree is perfectly balanced, and the leaves in the last level are all in the *leftmost* positions
- These two properties define a **max heap**
- If "greater" in the first property is replaced with "less," then the definition specifies a **min heap**

Heaps (continued)

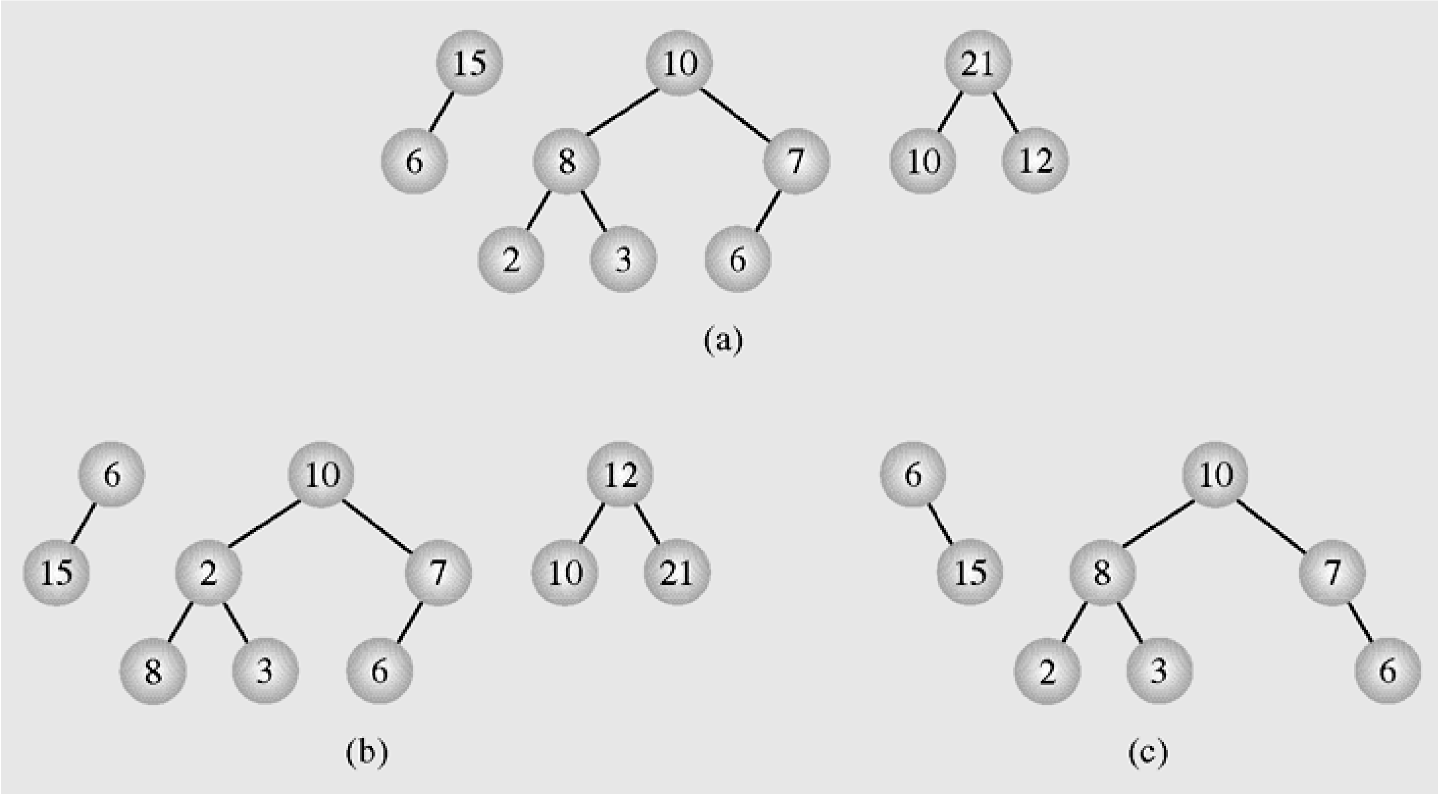


Figure 6-51 Examples of (a) heaps and (b–c) nonheaps

Heaps (continued)

