Advanced Linear Linked Lists

Skip Lists & Self-Organizing Lists

Advanced Linear Linked Lists

- Why?
  - The **search** operation on linked lists:
    - O(n) time
  - A better search operation?
- How?
  - **Skip lists**
  - **Self-organizing lists**

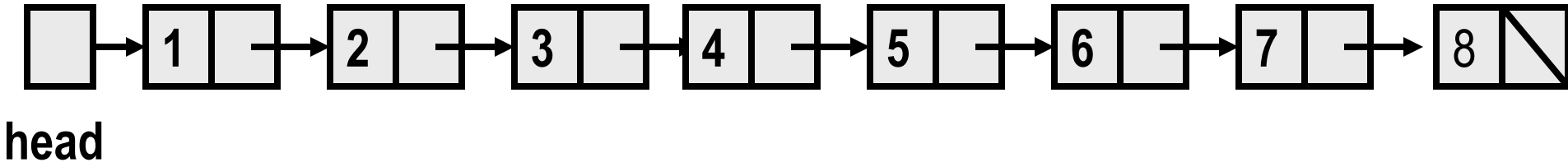
Skip (Linked) Lists – Randomized/Probabilistic Data Structure

Skip Lists

- Problem:
  - Linked lists – The search operation requires O(n) time!
- Goal:
  - **To improve the time complexity of the search operation.**
- Idea:
  - Use **a sorted list with some structural conditions**.
  - Called a **skip** list!

A Sorted (Ordered) Linked List

- A **sorted** linked list:
  - ➔ All nodes are sorted.
- Example:



- **Search?**
  - ➔ **O(n) time (Sequential)**
  - ➔ **O(n) time (Binary)!**

Skip Lists

- **Perfect** skip lists
- **Randomized/Probabilistic** skip lists

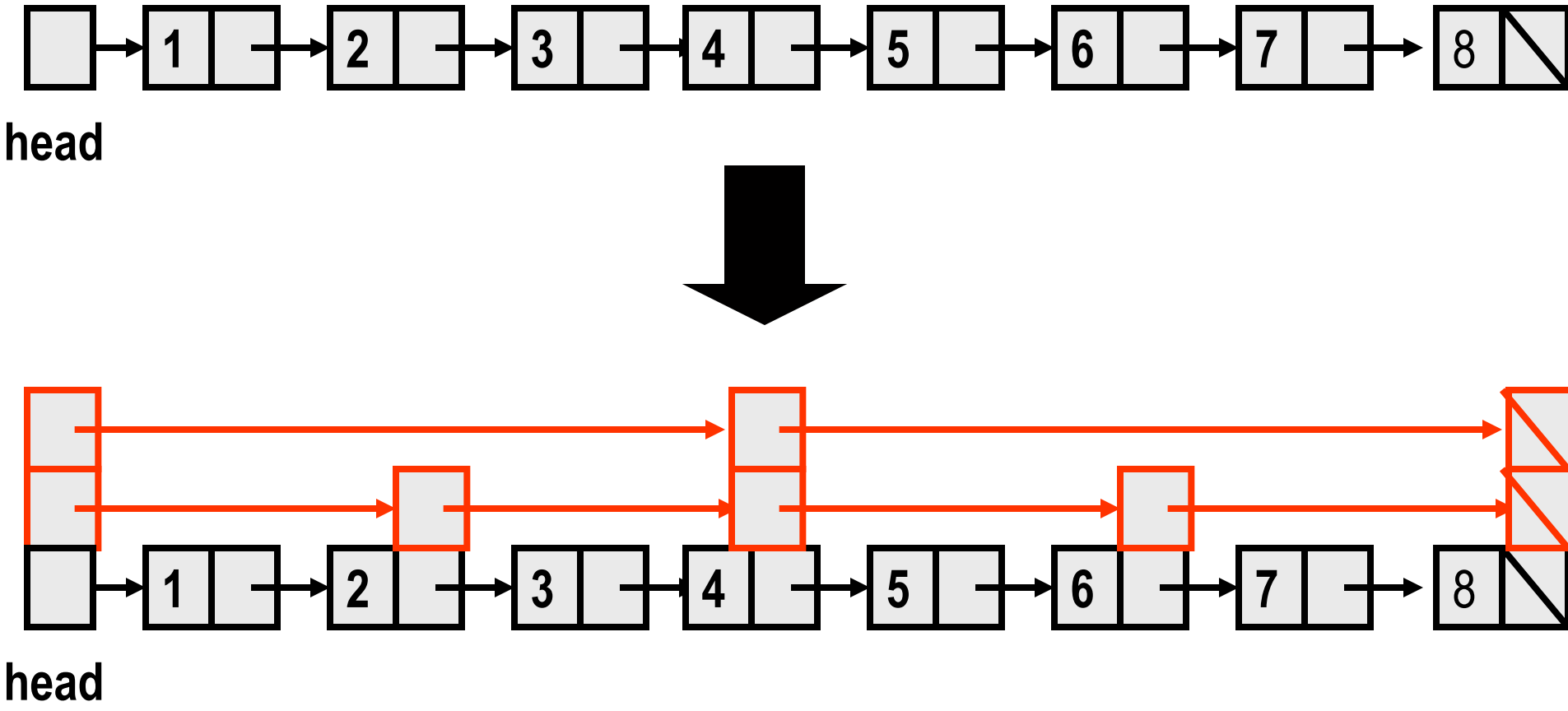
A Perfect Skip List

- A sorted linked list + Additional links with the following structural condition:

Structural Conditions

- Every 2nd node connected.
  - ➔Every 2nd element has 2 pointers.
- Every 4th node connected
  - ➔Every 4th element has 3 pointers.
- Every 8th node connected
  - ➔Every 8th element has 4 pointers.
- And so on …

Example



Structural Conditions

- Every 2^i-th node has a link to the node 2^i ahead of it.
  - Every 2^{nd} node has a link to the node 2 ahead of it.
  - Every 4^{th} node has a link to the node 4 ahead of it.
  - Every 8^{th} node has a link to the node 8 ahead of it
  - And so on …

