CS3230 – Design and Analysis of Algorithms (S1 AY2024/25)

Lecture 4a: Lower Bound for Comparison-Based Sorting

Sorting

- **Input:** an array $A = (a_1, a_2, \dots, a_n)$ of elements.

- **Goal:** Sort the elements in A in non-decreasing order.
  - A permutation $(a'_1, a'_2, \dots, a'_n)$ of A such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Examples:

- Insertion sort
- Selection sort
- Merge sort
- Heap sort
- Quick sort

Sorting

- **Input:** an array $A = (a_1, a_2, \dots, a_n)$ of elements.

- **Goal:** Sort the elements in A in non-decreasing order.
  - A permutation $(a'_1, a'_2, \dots, a'_n)$ of A such that $a'_1 \le a'_2 \le \cdots \le a'_n$.

- **Intuition:** $O(n \log n)$ should be the best possible bound attainable.

Worst-case time complexity

Examples:

- Insertion sort $O(n^2)$
- Selection sort $O(n^2)$
- Merge sort $O(n \log n)$
- Heap sort $O(n \log n)$
- Quick sort $O(n^2)$

Sorting

- **Input:** an array $A = (a_1, a_2, \ldots, a_n)$ of elements.
- **Goal:** Sort the elements in A in non-decreasing order.
  - A permutation $(a'_1, a'_2, \ldots, a'_n)$ of A such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Worst-case time complexity

Examples:

- Insertion sort $O(n^2)$
- Selection sort $O(n^2)$
- Merge sort $O(n \log n)$
- Heap sort $O(n \log n)$
- Quick sort $O(n^2)$

- **Intuition:** $O(n \log n)$ should be the best possible bound attainable.
  - How to turn this intuition into an actual **lower bound** proof?
  - There are too many ways of designing a sorting algorithm.

Sorting

- **Input:** an array $A = (a_1, a_2, \dots, a_n)$ of elements.
- **Goal:** Sort the elements in A in non-decreasing order.
  - A permutation $(a_1', a_2', \dots, a_n')$ of A such that $a_1' \leq a_2' \leq \cdots \leq a_n'$.

Worst-case time complexity

Examples:

- Insertion sort $O(n^2)$
- Selection sort $O(n^2)$
- Merge sort $O(n \log n)$
- Heap sort $O(n \log n)$
- Quick sort $O(n^2)$

- **Intuition:** $O(n \log n)$ should be the best possible bound attainable.
  - How to turn this intuition into an actual **lower bound** proof?
  - There are too many ways of designing a sorting algorithm.

We will restrict our attention to a certain class of algorithms.

Comparison-based sorting

- **Comparison-based** algorithms:
  - Elements can only be compared with each other:
    - $<, \leq, =, >, \geq$
  - No other information of the elements can be used.

All of them are comparison-based.

Examples:

- Insertion sort
- Selection sort
- Merge sort
- Heap sort
- Quick sort

Comparison-based sorting

- **Comparison-based** algorithms:
  - Elements can only be compared with each other:
    - $<$, $\leq$, $=$, $>$, $\geq$
  - No other information of the elements can be used.

All of them are comparison-based.

Examples:

- Insertion sort
- Selection sort
- Merge sort
- Heap sort
- Quick sort

Allowed

If $(A[i] < A[j])$, **then** { Do some work }

If $(A[i] = A[j])$, **then** { Do some work }

Not allowed

If $(A[i] + A[j] = A[k])$, **then** { Do some work }

If $(A[i] = k)$, **then** { Do some work }

If ($A[i]$ is odd), **then** { Do some work }

If (the jth bit of $A[i]$ is 1), **then** { Do some work }

Comparison-based sorting

- **Comparison-based** algorithms:
  - Elements can only be compared with each other:
    - $<, \leq, =, >, \geq$
  - No other information of the elements can be used.

All of them are comparison-based.

Examples:

- Insertion sort $O(n^2)$
- Selection sort $O(n^2)$
- Merge sort $O(n \log n)$
- Heap sort $O(n \log n)$
- Quick sort $O(n^2)$

Theorem: The worst-case time complexity of any comparison-based sorting algorithm is $\Omega(n \log n)$.

Comparison-based sorting

- **Comparison-based** algorithms:
  - Elements can only be compared with each other:
    - $<$, $\leq$, $=$, $>$, $\geq$
  - No other information of the elements can be used.

All of them are comparison-based.