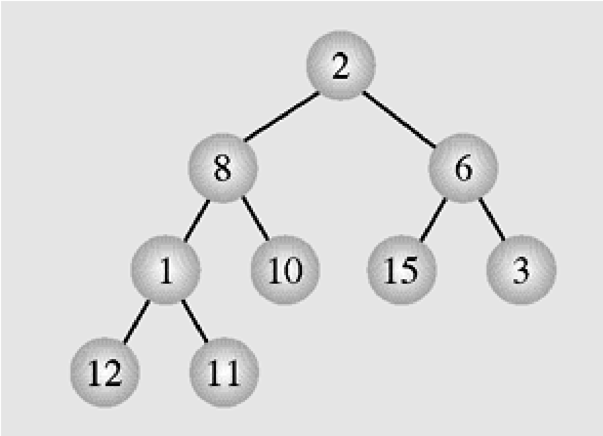


Figure 6-52 The array [2 8 6 1 10 15 3 12 11] seen as a tree

Heaps (continued)

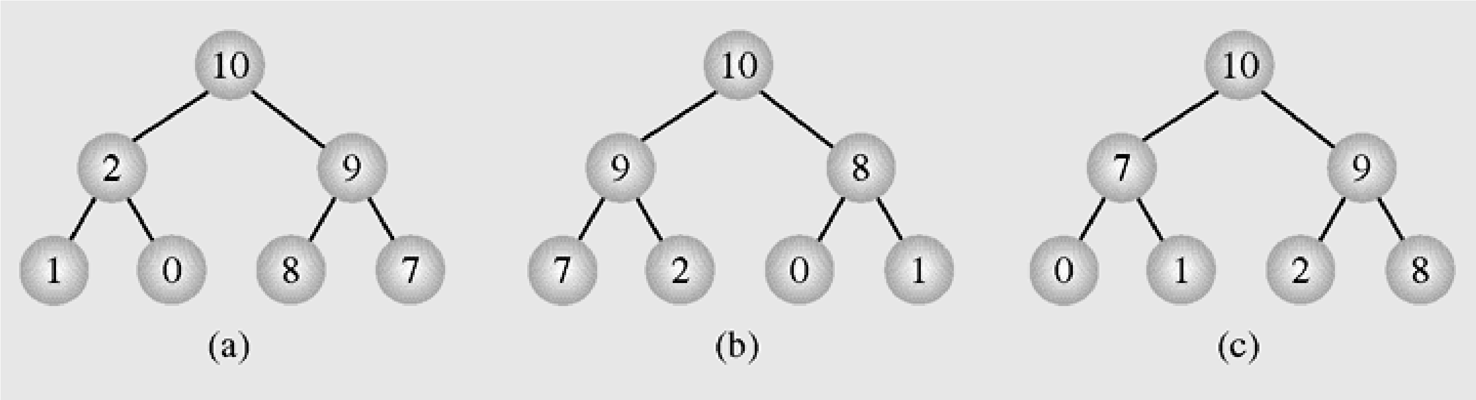


Figure 6-53 Different heaps constructed with the same elements

ADT Priority Queue

PQueueAdd(newItem) // adds a new item to
//the priority queue

PQueueRemove(priorityItem) // removes and
//retrieves from a priority queue the item
//with the highest priority value

createPQueue()

