Computational Complexity (Lower Bound) for The Searching Problem

Prof. Young Park

✓ Computational Complexity for The Searching Problem

- Lower Bounds for the Searching Problem Only by Comparison of Keys?

Computational Complexity for The Searching Problem

- Can we do better than O(n) search on an unsorted array?
- Can we do better than O(log n) search on a sorted array?

- Can we say that it is impossible to search faster than Ω(n) on an unsorted array in the worst case?
- Can we say that it is impossible to search faster than Ω(log n) on a sorted array in the worst case?
  - If we could, then this would be what it's called a lower bound.

Searching Only by Comparisons of Keys

- The problem of searching for a key:
  - Given an array *S* containing *n* keys and a key *x*, find an index *i* such that $x = S[i]$ if *x* equals one of the keys.
  - If *x* does not equal one of the keys, report failure.
- All algorithms that search only by comparison of keys.

Lower Bounds for Searching Only by Comparisons of Keys

1. The lower bound for searching on an **unsorted array** is **Ω(n)**.

2. The lower bound for searching on a **sorted array** is **Ω(log n)**.

1. Lower Bounds for Searching on an Unsorted Array Only by Comparisons of Keys

- Searching on an unsorted array?
  - **Ω(n)** comparisons at worst-case time
  - **Ω(n)** comparisons at average-case time

Lower Bounds for Searching on an Unsorted Array Only by Comparisons of Keys

- We prove a lower bound, **without going through all possible** searching algorithms?
- The lower bound for the problem of searching for K in an unsorted array of n elements is **n** comparisons.
  - **Proof by Contradiction**

2. Lower Bounds for Searching on a Sorted Array Only by Comparisons of Keys

- Searching on a sorted array?
  - **Ω(log n)** comparisons at worst-case time
  - **Ω(log n)** comparisons at average-case time

Lower Bounds for Searching on a Sorted Array Only by Comparisons of Keys

- How to prove a lower bound, **without going through all possible** searching algorithms?
- Idea?
  - The **Decision Tree** model
  - Similar to **the sorting lower bound**.

Decision Trees for Searching Algorithms on a Sorted Array

- We associate a **Decision Tree** with every deterministic algorithm that searches for a key x in an array of n keys.
  - Each leaf in a decision tree for searching n keys for a key x represent a point at which the algorithm stops and reports an index i such that $x = s_i$ or reports failure.
  - Every internal node represents a comparison.

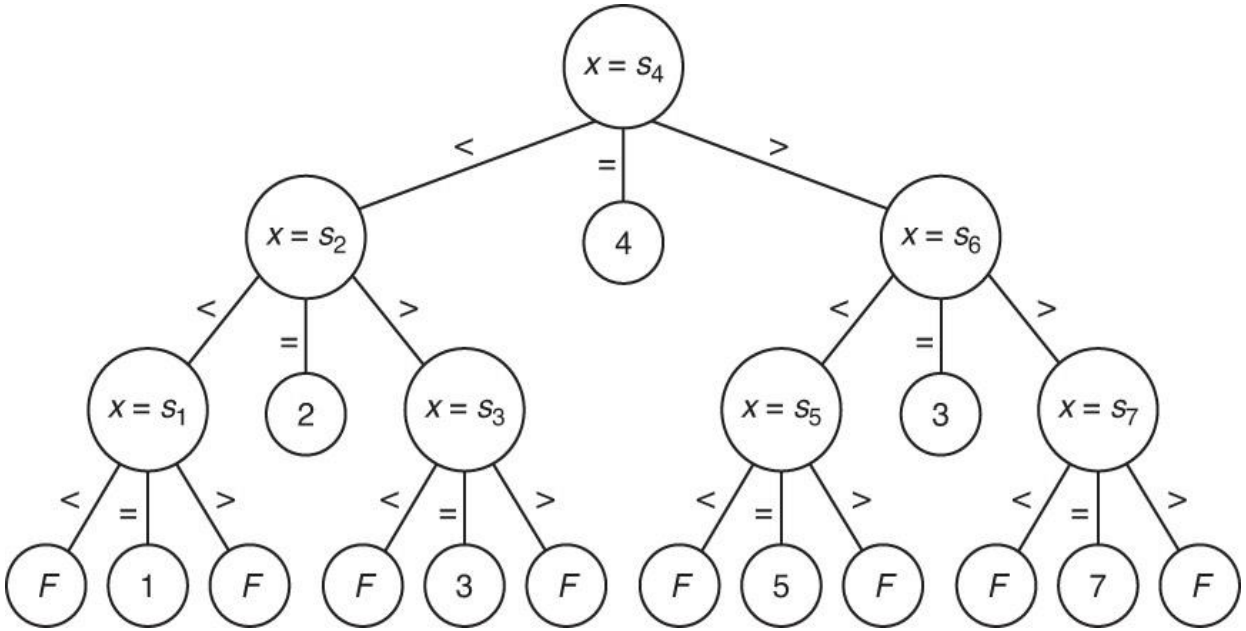
Search - Binary Search?

