# Balanced Search Trees

- The efficiency of the binary search tree implementation of the ADT table is related to the tree's height
  - Height of a binary search tree of n items
    - Maximum: n
    - Minimum: $\lceil \log_2(n + 1) \rceil$
- Height of a binary search tree is sensitive to the order of insertions and deletions
- Variations of the binary search tree
  - Can retain their balance despite insertions and deletions




# 2-3 Trees

- A 2-3 tree
  - Has 2-nodes and 3-nodes
    - A 2-node
      - A node with one data item and two children
    - A 3-node
      - A node with two data items and three children
  - All leaves at the same level
  - Is not a binary tree
  - Is never taller than a minimum-height binary tree
    - A 2-3 tree with n nodes never has height greater than $\lceil \log_2(n + 1) \rceil$




# 2-3 Trees

- Rules for placing data items in the nodes of a 2-3 tree
  - A 2-node must contain a single data item whose search key is
    - Greater than the left child's search key
    - Less than the right child's search key
  - A 3-node must contain two data items whose search keys S and L satisfy the following
    - S is
      - Greater than the left child's search key
      - Less than the middle child's search key
    - L is
      - Greater than the middle child's search key
      - Less than the right child's search key
  - A leaf may contain either one or two data items




# 2-3 Trees

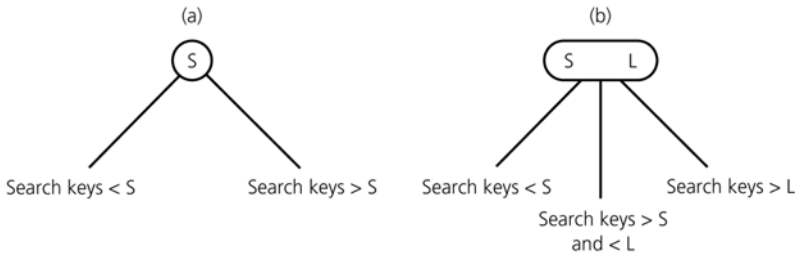



Figure 13-3

Nodes in a 2-3 tree a) a 2-node; b) a 3-node

# 2-3 Trees

- Traversing a 2-3 tree
  - To traverse a 2-3 tree
    - Perform the analogue of an inorder traversal
- Searching a 2-3 tree
  - Searching a 2-3 tree is as efficient as searching the shortest binary search tree
    - Searching a 2-3 tree is $O(\log_2 n)$
    - Number of comparisons required to search a 2-3 tree for a given item
      - Approximately equal to the number of comparisons required to search a binary search tree that is as balanced as possible




# 2-3 Trees

- Advantage of a 2-3 tree over a balanced binary search tree
  - Maintaining the balance of a binary search tree is difficult
  - Maintaining the balance of a 2-3 tree is relatively easy




# 2-3 Trees: Inserting Into a 2-3 Tree

- Insertion into a 2-node leaf is simple
- Insertion into a 3-node causes it to divide




# 2-3 Trees: The Insertion Algorithm

- To insert an item I into a 2-3 tree
  - Locate the leaf at which the search for I would terminate
  - Insert the new item I into the leaf
  - If the leaf now contains only two items, you are done
  - If the leaf now contains three items, split the leaf into two nodes, $n_1$ and $n_2$

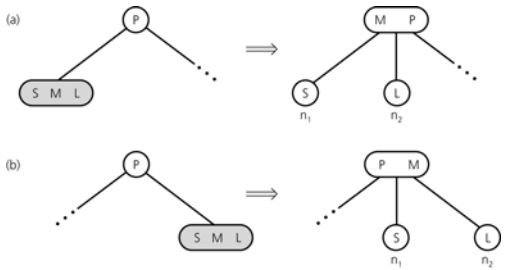



Figure 13-12
Splitting a leaf in a 2-3 tree

# 2-3 Trees: The Insertion Algorithm

- When an internal node contains three items
  - Split the node into two nodes
  - Accommodate the node's children

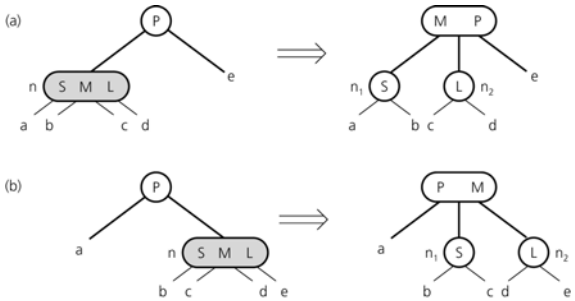



Figure 13-13

Splitting an internal node in a 2-3 tree





# 2-3 Trees: The Insertion Algorithm

- When the root contains three items
  - Split the root into two nodes
  - Create a new root node