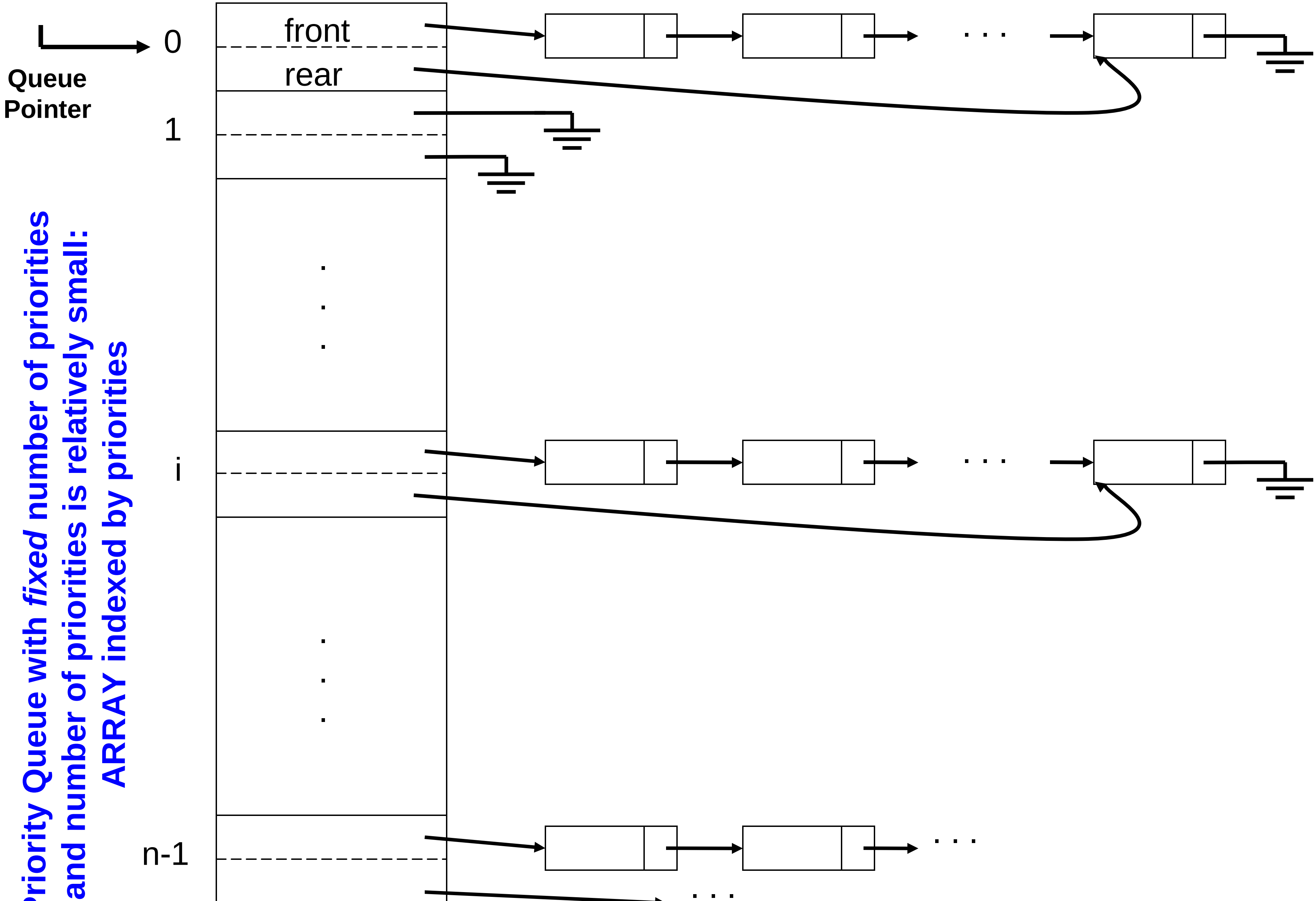
destroyPQueue()

isPQueueEmpty()

Implementations of ADT Priority Queue

- With an array of pointers

Priority Queue with *fixed* number of priorities and number of priorities is relatively small: ARRAY indexed by priorities



