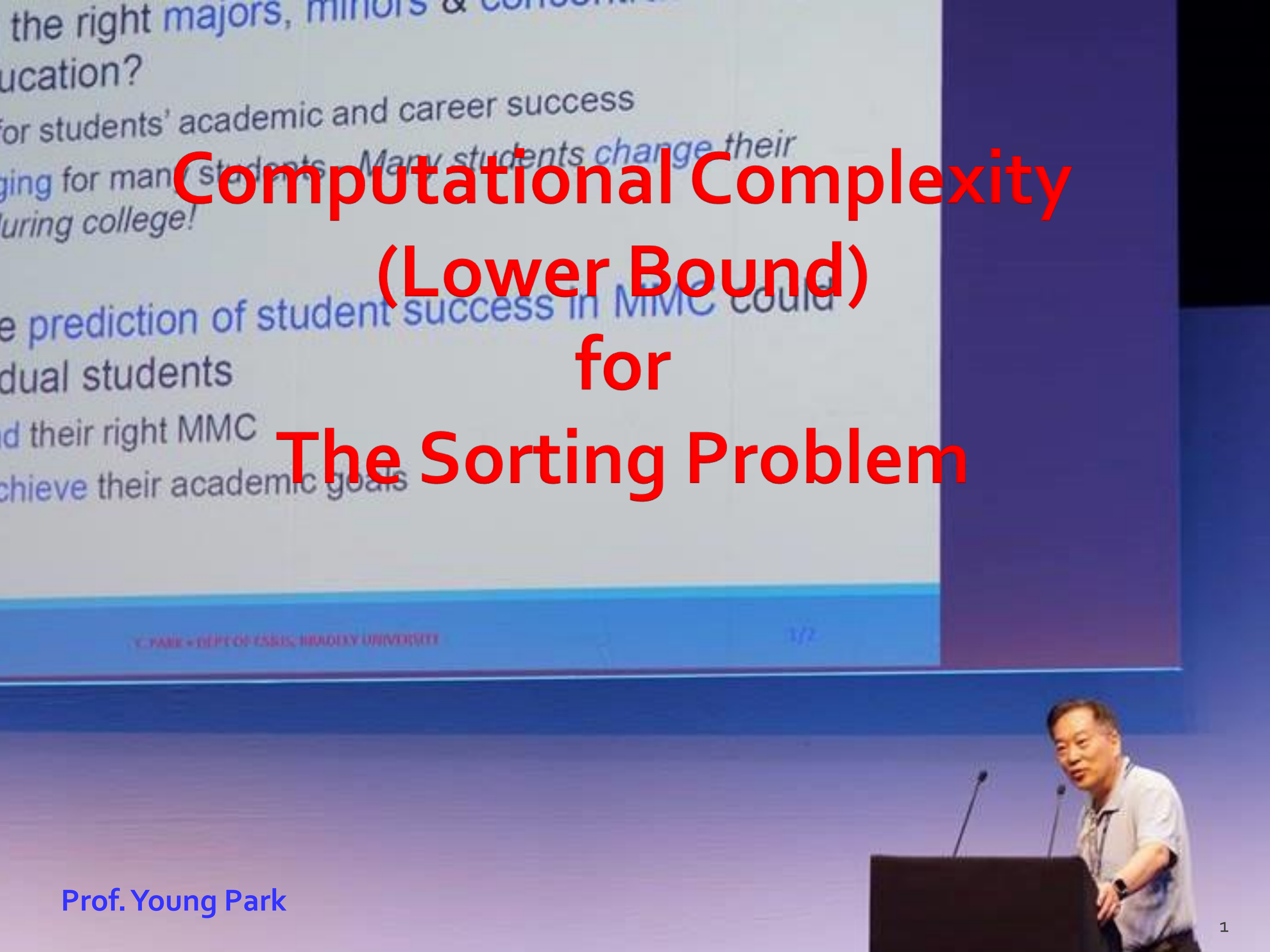


Computational Complexity (Lower Bound) for The Sorting Problem

Prof. Young Park

Two Approaches to Solving Problems

- Try to develop **a more efficient algorithm** for the problem using algorithm design methodologies.
- Try to determine a lower bound on the efficiency of **all algorithms** for the problem
  - Thus prove that a more efficient algorithm is not possible.
  - Then, quit trying to obtain a faster algorithm.

✓ Computational Complexity

Computational Complexity

- The study of **all possible algorithms** that can solve **a given problem**.
- **Computational complexity is a property of a problem!**
  - It is **not** a property of **any one algorithm** for that problem.

Lower Bound

- A **lower bound** for a problem is the worst-case running time of the best possible algorithm for that problem.

Lower Bound

- Determines a **lower bound** on the efficiency of all algorithms **for a given problem**.
  - This does not mean that it must be possible to create an algorithm with the time complexity for the problem.
  - It means only that is impossible to create one that is better than the time complexity.

Lower Bound

- The Matrix Multiplication Problem:
  - Computational complexity analysis has determined a lower bound is $\Omega(n^2)$.
    - Does not mean it is possible to create an algorithm $\Theta(n^2)$.
    - It means it is impossible to create an algorithm better than $\Theta(n^2)$
    - Best algorithm to date: $\Theta(n^{2.38})$.

Computational Complexity and Algorithm Design

- **For a given problem**, we
  - Try to determine a **lower bound of Ω(f (n)).**
  - Try to develop a **Θ(f (n)) algorithm** for the problem.
  - Once we have done this, we know that, except for improving the constant, we cannot improve on the algorithm any further.

✓ Computational Complexity for The Sorting Problem

1. Algorithms for The Sorting Problem
2. Lower Bounds of The Sorting Problem

1. Algorithms for The Sorting Problem

- Algorithms that sort only by comparison of keys.

Sorting Only by Comparisons of Keys

- All algorithms that sort only by comparison of keys.
  - Can compare two keys to determine which is larger, and can copy keys, but can do no other operations on them.
- We analyze the algorithms in terms of
  - The number of **comparisons** of keys and
  - The number of **assignments** of records.
  - How much extra space the algorithms require besides the space needed to store the input.