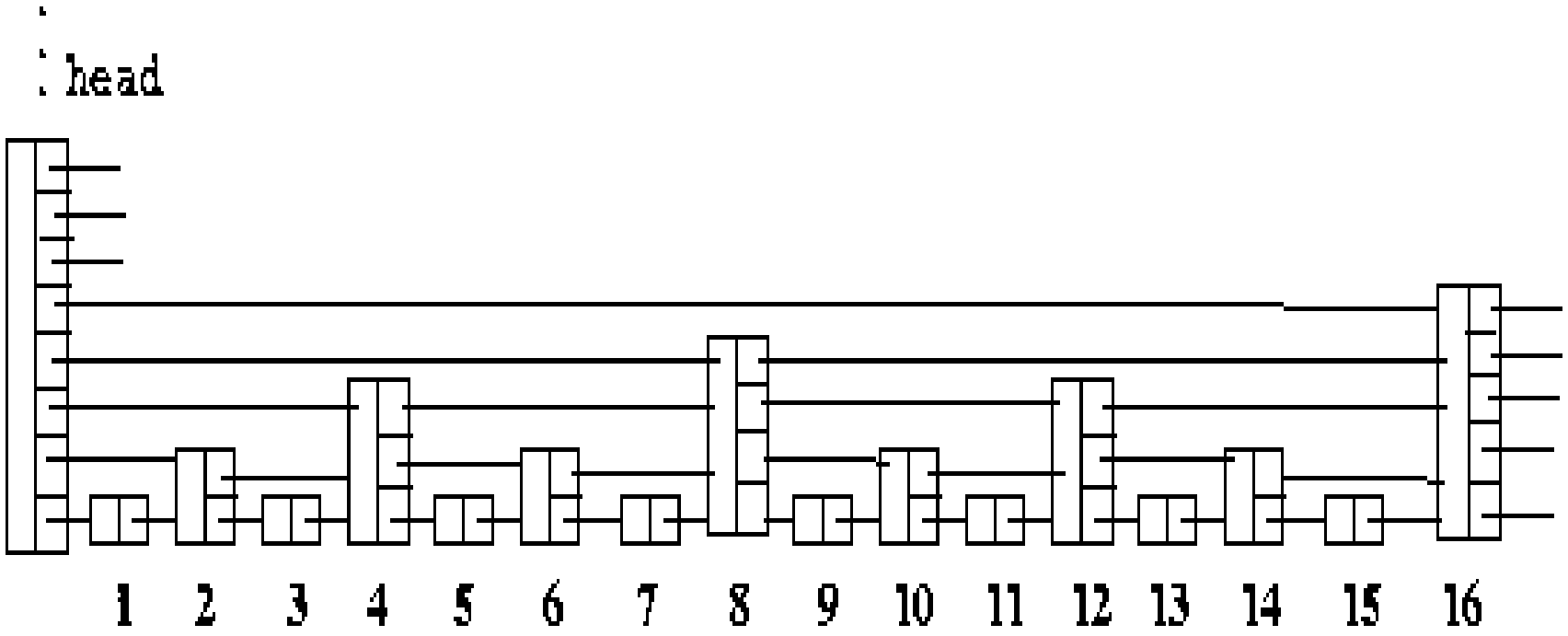
Pointers in A Perfect Skip List

- **Level-0** pointers:
  - ➔ **Original** pointers
- Level-1 pointers:
  - ➔ Pointers that connect every 2-nd nodes.
- Level-2 pointers:
  - ➔ Pointers that connect every 2^2-th nodes.
- Level-3 pointers:
  - ➔ Pointers that connect every 2^3-th nodes.
- **Level-i pointers:**
  - ➔ **Pointers that connect every 2^i-th nodes.**

Levels in A Perfect Skip List

- The level of a node is the number of pointers in the node.
  - ➔ **A level-i node has (i+1) pointers**!
- The header node points to the first node of each level!
- What is the **highest (maximum) level** of a skip list of N nodes?
  - ➔ The header points to the middle node and the middle node points to NULL.
  - ➔ **O(log N) levels**!

