Data Structures and Algorithms

Dr. N. L. Bhanu Murthy
Computer Science & Information Systems Department

BITS Pilani
Pilani Campus

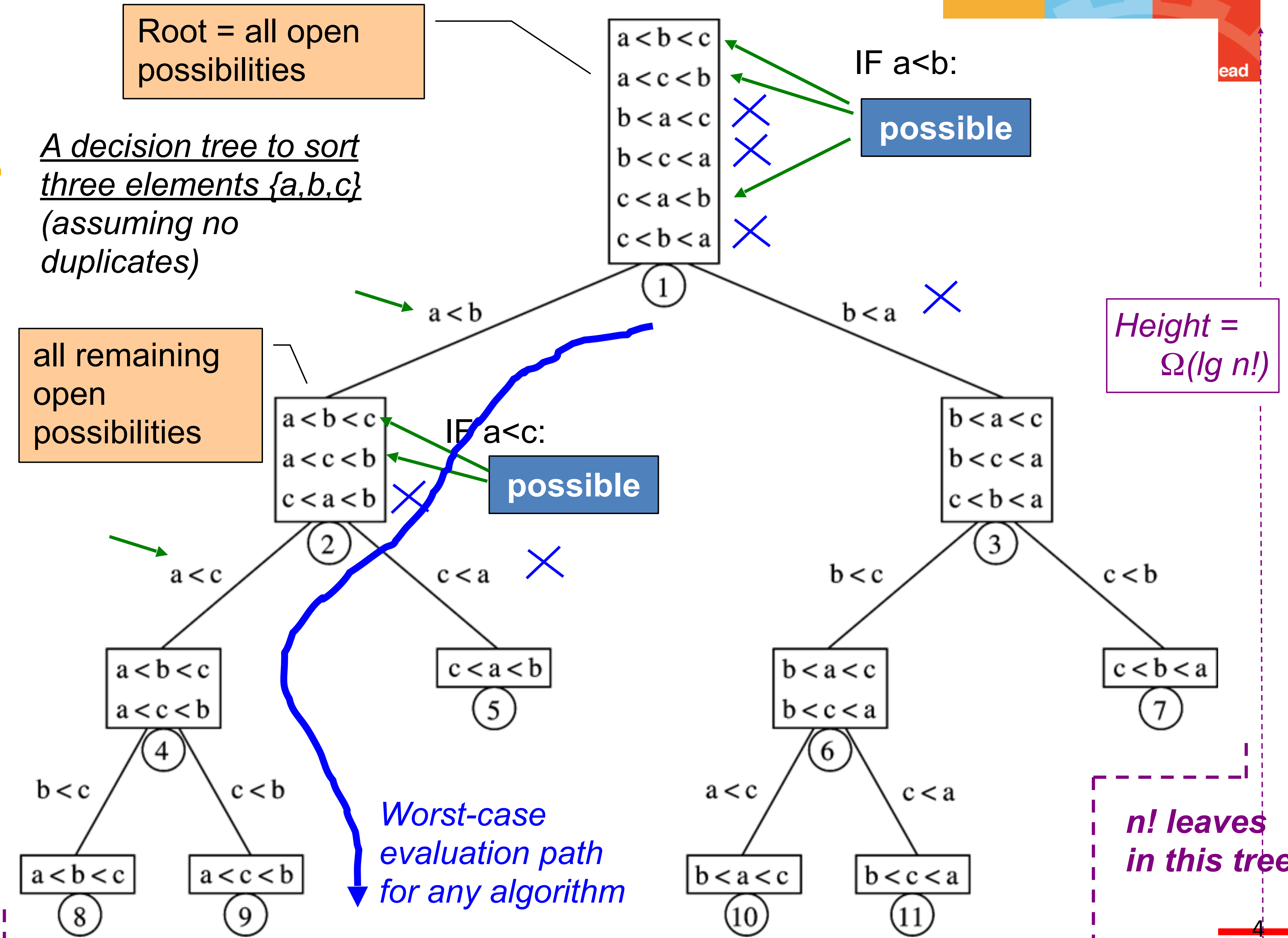
Lower Bound on Sorting Algorithms (comp based)

- ➢ What is the best we can do on comparison based sorting?
- ✓ Best worst-case sorting algorithm (so far) is O(N log N)
- ➢ Can we do better (or) Can we prove a lower bound on the sorting problem, independent of the algorithm?
- ✓ For comparison sorting, we can show lower bound of Ω(N log N)