Examples:

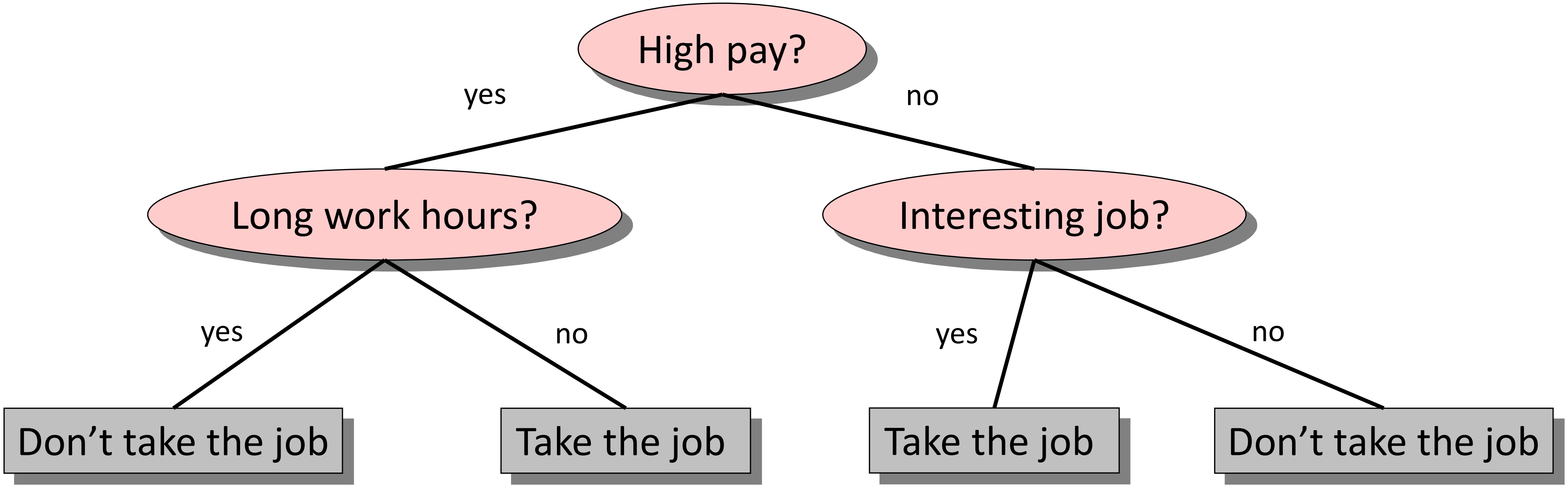
- Insertion sort $O(n^2)$
- Selection sort $O(n^2)$
- Merge sort $O(n \log n)$
- Heap sort $O(n \log n)$
- Quick sort $O(n^2)$

Theorem: The worst-case time complexity of any comparison-based sorting algorithm is $\Omega(n \log n)$.

Merge sort and heap sort are **asymptotically optimal**!

Decision trees

- The proof of the theorem uses **decision trees**.



Decision trees

- A **decision tree** is a rooted tree.
  - Start from the root.
  - At every vertex, a question is asked.
  - Depending on the answer, a child is chosen.
  - At a leaf, a decision is taken.