Priority headers

Heaps as Priority Queues

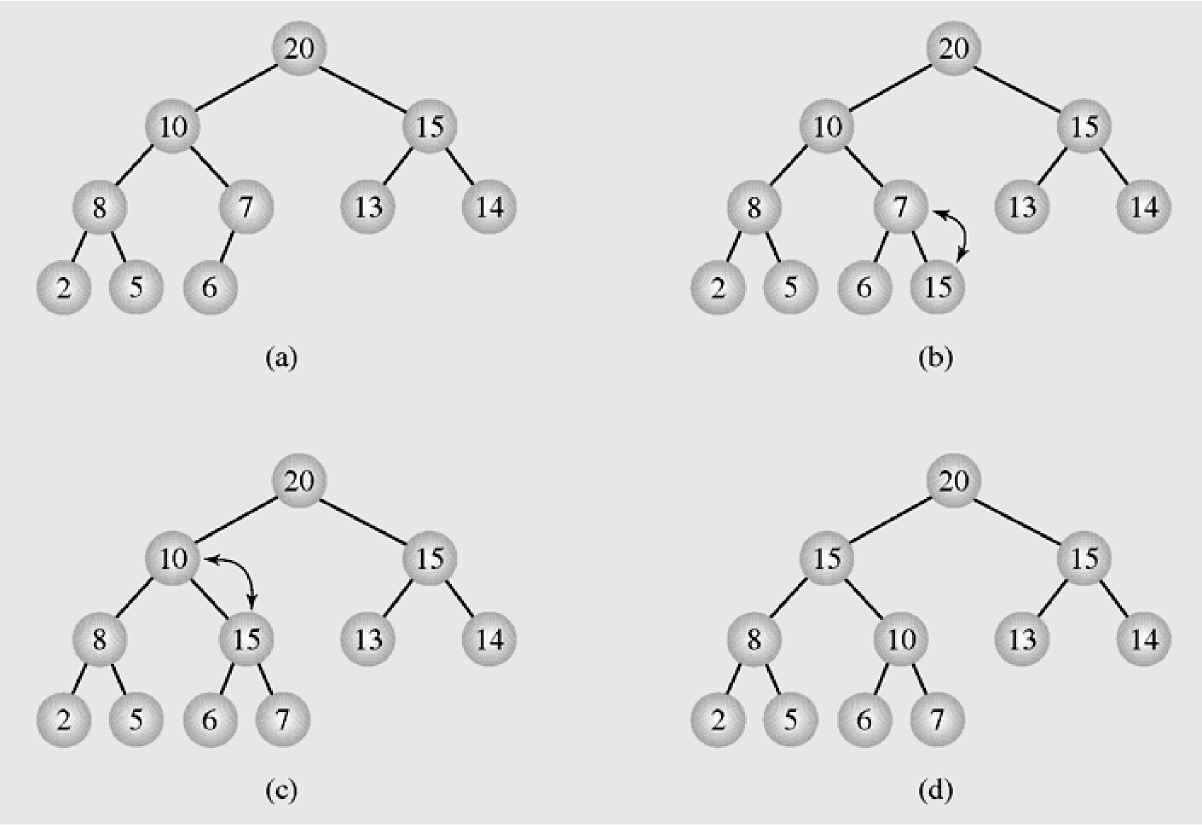


Figure 6-54 Enqueuing an element to a heap

Heaps as Priority Queues (continued)