Basic Quadratic $O(n^2)$ Sorting Algorithms

- **Quadratic $O(n^2)$** Sorting Algorithms
  - **Exchange Sort**
  - **Bubble Sort**
  - **Selection Sort**
  - **Insertion Sort**
  - **…**

Exchange Sort

- Sorting by Exchanging
- Algorithm 1.3 Exchange Sort
  - Idea: compare the first element with each following element and exchange!

- Analysis of Algorithm 1.3 Exchange Sort
  - **$O(n^2)$** comparisons worst-case

▶QUIZ?

- S = [84, 69, 76, 86, 94, 91]

- [94, 69, 76, 84, 86, 91]
- [94, 91, 69, 76, 84, 86]
- [94, 91, 86, 69, 76, 84]
- [94, 91, 86, 84, 69, 76]
- [94, 91, 86, 84, 76, 69]

Bubble Sort

- Sorting by Exchanging
  - Idea: compare two adjacent elements and exchange!
- Analysis of Algorithm Bubble Sort
  - **$O(n^2)$** comparisons worst-case

▶ QUIZ?

- S = [84, 69, 76, 86, 94, 91]

- [84, 76, 86, 94, 91, 69]
- [84, 86, 94, 91, 76, 69]
- [86, 94, 91, 84, 76, 69]
- [94, 91, 86, 84, 76, 69]
- [94, 91, 86, 84, 76, 69]

Selection Sort

- Sorting by Selection
- Algorithm 7.2 Selection Sort
  - Idea: Select the smallest/largest and put it in the right position!

- Analysis of Algorithm 7.2 Selection Sort
  - **$O(n^2)$** comparisons worst-case

▶QUIZ?

- S = [84, 69, 76, 86, 94, 91]

- [94, 69, 76, 86, 84, 91]
- [94, 91, 76, 86, 84, 69]
- [94, 91, 76, 86, 84, 69]
- [94, 91, 86, 76, 84, 69]
- [94, 91, 86, 84, 76, 69]
- [94, 91, 86, 84, 76, 69]

Insertion Sort

- Sorting by Insertion
- Algorithm 7.1 Insertion Sort
  - Idea: Pick one element and insert it in the right position!

- Analysis of Algorithm 7.1 Insertion Sort
  - **$O(n^2)$** comparisons worst-case

▶ QUIZ?

- S = [84, 69, 76, 86, 94, 91]

- [84, 69, 76, 86, 94, 91]
- [84, 69, 76, 86, 94, 91]
- [84, 76, 69, 86, 94, 91]
- [86, 84, 76, 69, 94, 91]
- [94, 86, 84, 76, 69, 91]
- [94, 91, 86, 84, 76, 69]

Exchange Sort, Selection Sort & Insertion Sort

- **Table 7.1**
- In practice, none of these algorithms is practical for extremely large instances
  - Because all of them are **quadratic-time** in both the average case and the worst case!

▶ QUIZ?

- Basic Sorting Algorithms Time Complexity: *Best? Average? Worst?*
  - **Exchange Sort**
  - **Bubble Sort**
  - **Selection Sort**
  - **Insertion Sort**

O(n log n) Sorting Algorithms

- **O(n log n)** Sorting Algorithms
  - **Merge Sort**
    - Sorting by Merging
  - **Quick Sort**
    - Sorting by Partitioning
  - **Heap Sort**
    - Sorting by Selection

Merge Sort

- Algorithm 2.4 Merge Sort
  - Worst-Case Time Complexity= O(nlogn)
  - Average-Case Time Complexity = O(nlogn)

$$T(n) = \begin{cases} O(1) & \text{if } n=1 \\ 2\,T(n/2) + O(n) & \text{if } n>1 \end{cases}$$

  - Worst-Case **Space** Complexity= O(n)

QUIZ?

- **MergeSort** with no extra space?
  - Merge with **no extra space**?
    - **$O(n^2)$**
  - Recurrence Equation?
    - $T(n) = 2\,T(n/2) + O(n^2)$
    - **$O(n^2)$**

Quicksort (Partition Exchange Sort)

- Algorithm 2.6 Quicksort
  - **$O(n^2)$** comparisons worst-case

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = T(n-1) + O(n) \quad \text{if } n>1$$

Quicksort (Partition Exchange Sort)

- Algorithm 2.6 Quicksort
  - **O(n log n)** comparisons average-case

$$T(n) = O(1) \quad \text{if } n=1$$

$$T(n) = \sum_{p=1,n} [T(p-1) + T(n-p)]\, 1/n + O(n) \quad \text{if } n>1,\ 1 =< p =< n$$

▶ QUIZ?

- **Suppose we have a O(n) time algorithm that finds median of an unsorted array.** Now consider a QuickSort implementation where we first find median using the above algorithm, then use median as pivot. What will be the **worst case time complexity** of this modified **QuickSort**?

  - **O(n log n)**

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = 2T(n/2) + O(n) \quad \text{if } n>1$$