- **Algorithm 1.5 Binary Search – Iterative**
  - Divide-and-Conquer
  - Top-down
  - O(log N)

  - The decision tree of Binary Search with 7 keys?

Decision Tree for Binary Search with 7 Keys

Figure 8.1 **Binary Search**



The decision tree of Binary Search with 7 keys

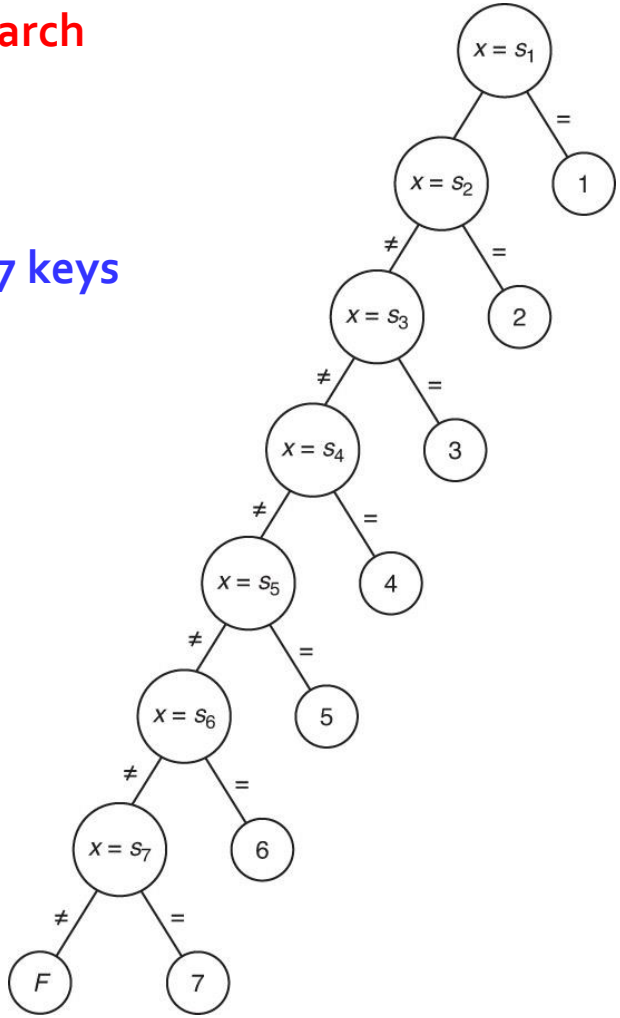
Search - Sequential Search?

- **Algorithm 1.1 Sequential Search**
  - Linear time O(N)
- The decision tree of Sequential Search with 7 keys?