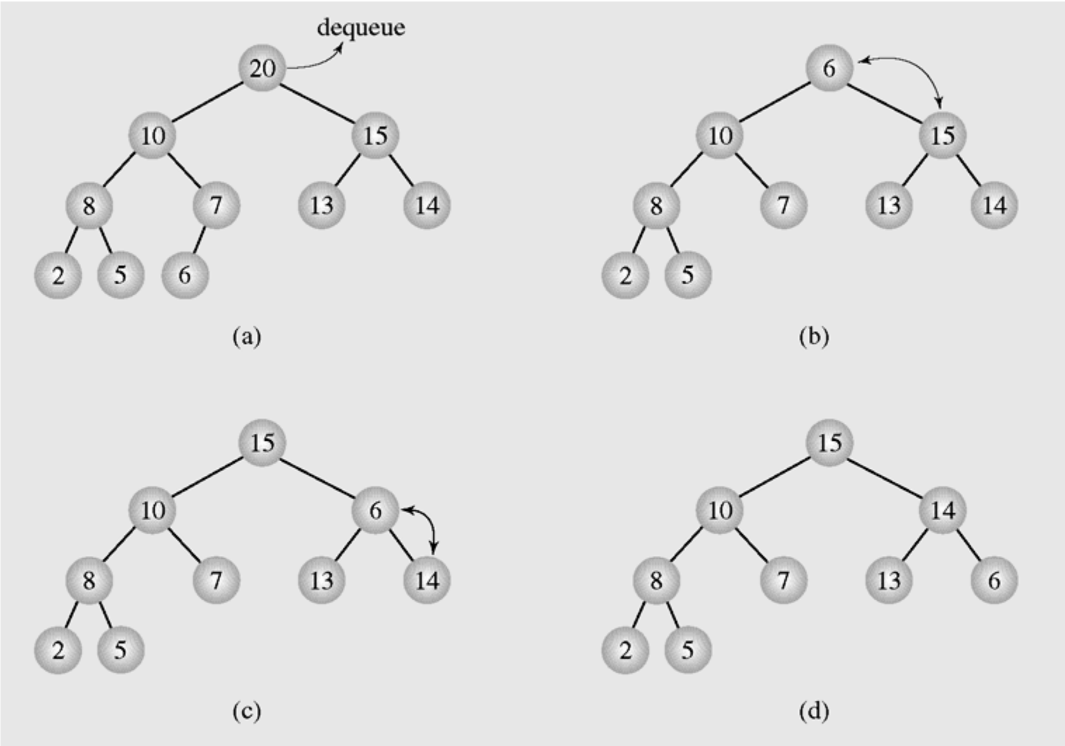


Figure 6-55 Dequeuing an element from a heap

Organizing Arrays as Heaps

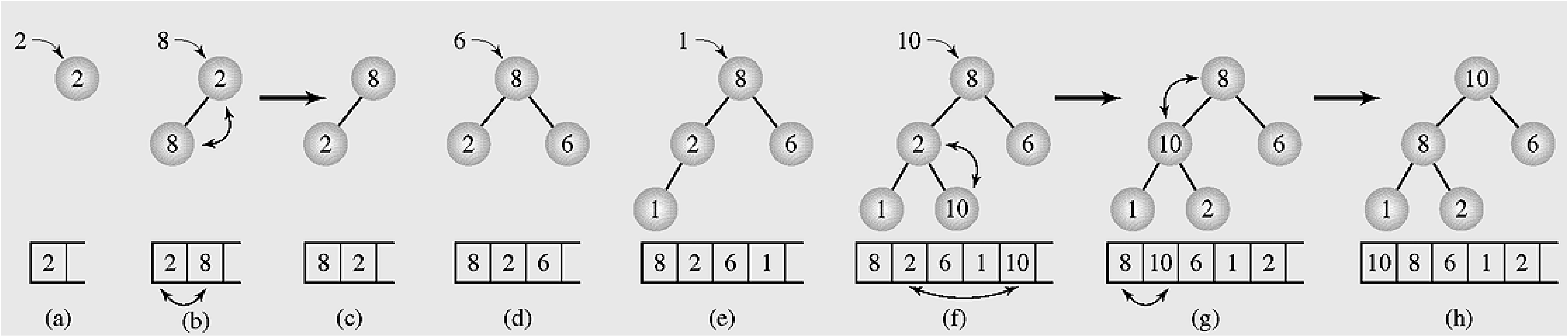


Figure 6-57 Organizing an array as a heap with a top-down method

Organizing Arrays as Heaps

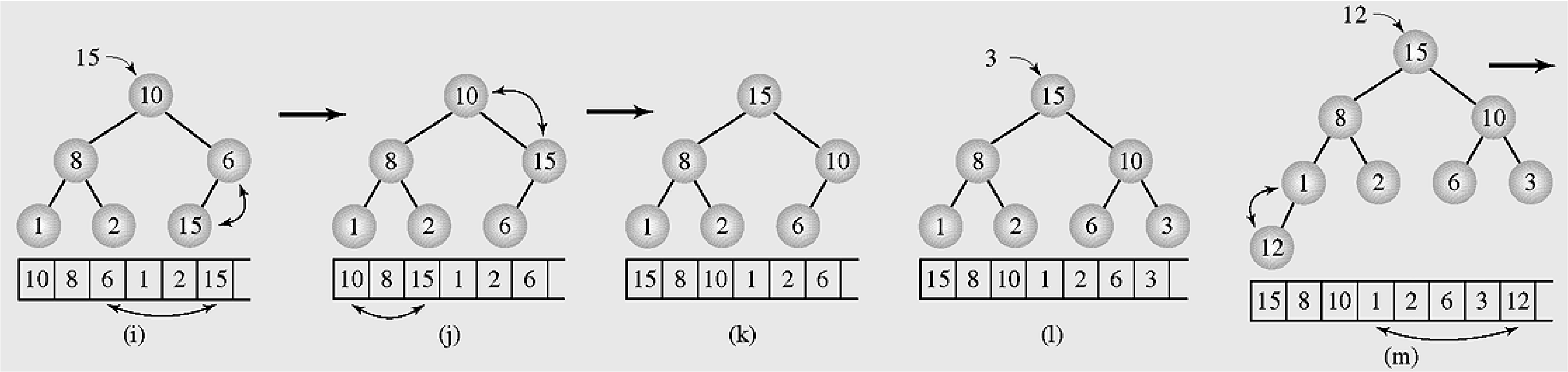


Figure 6-57 Organizing an array as a heap with a top-down method (continued)

Organizing Arrays as Heaps

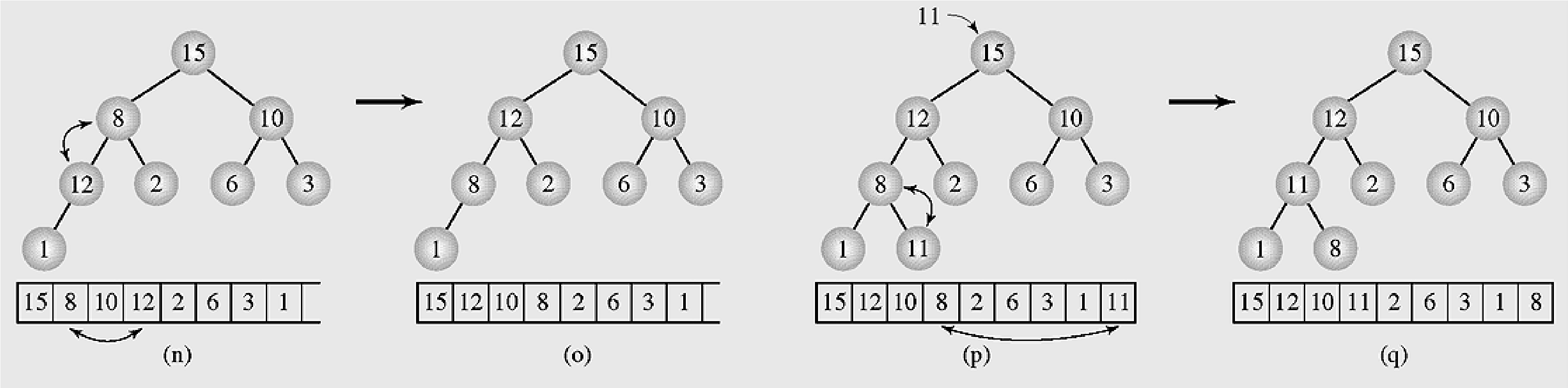


Figure 6-57 Organizing an array as a heap with a top-down method (continued)