Example: A Perfect Skip List



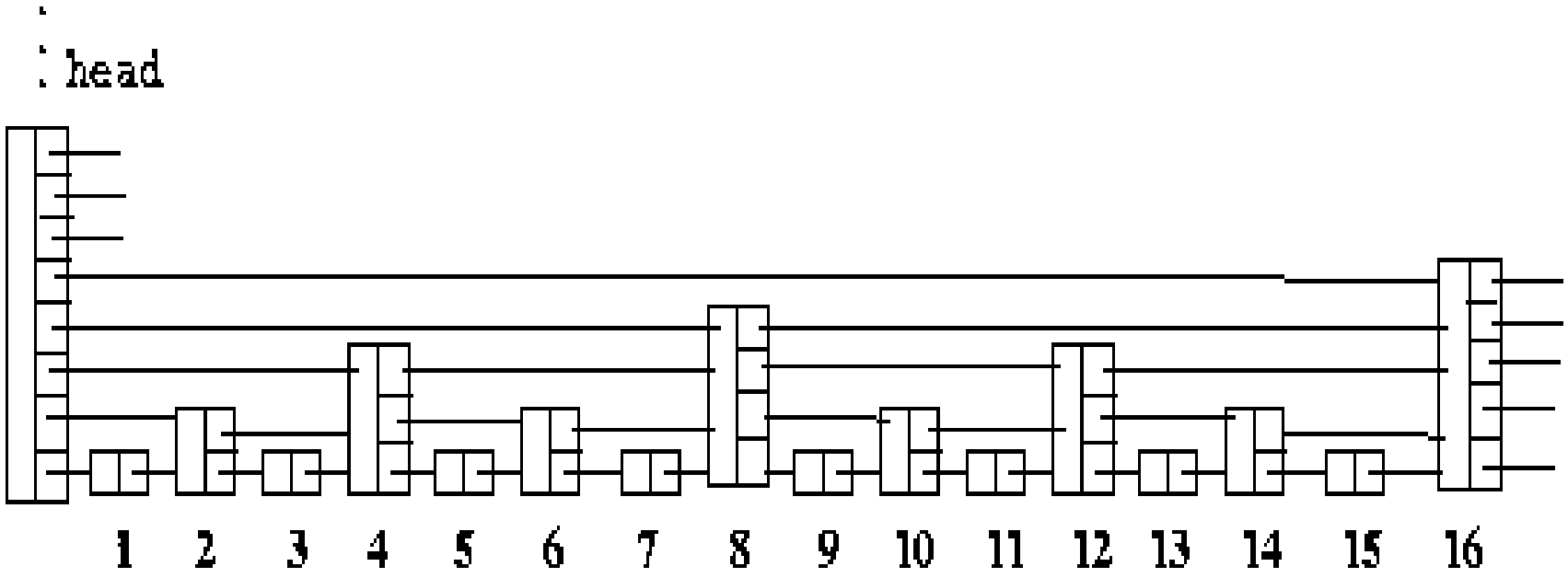
Quiz: A Perfect Skip List

- Construct by inserting 19, 8, 12, 5, 17, 9 and 7 into an empty perfect skip list.

Search in A Perfect Skip List

- A **non-sequential search** like a binary search is possible!!!
  - ➔Start at the highest link (level) at the header;
  - ➔Traverse along this level;
  - ➔ If the next node is larger or NULL then go to the next lower level and continue the strategy;
  - ➔Stop at level 0;
- The worst-case time for search is
  - ➔**O(log n) time**

Example: A Perfect Skip List Search



Insert & Delete with A Perfect Skip List

- The **insertion** and **deletion** operations can be very inefficient!
  - ➔Need restructuring!
  - ➔**O(n) time**
- This expense is too great, so maintaining a perfectly balanced condition is not worth it.

A Good Skip List

- We want a not-perfect-but-practically-good skip list?
  - ➔**Relax the structuring condition!**
- **Randomized Probabilistic Data Structure**
  - Makes some of its decision **at random**!