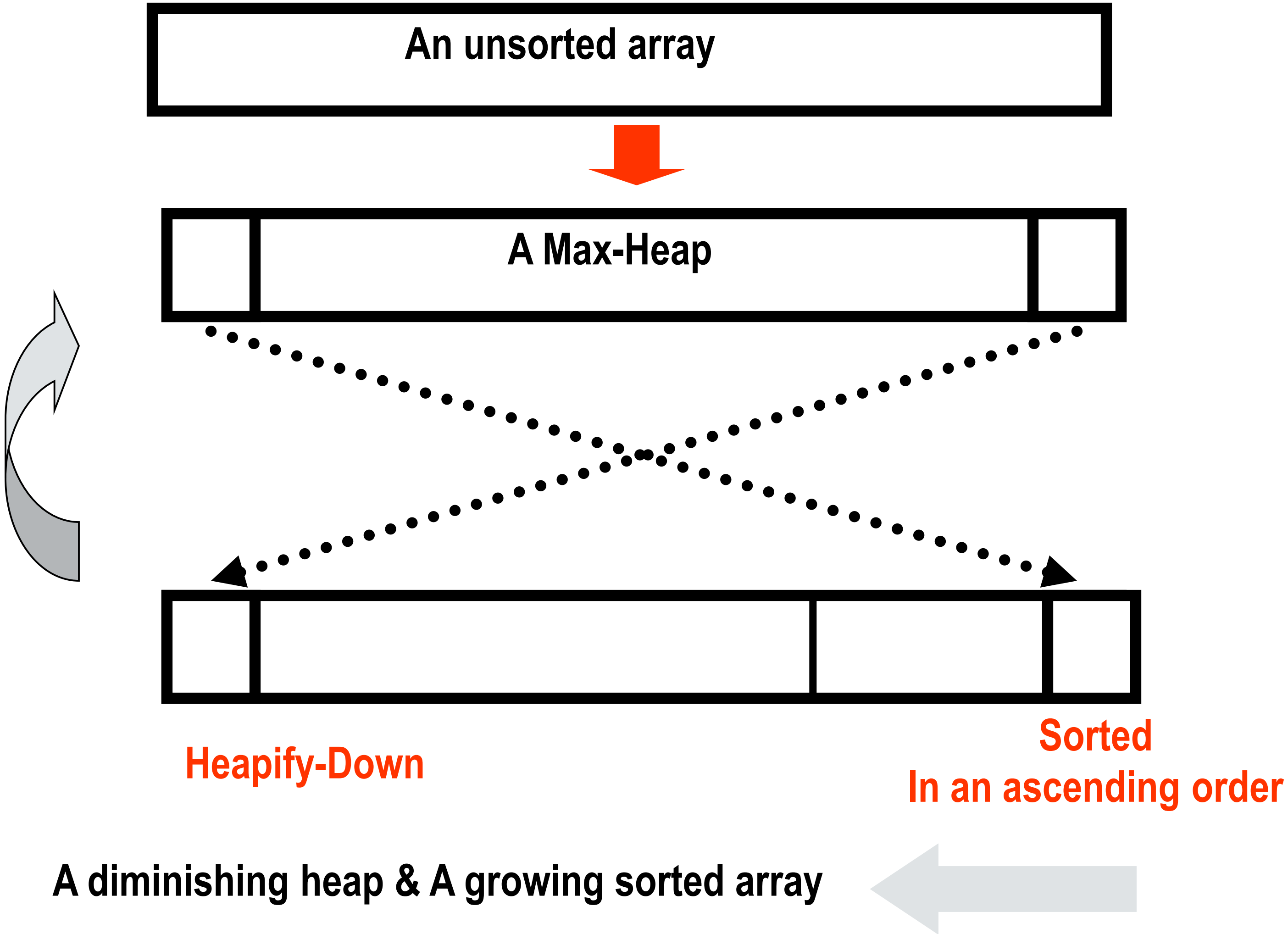
Heap Sort

- **Heap Sort** = Sorting by Selection
- Using a data structure called a heap!
- Heap Sort is an in-place **O(n log n)** algorithm!

Heap Sort

- **Convert an array of unsorted data into a max-heap.**
- Take the root element from the heap and put it into its place.
- Re-heap the remaining elements
  - via the reheapfication downward, i.e. **Heapify-Down**!
- Repeat until all elements are in the correct positions.

Heap Sort



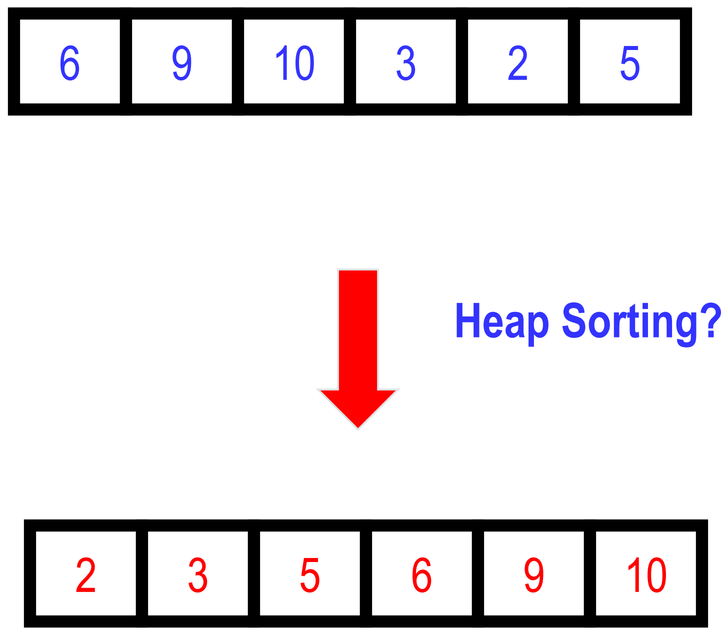
An unsorted array
A Max-Heap
Heapify-Down
Sorted
In an ascending order
A diminishing heap & A growing sorted array

RECALL: Time Complexity for Converting (Building) a Binary Heap

- **Approach 1:**
  - Using Heapify-Up
  - **Top-down heap building**
  - Worst-case running time **O(N log N)**

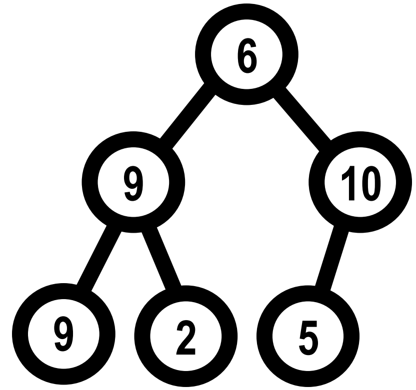
- **Approach 2:**
  - Using Heapify-Down
  - **Bottom-up heap building**
  - Worst-case running time **O(N)**