Organizing Arrays as Heaps (continued)

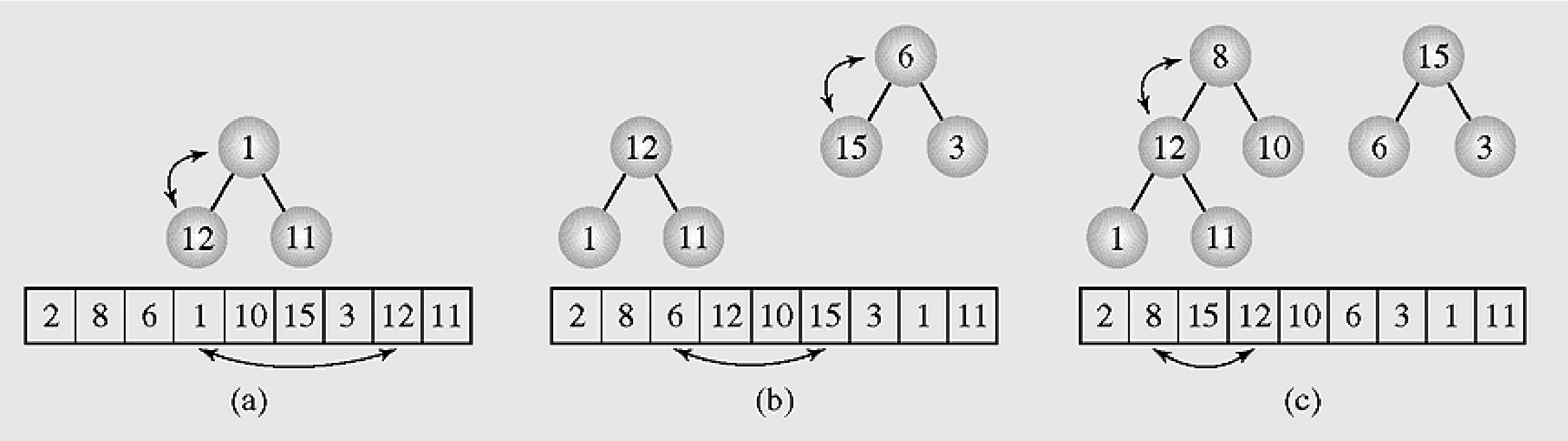


Figure 6-58 Transforming the array [2 8 6 1 10 15 3 12 11] into a heap with a bottom-up method

Organizing Arrays as Heaps (continued)