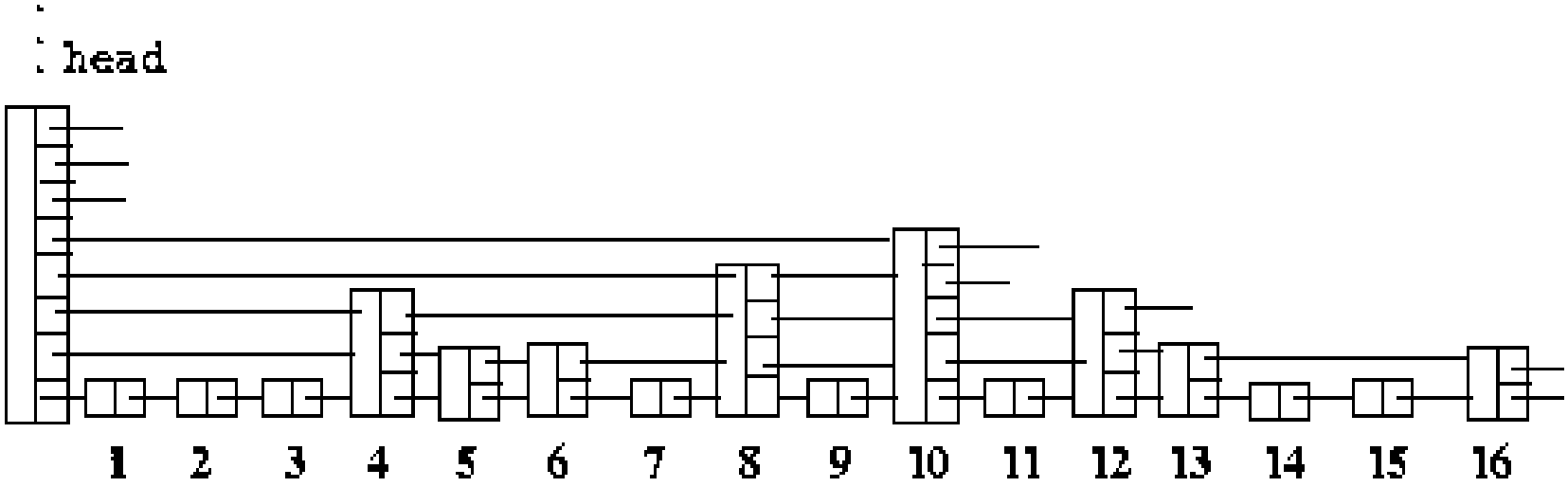
Observation

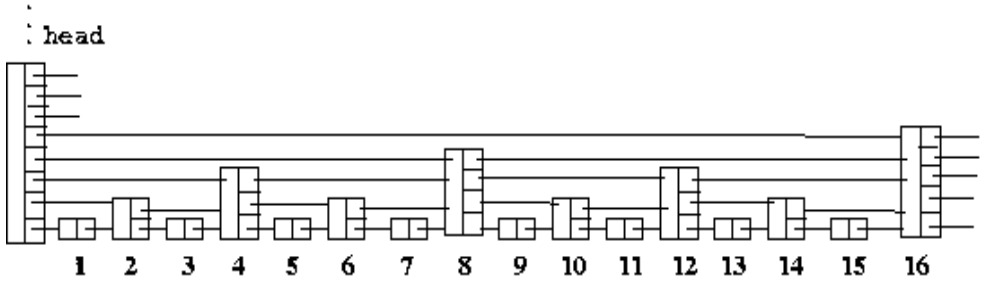
- **In a perfect skip list:**
  - **1/2 nodes at the level-0**
  - **1/4 nodes at the level-1**
  - **1/8 nodes at the level-2**
  - **1/16 nodes at the level-3**
  - **1/32 nodes at the level-4**
  - **And so on …**

Idea: A Good Skip List

- Idea?
  - ➔ Try to keep
    - 1/2 nodes at the level-0
    - 1/4 nodes at the level-1
    - 1/8 nodes at the level-2
    - 1/16 nodes at the level-3
    - 1/32 nodes at the level-4
    - And so on …
- How?
  - ➔ **Determine a level randomly with probability.**