Decision Tree for Sequential Search with 7 Keys

Figure 8.2 **Linear Search**

The decision tree of Sequential Search with 7 keys



Decision Trees for Searching Algorithms

- Every search algorithm that searches for a key *x* in a sorted array S of **n** *distinct* keys has a corresponding valid, pruned **decision tree.**

Lower Bounds for Worst-Case Behavior

- Each leaf is a position of x can be in S.
- Every leaf in a pruned, valid decision tree must be reachable.

Decision Trees for Searching Algorithms

- There must be **at least n+1 leaves** in the decision tree.
  - Because we have **n+1 distinct positions** of x can be in (any position in S, plus not in S at all).

Decision Trees for Searching Algorithms

- The **worst-case number of comparisons** of keys done by a decision tree is equal to its **height**.

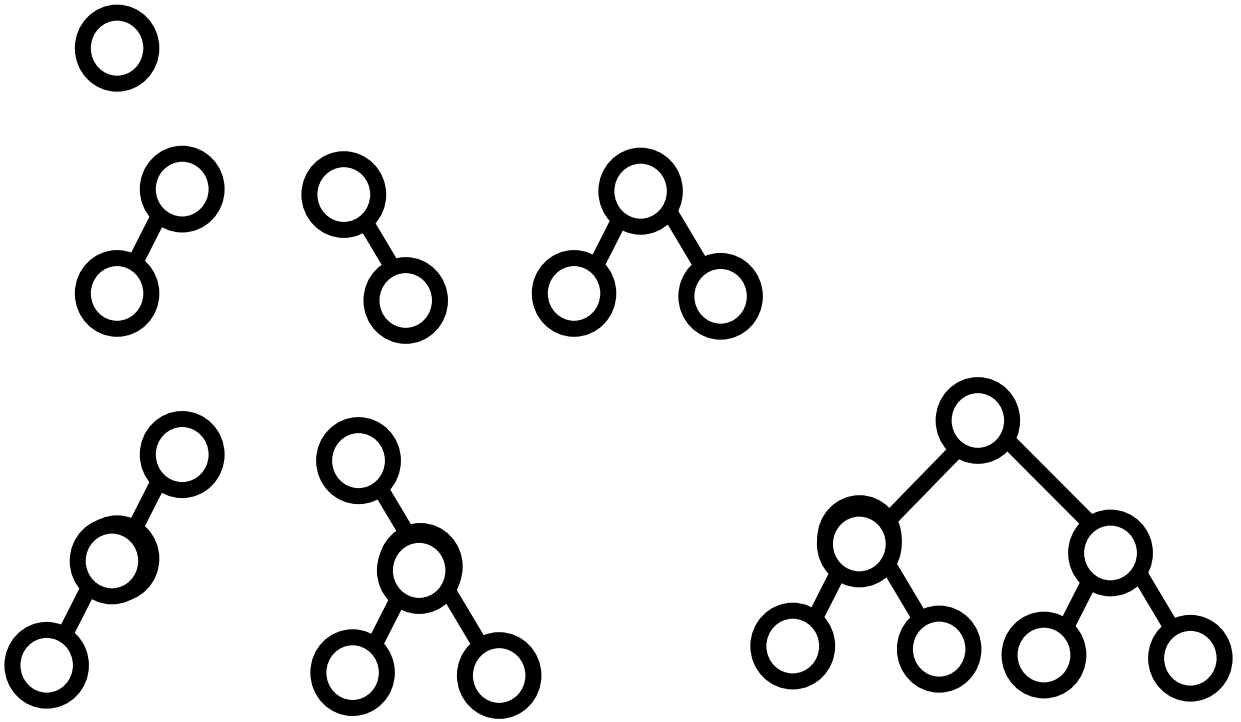
Decision Trees for Sorting Algorithms

- **h vs # of leaves** ➡ **# of leaves <= 2^h**

h=0

h=1

h=2



Decision Trees for Sorting Algorithms

- **Lemma 7.3:**
  - If **m** is the number of **leaves** in a binary tree and **h** is the **height**, then **h >= log m.**
    - height = h
    - **A binary tree with height h has at most 2^h leaves.**
    - **# of leaves <= 2^h**
    - 2^h >= # of leaves
    - **h >= log (#of leaves)**