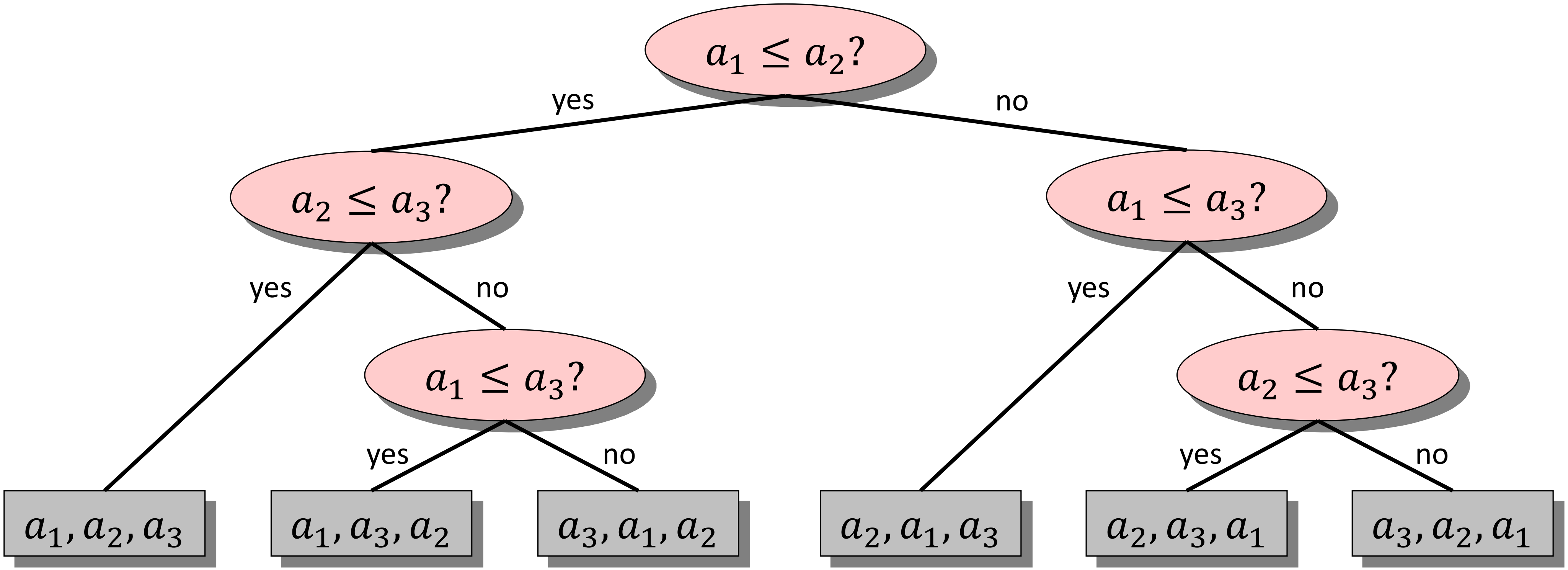
Decision trees

- A **decision tree** is a rooted tree.
  - Start from the root.
  - At every vertex, a question is asked.
  - Depending on the answer, a child is chosen.
  - At a leaf, a decision is taken.
- Any comparison-based algorithm can be modeled using a decision tree:
  - A comparison $\leftrightarrow$ A question asked at a node.
  - Program state depends on the result of the comparison $\leftrightarrow$ Chosen child depends on the answer to the question.
  - Output of the algorithm $\leftrightarrow$ Decision at a leaf.