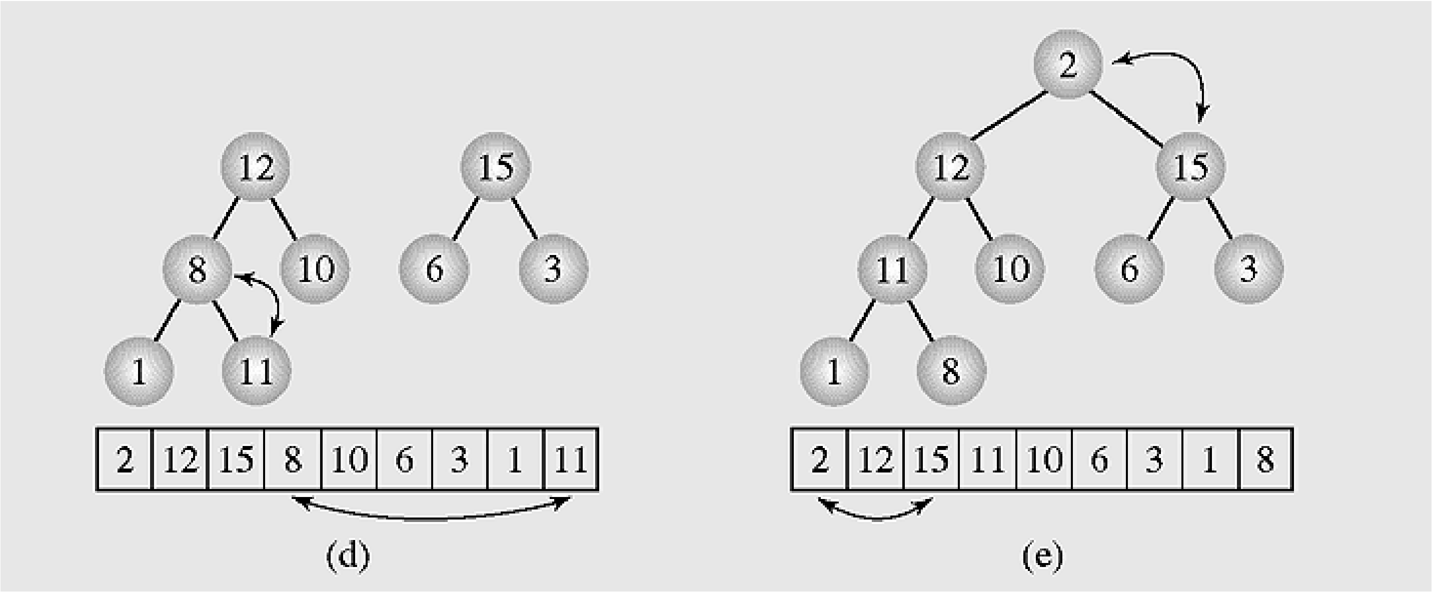


Figure 6-58 Transforming the array [2 8 6 1 10 15 3 12 11] into a heap with a bottom-up method (continued)

Organizing Arrays as Heaps (continued)

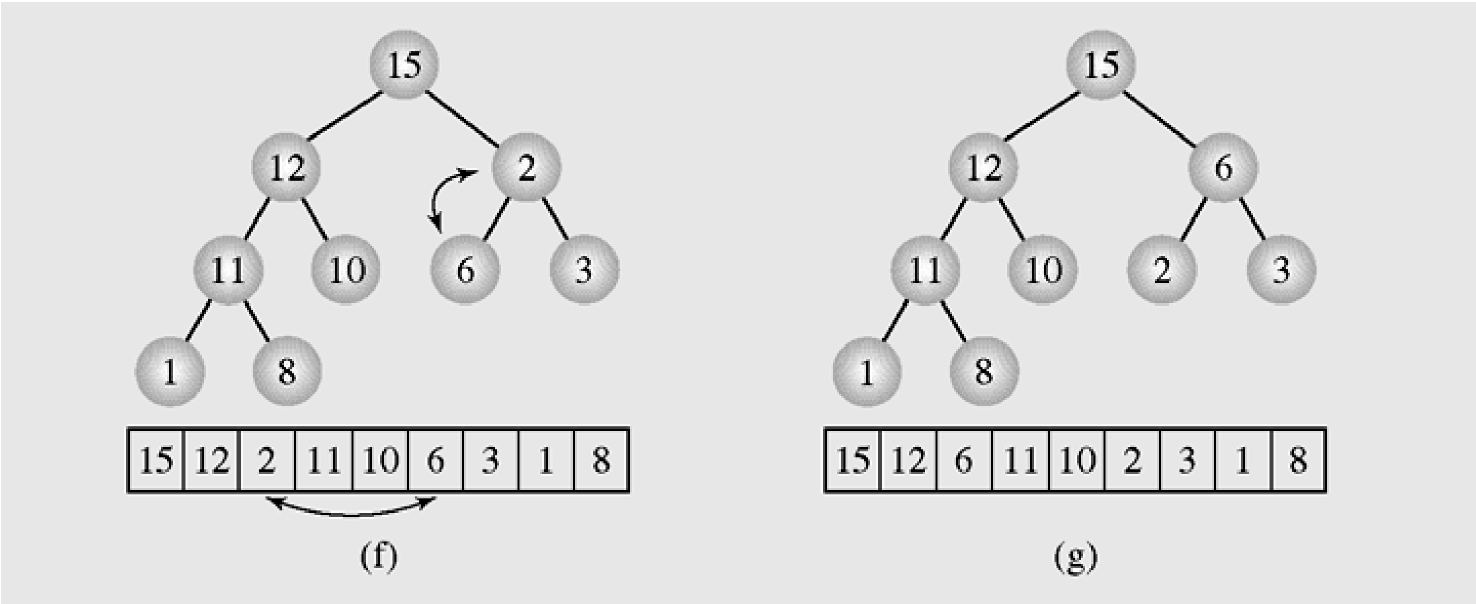


Figure 6-58 Transforming the array [2 8 6 1 10 15 3 12 11] into a heap with a bottom-up method (continued)

Polish Notation and Expression Trees

- **Polish notation** is a special notation for propositional logic that eliminates all parentheses from formulas
- The compiler rejects everything that is not essential to retrieve the proper meaning of formulas rejecting it as “syntactic sugar”

