Balanced Search Tree

Balanced + *Binary Search Tree*

Outline

- 2-3 tree
- 2-3-4 tree
- AVL tree
  - [Adelson-Velskii & Landis, 1962]
- Red-black tree
  - [Rudolf Bayer, 1972]… B-tree

Search?

The ADT *table*

Data Structures

- **The ADT *table*, or *dictionary***
  - Uses a *search key* to identify its items
  - Its items are *records* that contain several pieces of data

City	Country	Population
Athens	Greece	2,500,000
Barcelona	Spain	1,800,000
Cairo	Egypt	9,500,000
London	England	9,400,000
New York	U.S.A.	7,300,000
Paris	France	2,200,000
Rome	Italy	2,800,000
Toronto	Canada	3,200,000
Venice	Italy	300,000

The ADT *table*

- Various sets of table operations are possible

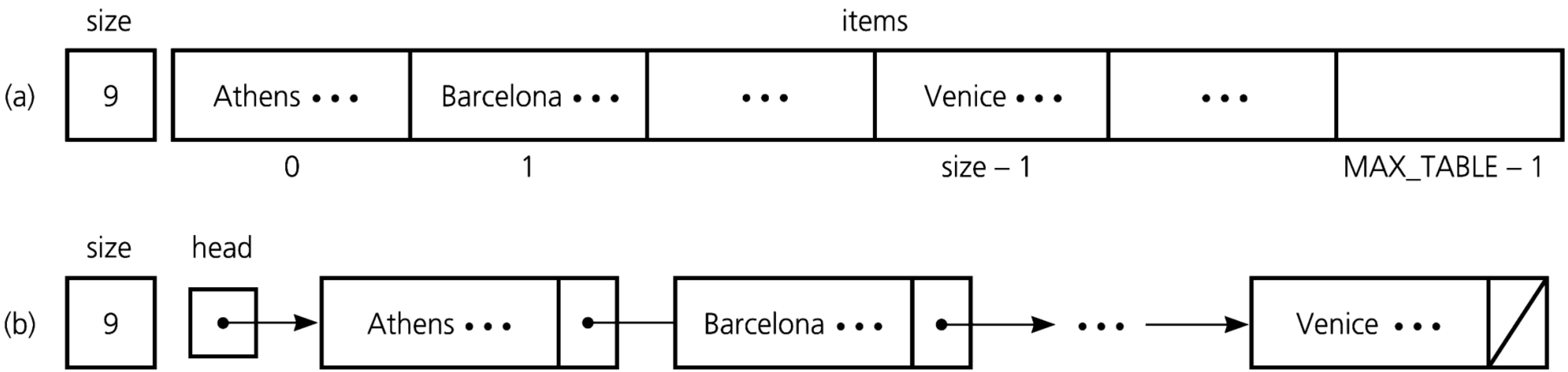
Table
items
createTable() *destroyTable()* *tableIsEmpty()* *tableLength()* *tableInsert()* *tableDelete()* *tableRetrieve()* *traverseTable()*

The ADT *table*

- **Our table assumes distinct search keys**
  - Other tables could allow duplicate search keys
- **The `traverseTable` operation visits table items in a specified order**
  - One common order is by *sorted* search key