Lower Bounds for Worst-Case Behavior

- At least log n comparisons needed.
- Theorem 8.1
  - Ω(log n) worst-case time

Lower Bounds for Average-Case Behavior

- **Theorem 8.2**
  - **Ω(log n) average-case time**

Lower Bounds for Searching on a Sorted Array Only by Comparisons of Keys

- Binary Search is very efficient for solving this problem when the array is sorted.
  - **O(log n)** worst-case time complexity

- As long as we limit ourselves to algorithms that search only by comparisons of keys, there is no improvement possible!

▶QUIZ?

- What is the lower bound for searching on a sorted array at worst-case only by comparisons of Keys?

✓ Algorithms for The Searching Problem

- Algorithms that **search only by comparison of keys**.

Static Searching vs Dynamic Searching

- **Static Searching** is a process in which the records are all added to the file at one time, and there is no need to add or delete records later.
- Many applications require **Dynamic Searching**, which means that records are added and deleted frequently.

Dynamic Searching – Arrays?

- An array structure is inappropriate for dynamic searching.
  - Because, when we add a record in sequence to a sorted array, we must move all the records following the added record.

Dynamic Searching – Binary Search

- Binary Search requires that the keys be structured in an array, because there must be an efficient way to find the middle item.
- Binary Search cannot be used for dynamic searching!
- Worst-case:
  - **O(log n) Search**
  - **O(n) Insert & Delete**

Dynamic Searching – Linked Lists?

- Although we can readily add and delete records using a linked list, there is no efficient way to search a linked list.
- Worst-case:
  - **O(n) Search**
  - **O(1) – O(n) Insert & Delete**

Dynamic Searching – Trees!

- **Dynamic searching** can be implemented efficiently using a **tree structure!**

Binary Search Trees

- **Binary search trees** are appropriate for dynamic searching.
  - **All data items are kept in sorted order!**

Binary Search Trees

- When keys are dynamically added and deleted, there is no guarantee that the resulting tree will be balanced.
  - Often obtains a skewed tree, which is simply a linked list.
  - If the keys are added at random, the resulting tree will be closer to a balanced tree than it will be closer to a linked list.

Binary Search Trees

- Worst-case:
  - O(n) Search
  - O(n) Insert & Delete
- **Average-case:**
  - **O(log n) Search**
  - **O(log n) Insert & Delete**

Height-Balanced Search Trees

- In a very dynamic environment, it would be better if the tree never became unbalanced in the first place.
- A **height balanced** search tree with some balancing condition.
  - **All data items are kept in sorted order!**
  - **Height is balanced!**

Height-Balanced Search Trees

- AVL trees
- 2-3 trees
- 2-3-4 trees
- B-trees
- Red-Black trees
- …

Height-Balanced Search Trees

- The search, addition, and deletion times are guaranteed to be Θ (log n).
- **Worst-case:**
  - **O(log n) Search**
  - **O(log n) Insert & Delete**

✓ Computational Complexity for The Searching Problem Summary

- Searching on a sorted array?
  - **Ω(log n)** comparisons at worst-case time
  - **Ω(log n)** comparisons at average-case time
- Searching on an unsorted array?
  - **Ω(n)** comparisons at worst-case time
  - **Ω(n)** comparisons at average-case time

Homework Assignment

Homework Assignment?

- What is the lower bound for searching on a sorted array only by comparison of keys? Explain.

✓ Textbook Readings

- Chapter 8:
  - 8.1 (8.1.1 only)
  - 8.3

the right majors, minors &

ucation?

or students' academic and career success

ging for many students - *Many students change their during college!*

e prediction of student success in MMC could dual students

d their right MMC

chieve their academic goals

END

Prof. Young Park