Example: Heap Sort

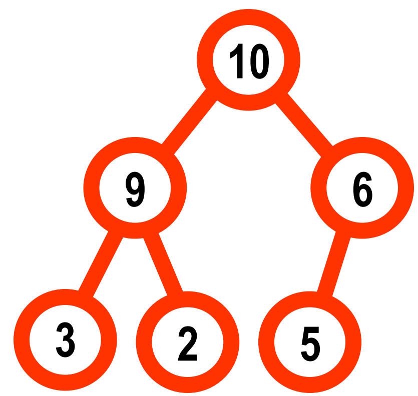


Example: Transform an Array into a Heap

6	9	10	3	2	5



10	9	6	3	2	5



Max-Heap

Example: Heap Sort

Max-Heap

10	9	6	3	2	5

↓

5	9	6	3	2	10

↓

9	5	6	3	2	10

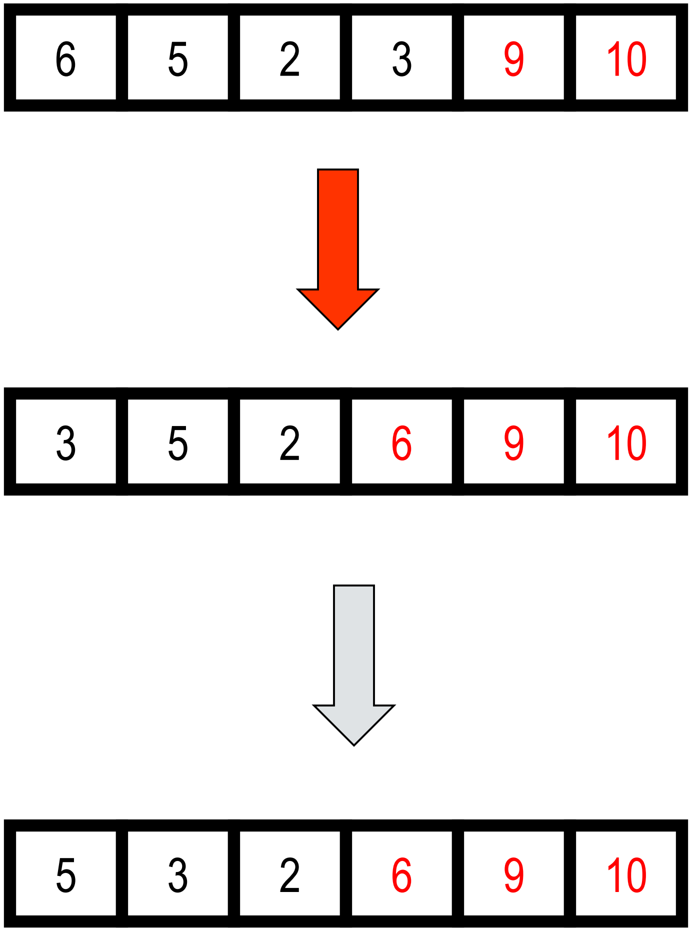
Example: Heap Sort

9	5	6	3	2	10

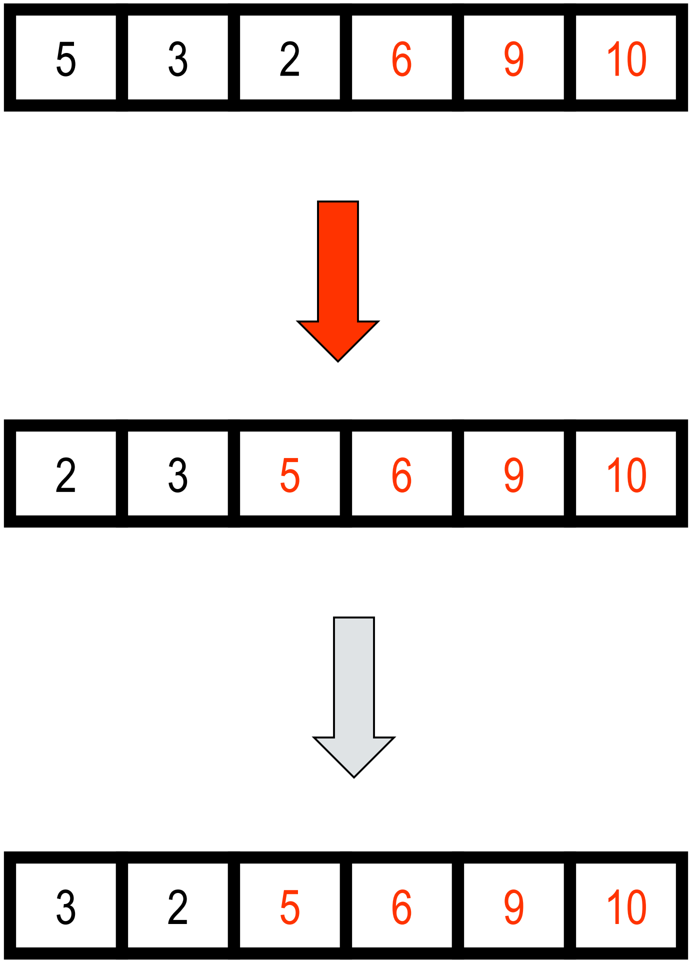
2	5	6	3	9	10

6	5	2	3	9	10

Example: Heap Sort



Example: Heap Sort



Example: Heap Sort

3	2	5	6	9	10

↓

2	3	5	6	9	10

↓

2	3	5	6	9	10

Example: Heap Sort

2	3	5	6	9	10

↓

2	**3**	**5**	**6**	**9**	**10**

Sorted!