Example: A Good Skip List

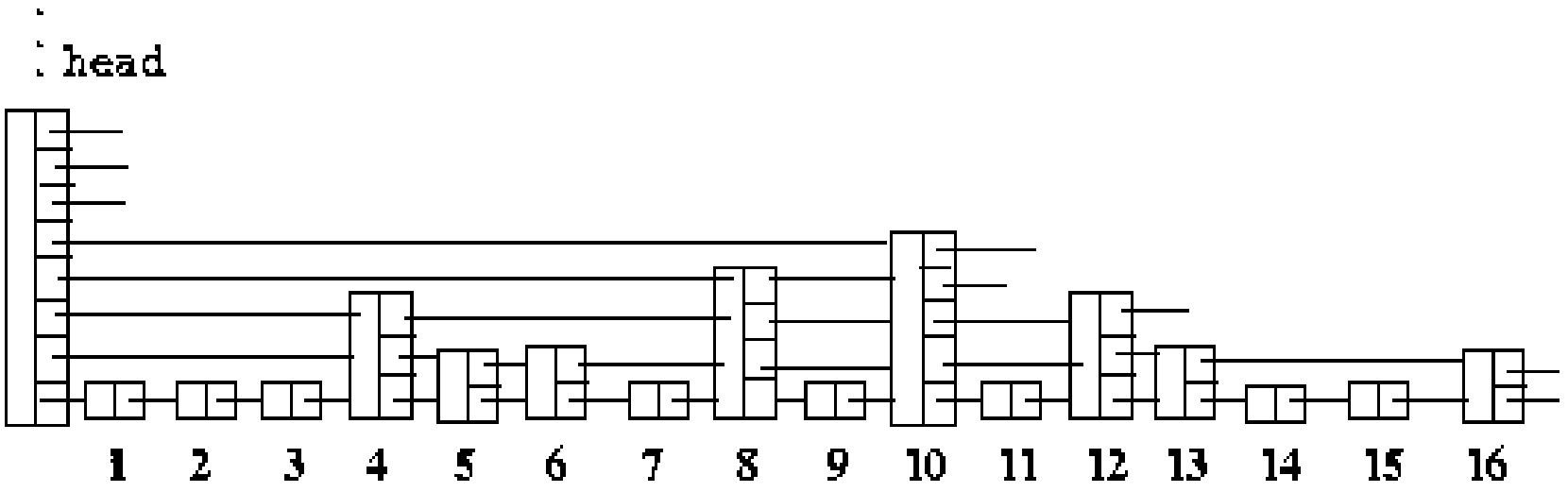




Quiz: A Probabilistic Skip List

- Construct by inserting 19, 8, 12, 5, 17, 9 and 7 into an empty probabilistic skip list.

Example: A Probabilistic Skip List Search



Search in A Good Skip List

- Not perfect anymore, but we can still skip large distances and so the performance is still good.
- **Worst case:**
  - ➔ **O(n) time**
- **Average case:**
  - ➔ **O(log n) time**

Insert & Delete in A Good Skip List

- Insert:
  - Find the insertion point using search;
  - Keep track of each pointer where we switch to a lower level;
  - **Determine the level randomly with probability for the new node;**
  - Splice the new node into the list;
- Delete:
  - Delete the new node from the list;

Insert & Delete in A Good Skip List

- Worst case:
  - ➔O(n) time
- **Average case:**
  - ➔**O(log n) time**

Self-Organizing (Linked) Lists

Self-Organizing Lists

- Problem:
  - ➔Linked lists – Linear search O(n)
- Goal:
  - ➔To improve the **average time** complexity of the search operation.

Self-Organizing Lists

- Another Idea:
  - By dynamically organizing (restructuring or adjusting) the list in a certain manner.
  - **Heuristics!**

Self-Organizing Methods

- **Move-to-front** method
  - Move it to the beginning of the list.
  - A B C D
  - D
  - D A B C

  - A B D
  - C
  - A B D C

Self-Organizing Methods

- **Transpose** method
  - Swap it with its predecessor.
  - A B C D
  - D
  - A B D C

  - A B D
  - C
  - A B D C

Self-Organizing Methods

- **Count** method
  - Order the list by the number of times accessed.
  - A3 B1 C1 D1
  - D
  - A3 D2 B1 C1
  - A3 B1 D1
  - C
  - A3 B1 D1 C1

Self-Organizing Methods

- **Ordering** method
  - Order the list using certain criteria.
  - A B C D
  - D
  - A B C D

  - A B D
  - C
  - A B C D

Example: Self-Organizing Lists

- Access A, C, B, C, D, A, D, A, C, A, C, C, E and E.
  - Plain
  - Move-to-front method
  - Transpose method
  - Count method
  - (Alphabetical) Ordering method

Example: Self-Organizing Lists

- Access A, C, B, C, D, A, D, A, C, A, C, C, E and E.
  - Plain ACBDE
  - Move-to-front method ECADB
  - Transpose method CADEB
  - Count method CAEDB
  - (Alphabetical) Ordering method ABCDE