Polish Notation and Expression Trees (continued)

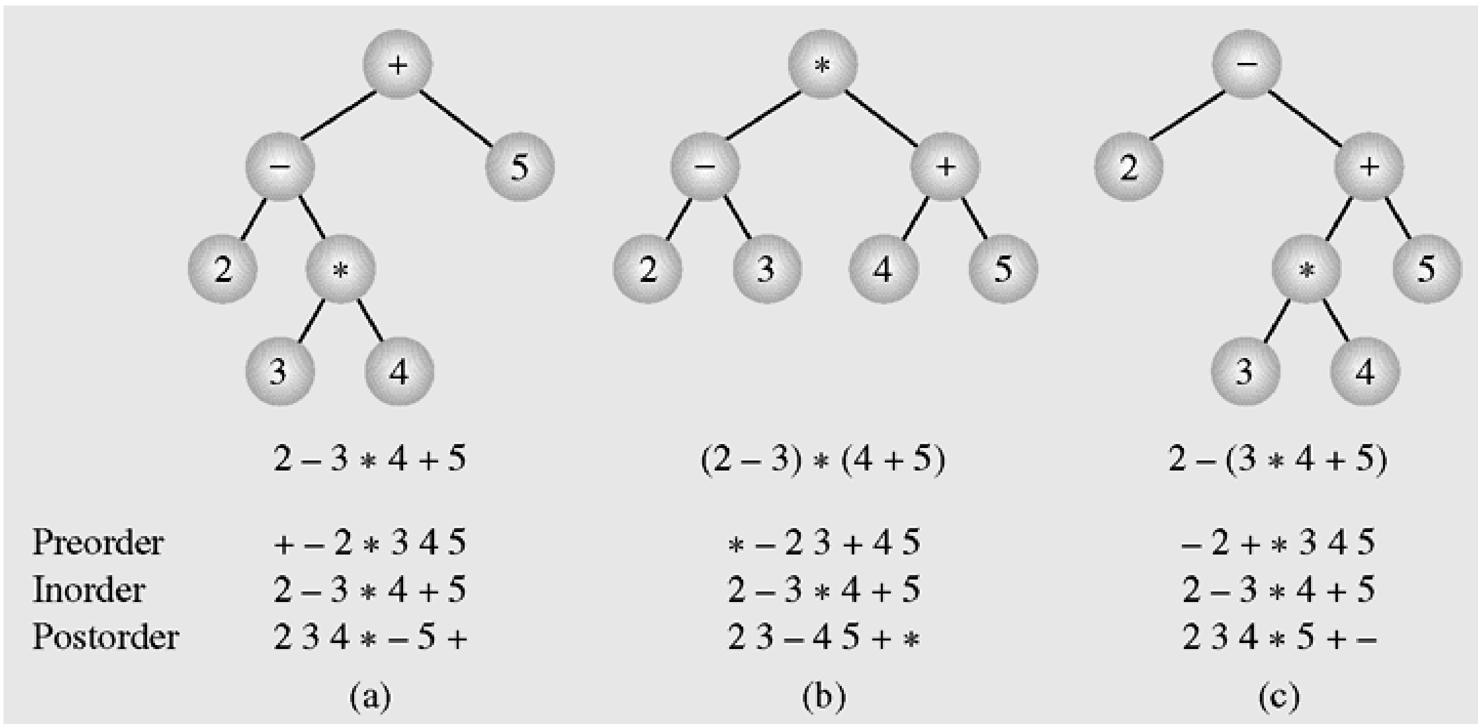


Figure 6-59 Examples of three expression trees and results of their traversals

Summary

- A tree is a data type that consists of nodes and arcs.
- The root is a node that has no parent; it can have only child nodes.
- Each node has to be reachable from the root through a unique sequence of arcs, called a path.
- An orderly tree is where all elements are stored according to some predetermined criterion of ordering.

Summary (continued)

- A binary tree is a tree whose nodes have two children (possibly empty), and each child is designated as either a left child or a right child.
- A decision tree is a binary tree in which all nodes have either zero or two nonempty children.
- Tree traversal is the process of visiting each node in the tree exactly one time.
- Threads are references to the predecessor and successor of the node according to an inorder traversal.

Summary (continued)

- An AVL tree is one in which the height of the left and right subtrees of every node differ by at most one.
- A modification of the move-to-the-root strategy is called splaying.
- The complexity of the top-down array-heapify is O(n log (n)); the bottom-up version has complexity O(n).
- Polish notation is a special notation for propositional logic that eliminates all parentheses from formulas.