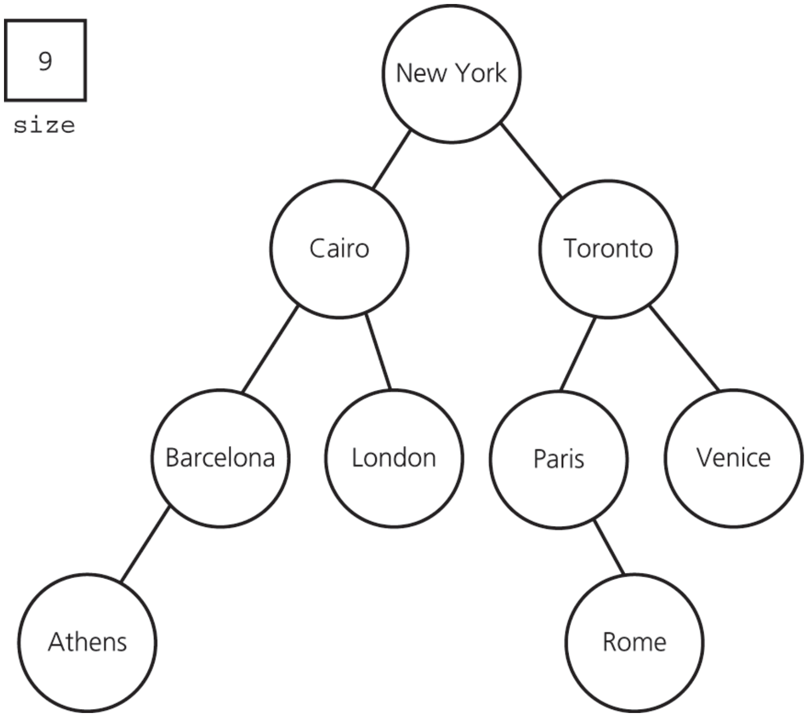
Selecting an Implementation

- **Linear implementations: *Four* categories**
  - Unsorted: *array* based or *pointer* based
  - Sorted (by search key): *array* based or *pointer* based



Selecting an Implementation

- **Nonlinear implementations**
  - Binary search tree implementation
    - **Offers several advantages over linear implementations**



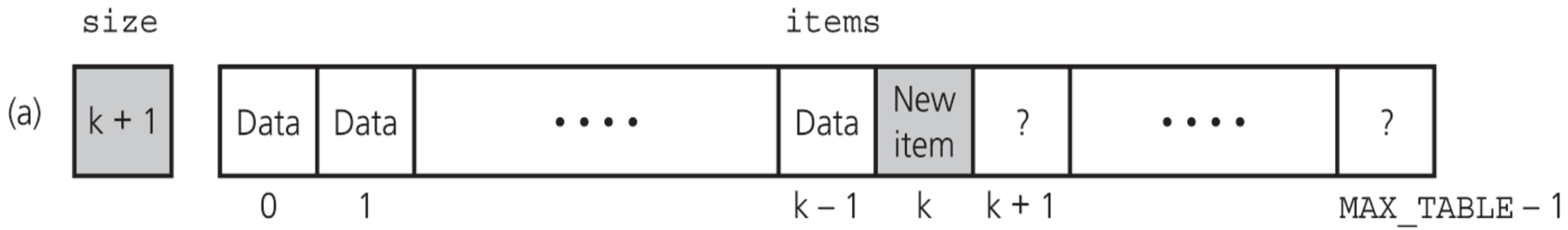
Selecting an Implementation

- **The requirements of a particular application influence the selection of an implementation**
  - Questions to be considered about an application before choosing an implementation
    - ***What operations are needed?***
    - ***How often is each operation required?***
    - ***Are frequently used operations efficient given a particular implementation?***

Comparing Linear Implementations

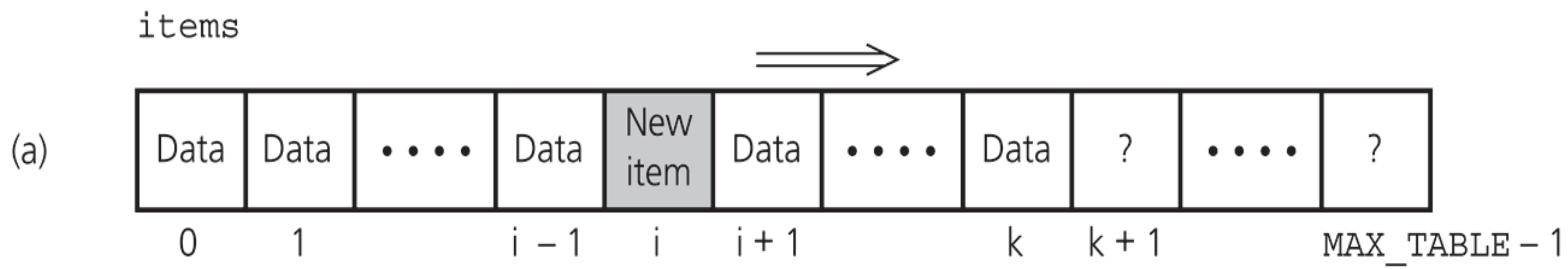
Unsorted array-based implementation

- Insertion is made efficiently after the *last* table item in an array O(?)
- Deletion usually requires shifting data O(?)
- Retrieval requires a sequential search O(?)