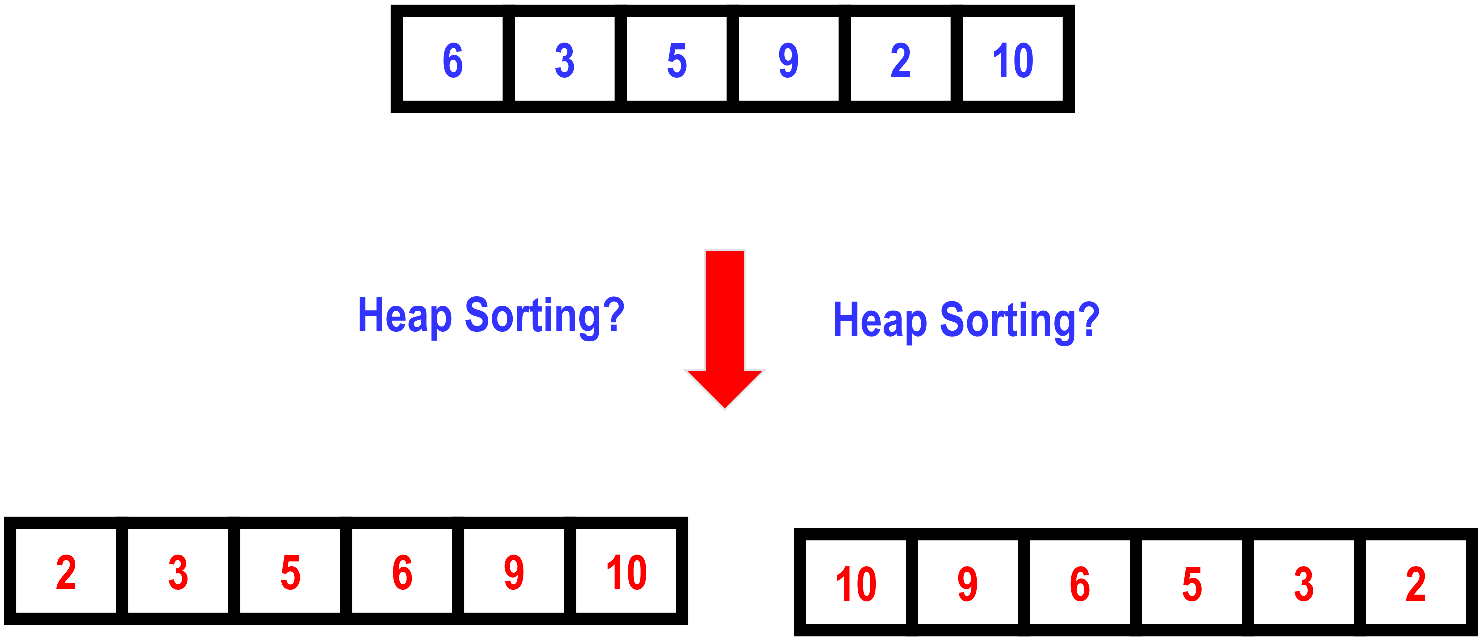
▶ QUIZ? Heap Sort

6	9	10	3	2	5

↓ Heap Sorting?

10	9	6	5	3	2

QUIZ? Heap Sort



Heap Sort

- **Algorithm 7.5** Heapsort
- Analysis of Algorithm 7.5 Heapsort
  - Worst-Case Time Complexity= (**n logn** or **n**) + (**n log n**)
    - **O(n log n)** comparisons worst-case
  - Extra Space Usage
    - **O(1)** extra space worst-case, i.e., in-place

▶ QUIZ?

- Describe the **heap sort** algorithm?
- Worst-case running time of **heap sort**?

▶ QUIZ: Ternary Heap Sort?

- **HeapSort** using **Ternary** Heap?
- Exercise 45

▶ QUIZ: Tree Sort?

- **Tree Sort** - Sorting by Selection using a **Balanced Binary Search Tree**?

Heap Sort Visualization

- *Heap Sort Visualization*

Comparison-Based Sorting Visualization

- *Comparison-Based Sorting Visualization*

2. Lower Bounds of The Sorting Problem

- Lower bound for sorting only by comparison of keys.

Lower Bounds for Sorting Only by Comparison of Keys

- Can we develop sorting algorithms whose time complexities are of an even **better order than O(n log n)**?

  - As long as we limit ourselves to sorting only by comparisons of keys, such algorithms are **not possible!**

Lower Bounds for Sorting Only by Comparison of Keys

- **$\Omega(n \log n)$** comparisons worst-case
- **$\Omega(n \log n)$** comparisons average-case

- Idea?

Lower Bounds for Sorting Only by Comparison of Keys

- To prove a lower bound of $\Omega(n \log n)$ for sorting,
  - We have to prove that no sorting algorithm could possibly be faster than $n \log n$.

Lower Bounds for Sorting Only by Comparison of Keys

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

Decision Trees for Sorting Algorithms

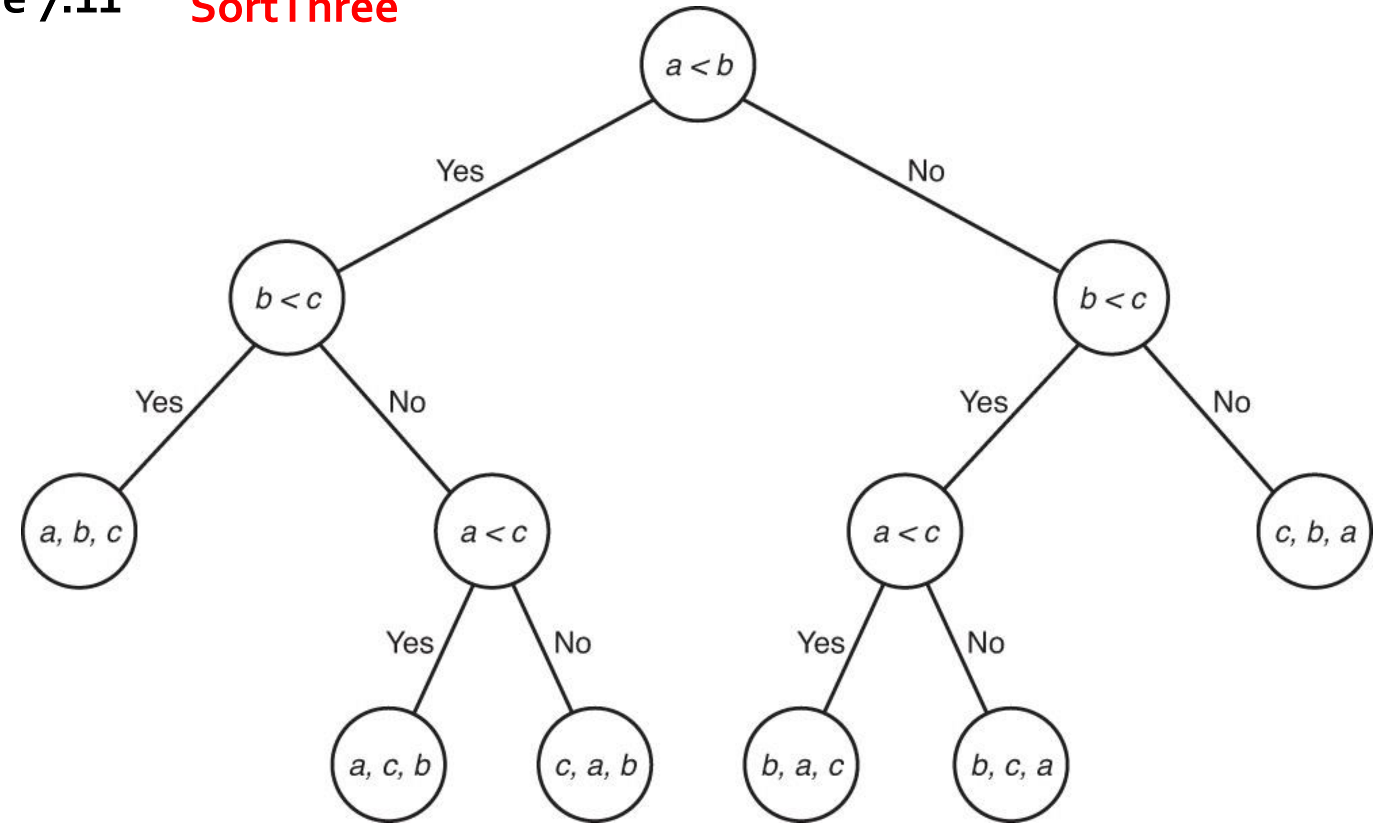
- A **Decision Tree** constructed in such a way that a decision must be made at each node about which node to visit next.
  - valid for sorting *n* keys if, for each permutation of the *n* keys, there is a path from the root to a leaf that sorts that permutation.
  - pruned if every leaf can be reached from the root by making a consistent sequence of decisions.