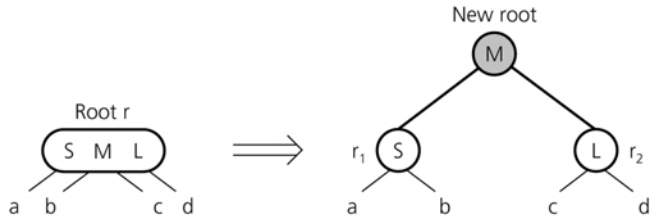


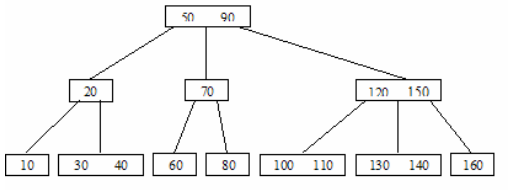

Figure 13-14

Splitting the root of a 2-3 tree

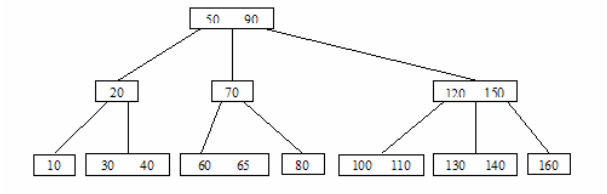




# 2-3 Tree insertion examples




Add 65

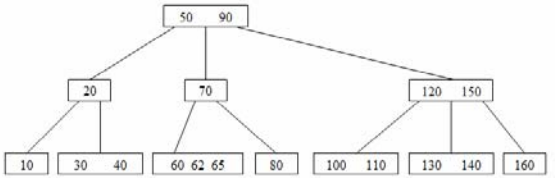


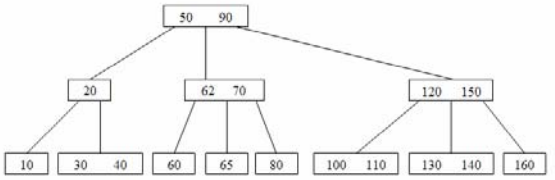





# 2-3 Tree insertion examples

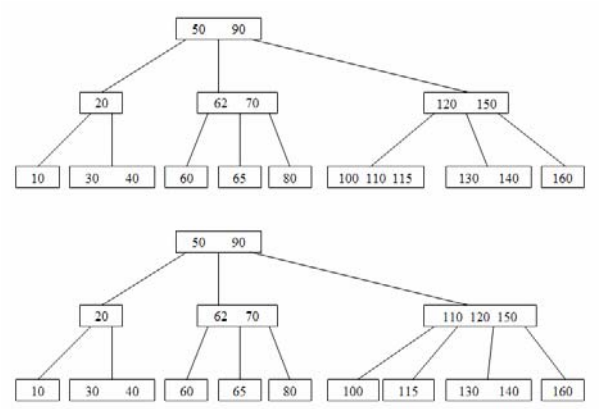
Add 62



# 2-3 Tree insertion examples

Add 115

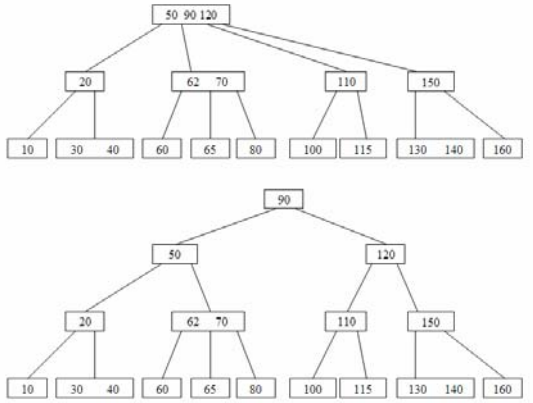






# 2-3 Tree insertion examples

Add 115 continued







# 2-3 Trees: Deleting from a 2-3 Tree

- Deletion from a 2-3 tree
  - Does not affect the balance of the tree
- Deletion from a balanced binary search tree
  - May cause the tree to lose its balance




# 2-3 Trees Efficiency

- When analyzing the efficiency of the `insertItem` and `deleteItem` algorithms, it is sufficient to consider only the time required to locate the item
- A 2-3 implementation of a table is $O(\log_2 n)$ for all table operations
- A 2-3 tree is a compromise
  - Searching a 2-3 tree is not quite as efficient as searching a binary search tree of minimum height
  - A 2-3 tree is relatively simple to maintain

# AVL Trees

- An AVL tree (Adelson-Velsky and Landis)
  - A balanced binary search tree
  - Can be searched almost as efficiently as a minimum-height binary search tree
  - Maintains a height close to the minimum
  - Requires far less work than would be necessary to keep the height exactly equal to the minimum
- Basic strategy of the AVL method
  - After each insertion or deletion
    - Check whether the tree is still balanced
    - If the tree is unbalanced, restore the balance




# AVL Trees

- Rotations
  - Restore the balance of a tree
  - Two types
    - Single rotation
    - Double rotation

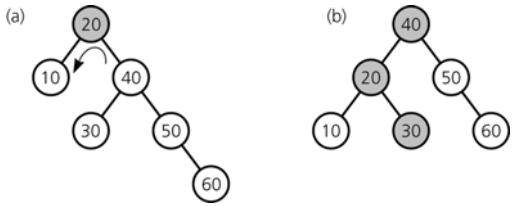



Figure 13-38

a) An unbalanced binary search tree; b) a balanced tree after a single left rotation




# AVL Trees

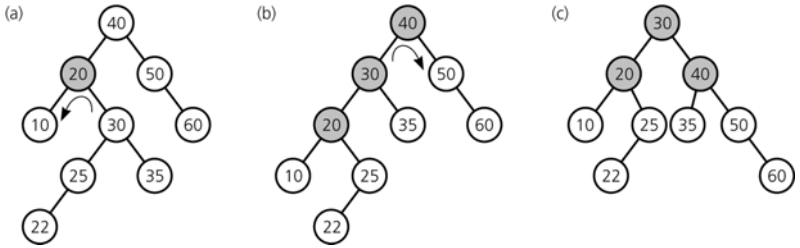



Figure 13-42

a) Before; b) during; and c) after a double rotation




# AVL Trees

- Advantage
  - Height of an AVL tree with n nodes is always very close to the theoretical minimum
- Disadvantage
  - An AVL tree implementation of a table is more difficult than other implementations