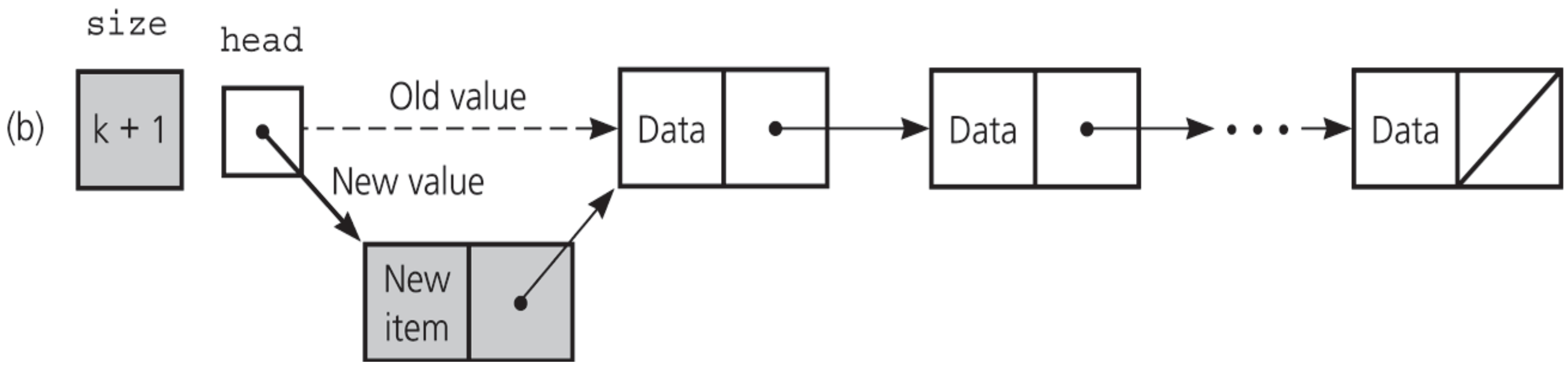
Comparing Linear Implementations

- **Sorted array-based implementation**
  - Both insertions and deletions require shifting data O(?)
  - Retrieval can use an efficient binary search O(?)



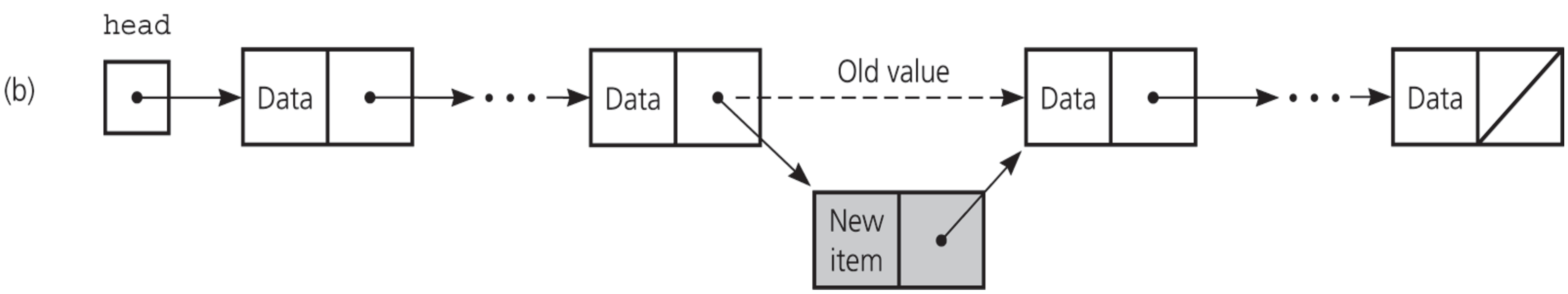
Comparing Linear Implementations

- **Unsorted pointer-based implementation**
  - No data shifts
  - Insertion is made efficiently at the *beginning* of a linked list O(?)
  - Deletion requires a sequential search O(?)
  - Retrieval requires a sequential search O(?)