Decision Tree for Sorting Algorithms

A *decision tree* is a binary tree where:

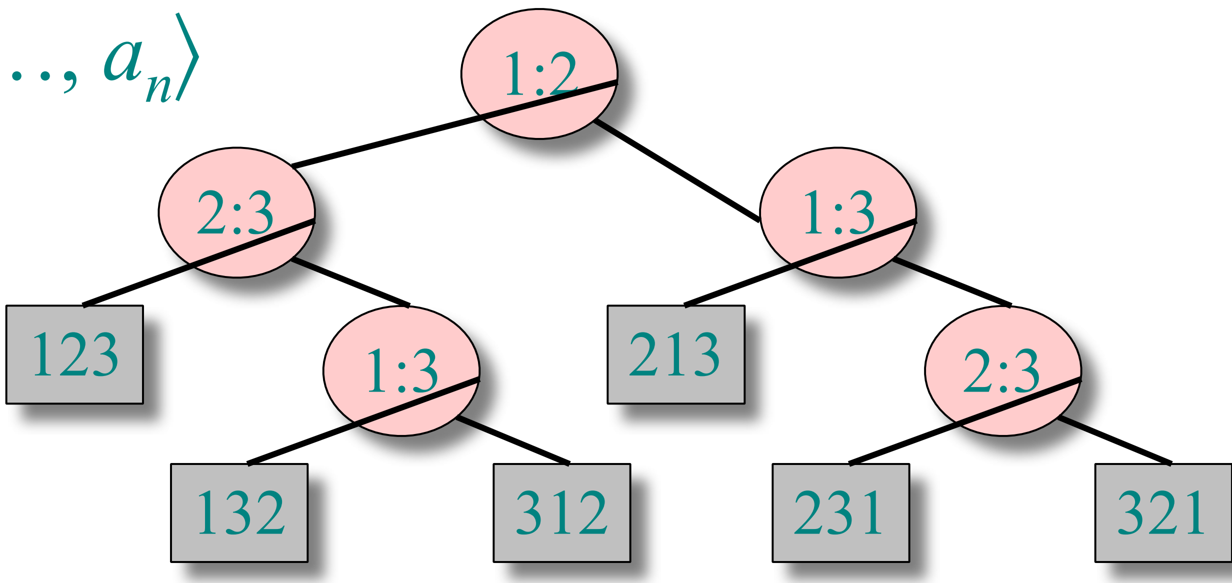
- Each node
  - lists all left-out open possibilities (for deciding)
- Path of each node
  - represents a decided sorted prefix of elements
- Each branch
  - represents an outcome of a particular comparison
- Each leaf
  - represents a particular ordering of the original array elements



Root = all open possibilities
A decision tree to sort three elements {a,b,c} (assuming no duplicates)
a<b<c
a<c<b
b<a<c
b<c<a
c<a<b
c<b<a
IF a<b:
possible
1
a<b
b<a
Height = Ω(lg n!)
all remaining open possibilities
a<b<c
a<c<b
c<a<b
IF a<c:
possible
2
b<a<c
b<c<a
c<b<a
3
a<c
c<a
b<c
c<b
a<b<c
a<c<b
4
c<a<b
5
b<a<c
b<c<a
6
c<b<a
7
b<c
c<b
a<c
c<a
Worst-case evaluation path for any algorithm
n! leaves in this tree
a<b<c
8
a<c<b
9
b<a<c
10
b<c<a
11