A permutation $(a_1', a_2', \dots, a_n')$ of A

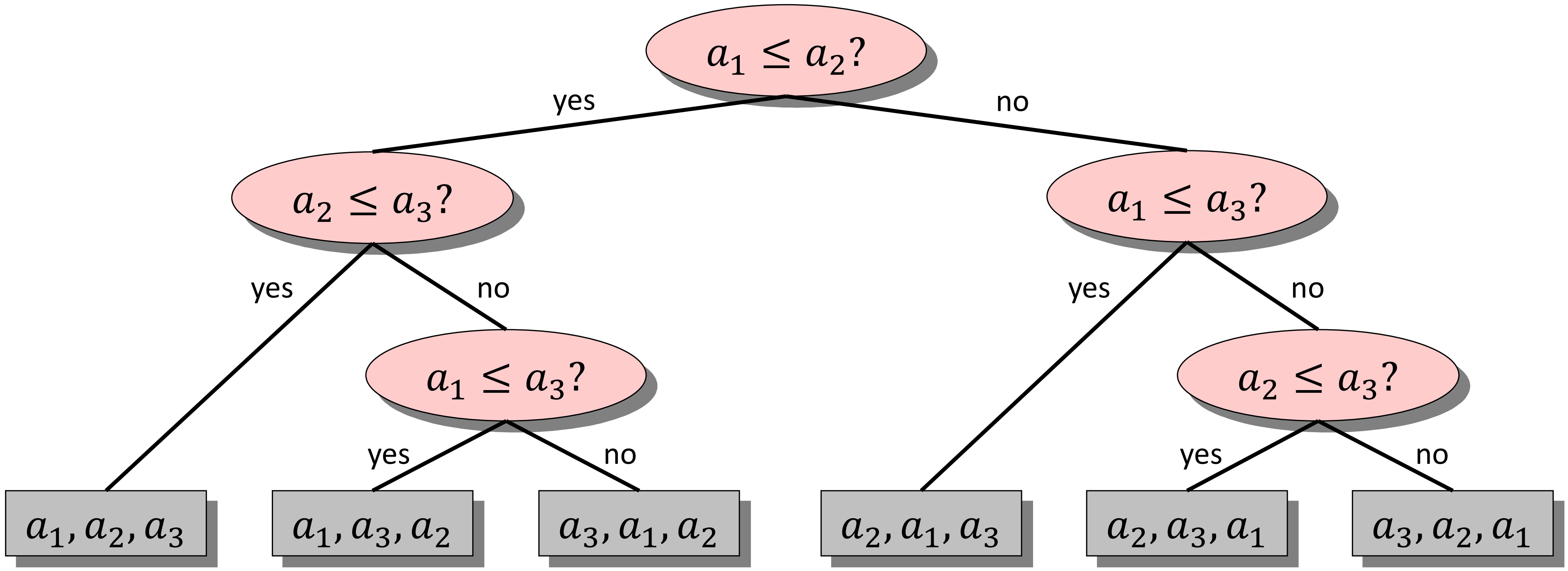
An example

- A comparison-based algorithm for sorting $A = (a_1, a_2, a_3)$.



An example

Worst-case running time $\geq$ worst-case number of comparisons $=$ height of the tree



Proof of the theorem

Theorem: The worst-case time complexity of any comparison-based sorting algorithm is $\Omega(n \log n)$.

Proof:

- Model the algorithm as a decision tree, which is a binary tree with at least $n!$ leaves:
  - Each permutation is a possible answer.
- The height of the binary tree is at least $\log(n!)$.

Proof of the theorem

Theorem: The worst-case time complexity of any comparison-based sorting algorithm is $\Omega(n \log n)$.

Proof:

- Model the algorithm as a decision tree, which is a binary tree with at least $n!$ leaves:
  - Each permutation is a possible answer.
- The height of the binary tree is at least $\log(n!)$.

$$\log(n!) \in n \log n - n \log e + O(\log n) \subseteq \Omega(n \log n)$$

Stirling's approximation
https://en.wikipedia.org/wiki/Stirling%27s_approximation

Question 1 @ VisuAlgo online quiz

- Is the following claim **true** or **false**?

> There exists a comparison-based sorting algorithm that can sort any 5-element array using at most 6 comparisons.

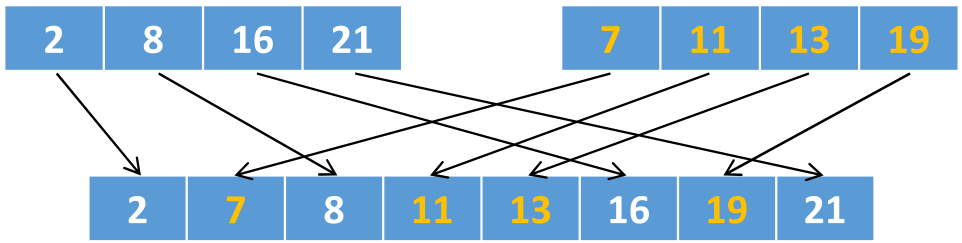
Question 2 @ VisuAlgo online quiz

Input: k sorted arrays $A_1[1..n], A_2[1..n], \ldots, A_k[1..n]$.

Goal: Merge the k sorted arrays into one sorted array of length kn.

Question: What is a tight lower bound of the worst-case running time for comparison-based algorithms for this task?

- $\Omega(kn)$
- $\Omega(kn \log k)$
- $\Omega(kn \log n)$
- $\Omega(k^2 n)$



Non-comparison sorts

Question: Can we bypass the $\Omega(n \log n)$ lower bound by an algorithm that is not comparison-based?

Non-comparison sorts

Question: Can we bypass the $\Omega(n \log n)$ lower bound by an algorithm that is not comparison-based?

Suppose each element in the array A belongs to the range $\{1,2,\ldots,k\}$.

$\mathbf{CountingSort}(A)$
- For all $i \in \{1,2,\ldots,k\}$, compute $\mathbf{count}_i =$ the number of appearances of i in A.
- Set the initial $\mathbf{count}_1$ entries of A to be 1.
- Set the next $\mathbf{count}_2$ entries of A to be 2.
- Set the next $\mathbf{count}_3$ entries of A to be 3.
- …

Exercise: Show that the algorithm can be implemented to finish in $O(n+k)$ time.

Acknowledgement

- The slides are modified from previous editions of this course and similar course elsewhere.
- **List of credits:**
  - Diptarka Chakraborty
  - Yi-Jun Chang
  - Erik Demaine
  - Steven Halim
  - Sanjay Jain
  - Wee Sun Lee
  - Charles Leiserson
  - Hon Wai Leong
  - Wing-Kin Sung