Comparing Linear Implementations

- **Sorted pointer-based implementation**
  - No data shifts
  - Insertions, deletions, and retrievals each require a sequential search O(?)



Selecting an Implementation

- **Linear**
  - Easy to understand conceptually
  - May be appropriate for *small* tables or *unsorted* tables with few deletions
- **Nonlinear**
  - Is usually a better choice than a linear implementation
  - A *balanced* binary search tree
    - **Increases the efficiency of the table operations**

Selecting an Implementation

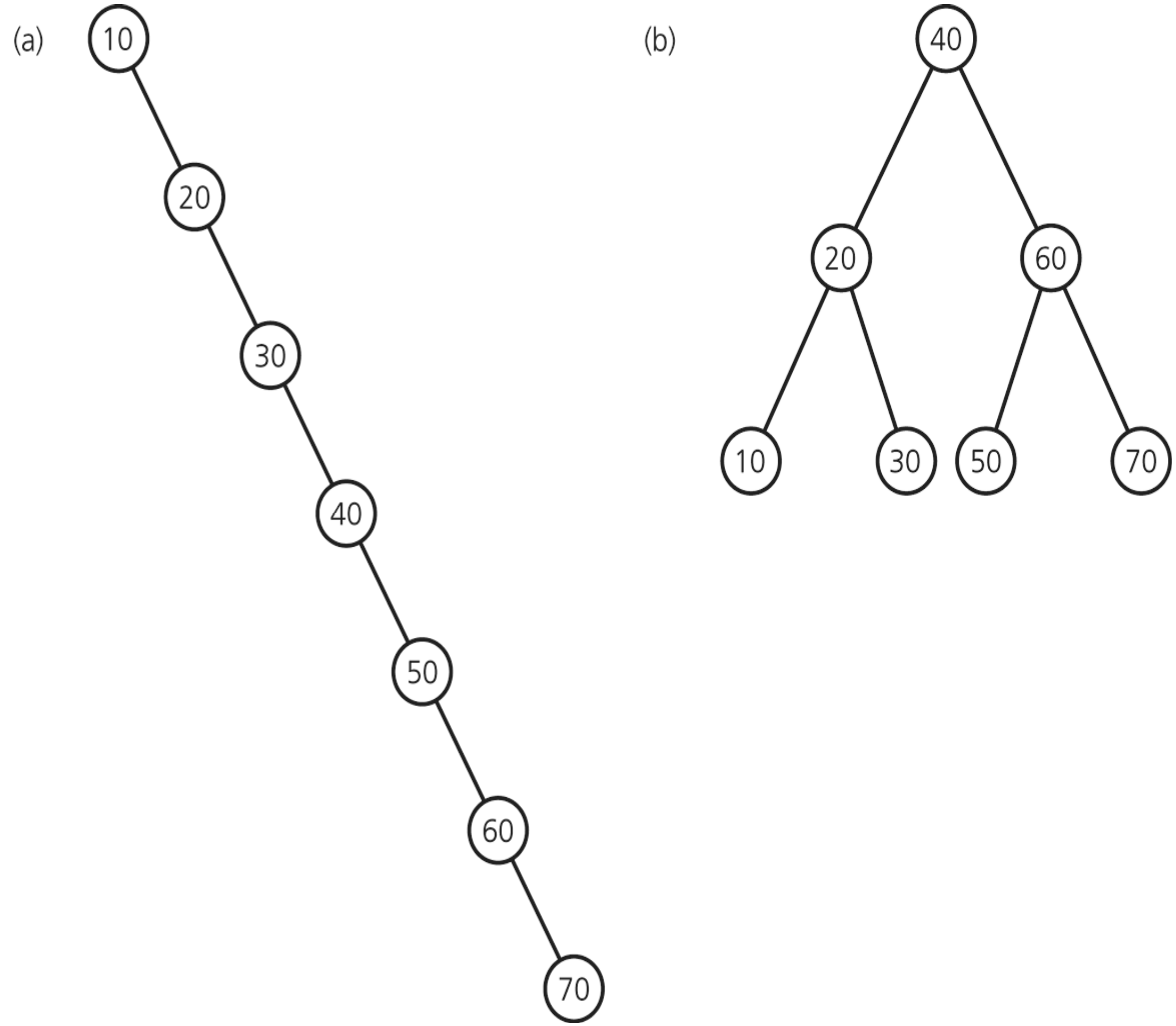
	Insertion	Deletion	Retrieval	Traversal
Unsorted array based	O(1)	O(*n*)	O(*n*)	O(*n*)
Unsorted pointer based	O(1)	O(*n*)	O(*n*)	O(*n*)
Sorted array based	O(*n*)	O(*n*)	O(log *n*)	O(*n*)
Sorted pointer based	O(*n*)	O(*n*)	O(*n*)	O(*n*)
Binary search tree	O(log *n*)	O(log *n*)	O(log *n*)	O(*n*)