Sorting 3 Keys

```
void SortThree (keytype S[]) //S indexed from 1 to 3
{
  keytype a, b, c;
  a = S[1]; b = S[2]; c = S[3];
  if (a < b)
    if (b < c)
          S = a, b, c; //means S[1]=a; S[2]=c; S[3]=c;
    else if (a < c)
          S = a, c, b;
    else
          S = c, a, b;
  else if (b < c)
          if (a < c)
          S = b, a, c;
          else
          S = b, c, a;
      else
          S = c, b, a;
}
```

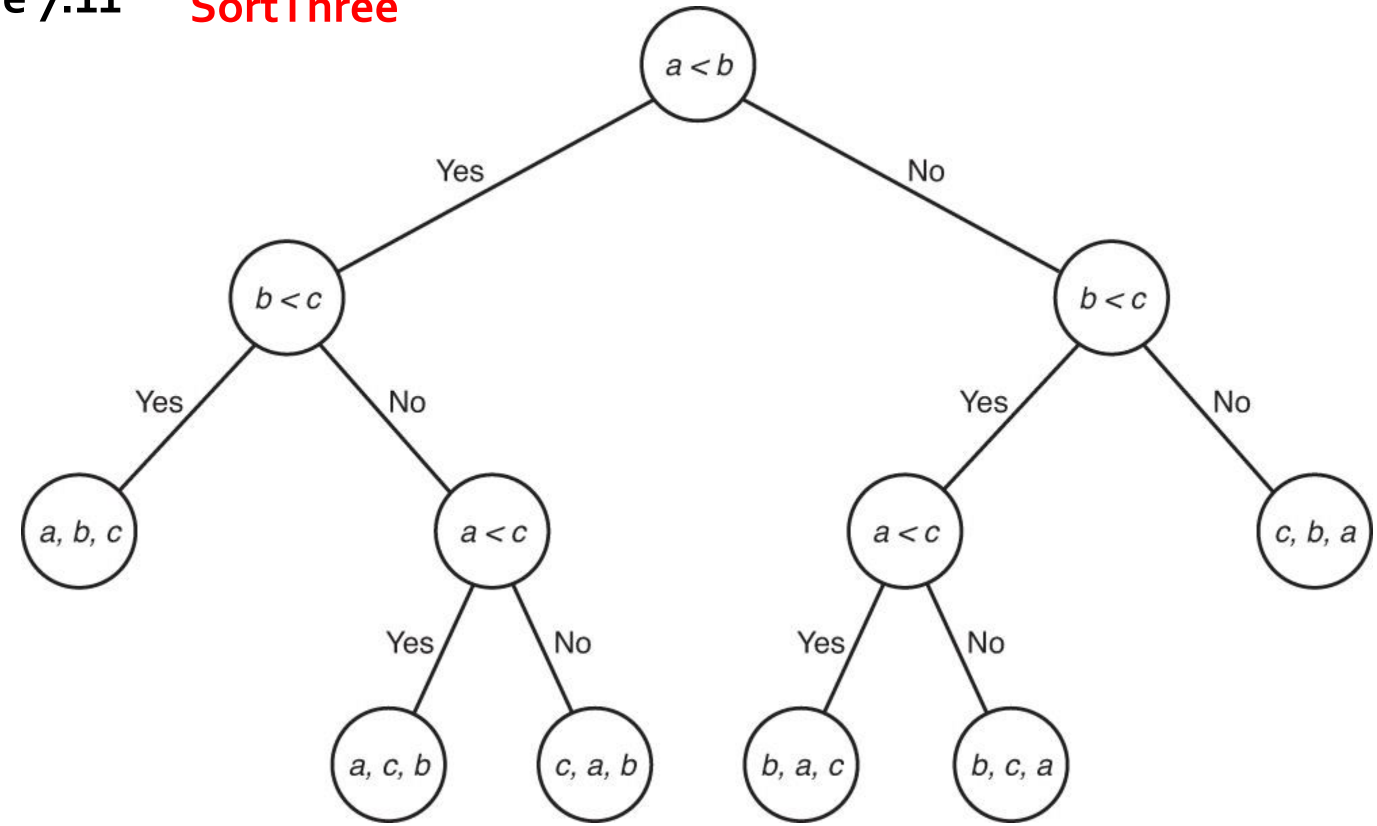
Decision Tree for Sorting 3 Keys

Figure 7.11 SortThree



Decision Tree for Sorting 3 Keys

Figure 7.11 SortThree



Total $3! = 6$ Permutations (Leaves)