Binary Search Tree

Data Structures

- **The efficiency of the binary search tree implementation of the ADT table is related to the tree height**
  - Height of a binary search tree of *n* items
    - Maximum: n
    - Minimum: $\lceil \log_2(n + 1) \rceil$
    - Sensitive to the *order* of insertions and deletions

Binary Search Tree (Input Order?)



Balanced Binary Search Tree

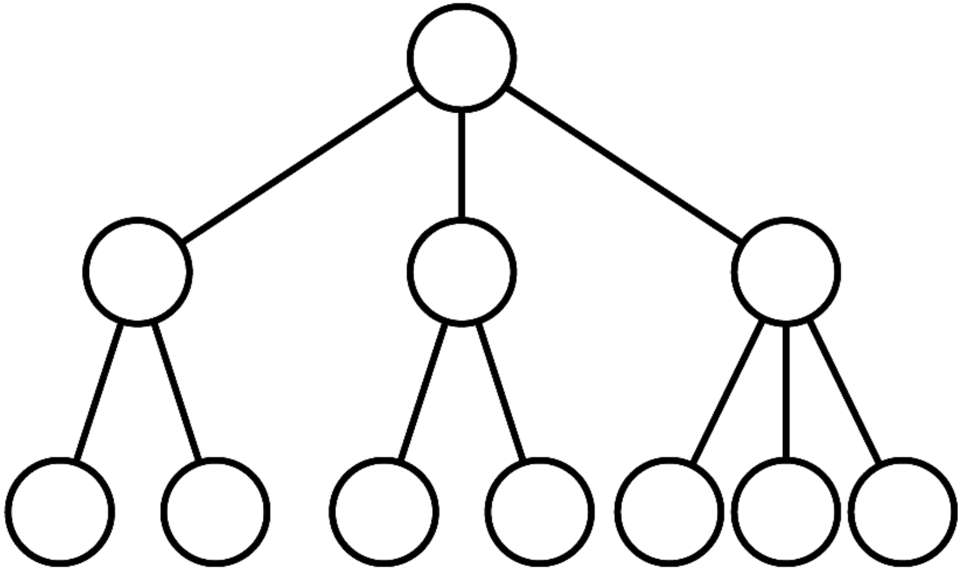
- **Other search trees can retain their *balance* despite insertions and deletions**
  - 2-3 tree
  - 2-3-4 tree
  - AVL tree
  - Red-black tree

2-3 Tree

- **All external nodes (leaves) are at the same level**
- ***Degree* of each internal node = 2 or 3**
  - 2-nodes
  - 3-nodes
- **Main operations**
  - Search == Binary Search Tree
  - Insertion
  - Deletion

2-3 Tree

- **Have 2-nodes and 3-nodes**
  - A 2-node has one data item and two children
  - A 3-node has two data items and three children



2-3 Tree

- **Are general trees, not binary trees**
- **Are never taller than a minimum-height binary tree**
  - A 2-3 tree with n nodes never has height greater than $\lceil \log_2(n + 1) \rceil$

Binary Search Tree vs. 2-3 Tree