Exchange Sort

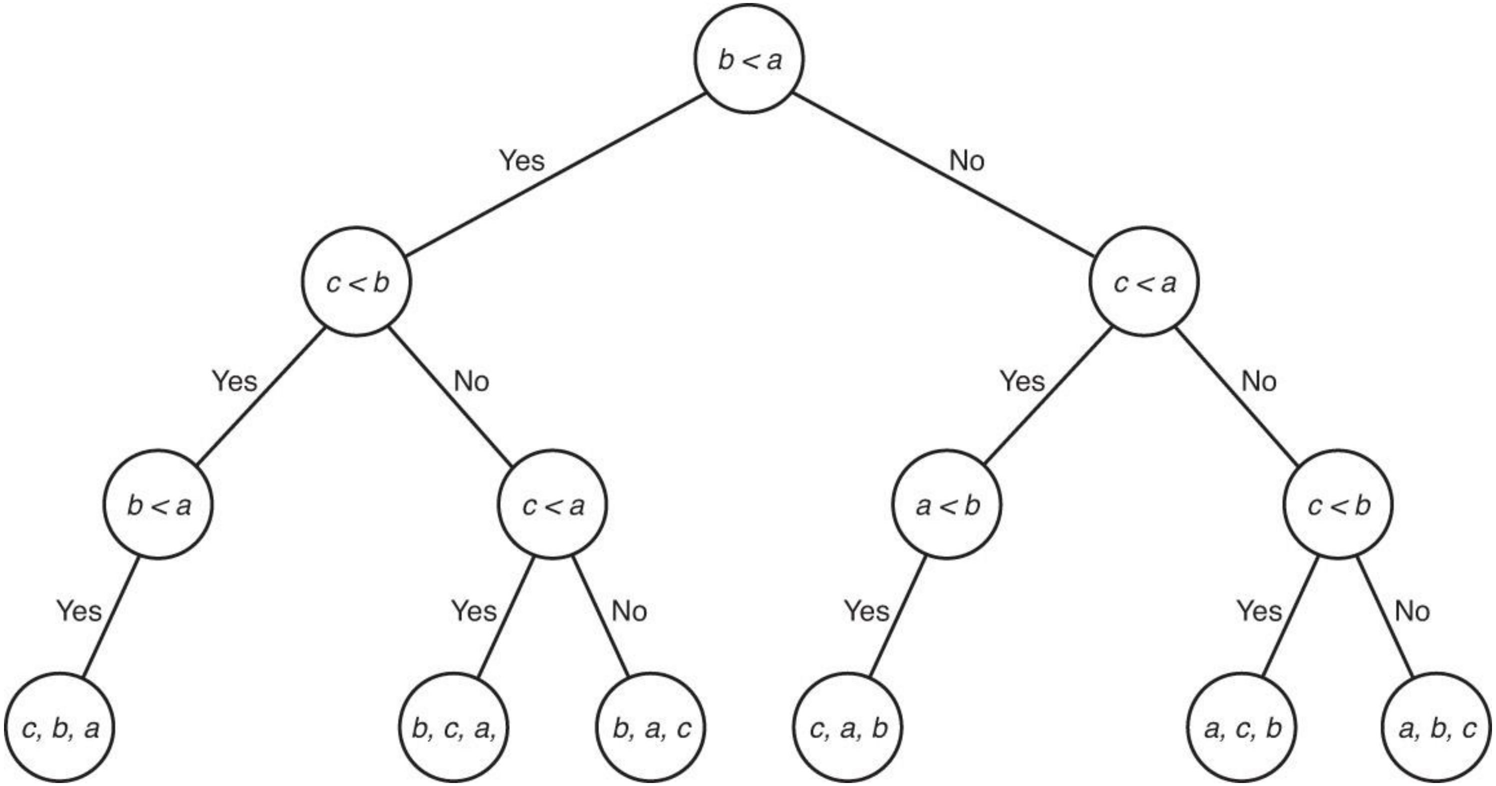
- Sorting by Exchanging
- **Algorithm 1.3** ExchangeSort
  - Idea: compare the first element with each following element and exchange!

Exchange Sorting 3 Keys

- S = [a, b, c]
  = [84, 69, 76]
- [84, 69, 76]
  - b<a, Yes bac
- [69, 84, 76]
  - c<b No bac
- [69, 84, 76]
  - c<a Yes bca
- [69, 76, 84] = [b, c, a]

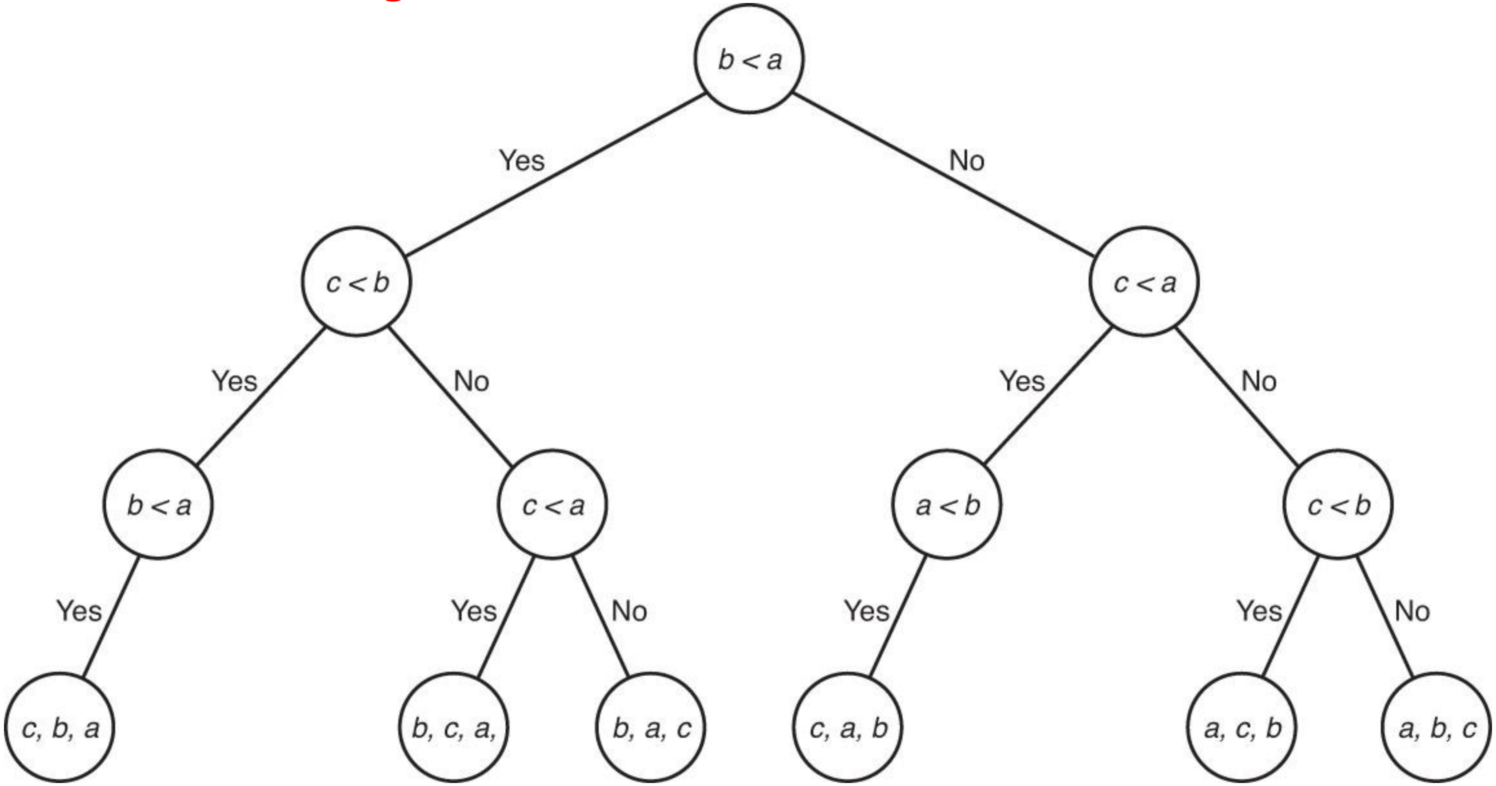
Decision Tree for Exchange Sorting 3 Keys

Figure 7.12 **ExchangeSort**



Decision Tree for Exchange Sorting 3 Keys

Figure 7.12 **ExchangeSort**



Total 3! = 6 Permutations (Leaves)

Decision Trees for Sorting Algorithms

- **Lemma 7.1**:
  - To **every** **deterministic algorithm** for sorting **n distinct keys**, there corresponds a pruned, valid binary decision tree containing **exactly** **n!** **leaves**.

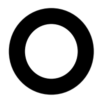
Decision Trees for Sorting Algorithms

- **Lemma 7.2**:
  - The **worst-case** **number of comparisons** of keys done by a decision tree is equal to its **height**.

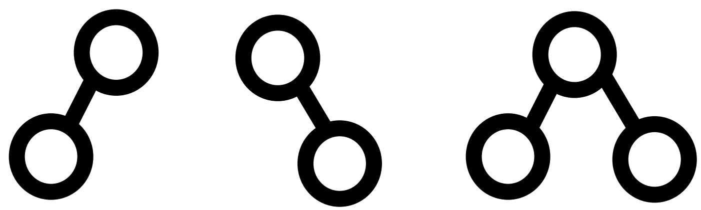
Decision Trees for Sorting Algorithms

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

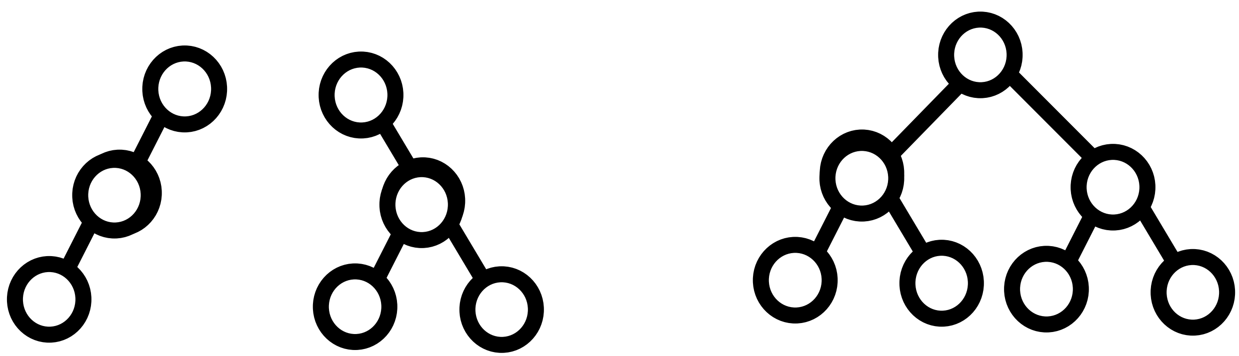
h=0



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

- **Theorem 7.2**:
  - **At least log n! comparisons**

Lower Bounds for Worst-Case Behavior

- **Lemma 7.4**
  - **$\log n! = \Omega(n \log n)$**

- **$\log (n!) =$** $\log (n (n-1)(n-2) \ldots 2)$

  $= \log n + \log (n-1) + \log (n-2) + \ldots + \log 2$

  $\geq (n/2) \log n/2$

  $= n/2 (\log n - \log 2)$

  $= (n/2) \log n - n/2$

  $\geq \frac{1}{4} n \log n$

Lower Bounds for Worst-Case Behavior

- **Theorem 7.3 Sorting Lower bound**
  - **Ω(n log n)** comparisons worst-case

Lower Bound for Average Behavior

- We assume that all possible permutations are equally likely to be the input.
- The external path length (EPL) of a decision tree is the total number of comparisons done by the decision tree to sort all possible inputs.
- The average number of comparisons done by a decision tree for sorting n distinct keys is EPL/n!.

Lower Bound for Average Behavior

- **Theorem 7.4**
  - **Ω(n log n)** comparisons average-case

▶ QUIZ?

- What is the worst-case Lower Bound for Sorting Only by Comparison of Keys? Explain.

✓ Lower Bounds of The Sorting Problem Summary

- **Ω(n log n)** comparisons **worst**-case
- **Ω(n log n)** comparisons **average**-case
  - Mergesort, Quicksort & Heapsort

Homework Assignment

▶ Homework Assignment?

- What is the worst-case time complexity of HeapSort using ternay heap? Explain.
- What is the lower bound for sorting only by comparison of keys? Explain.

✓ Textbook Readings

- Chapter 7:
  - 7.1
  - 7.2
  - 7.3
  - 7.4
  - 7.5
  - 7.6
  - 7.7
  - 7.8 (7.8.1 & 7.8.2 only)

END

Prof. Young Park