Decision-tree example

Sort $\langle a_1, a_2, \ldots, a_n \rangle$

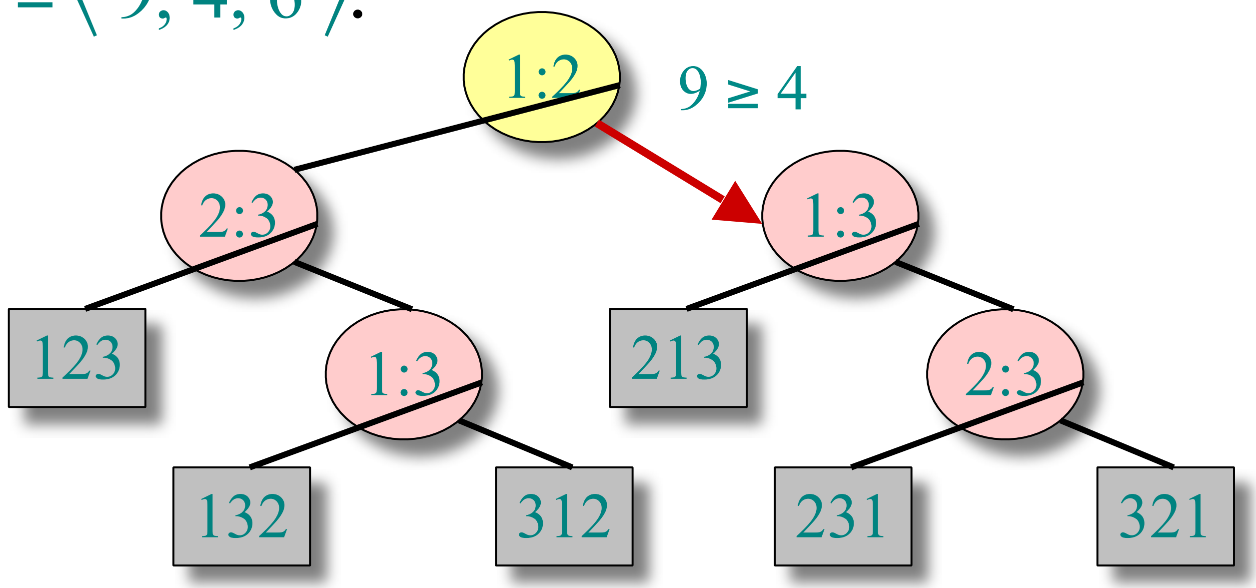


Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \le a_j$.
- The right subtree shows subsequent comparisons if $a_i \ge a_j$.

Decision-tree example

Sort $\langle a_1, a_2, a_3 \rangle = \langle\ 9, 4, 6\ \rangle$:

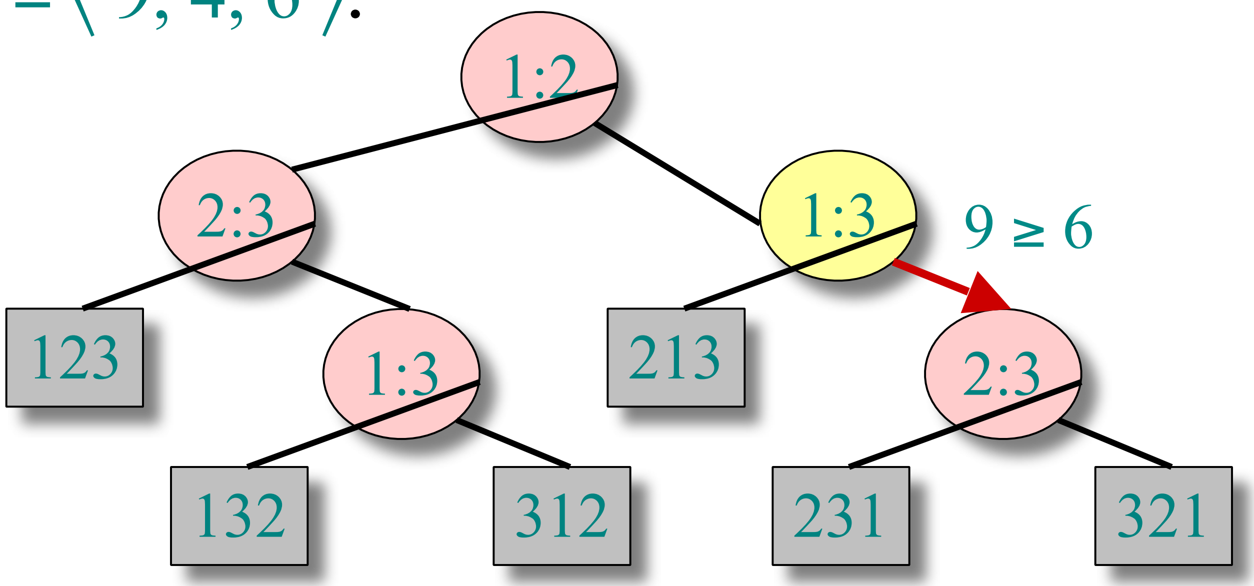


Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \leq a_j$.
- The right subtree shows subsequent comparisons if $a_i \geq a_j$.

Decision-tree example

Sort $\langle a_1, a_2, a_3 \rangle = \langle\ 9, 4, 6\ \rangle$:

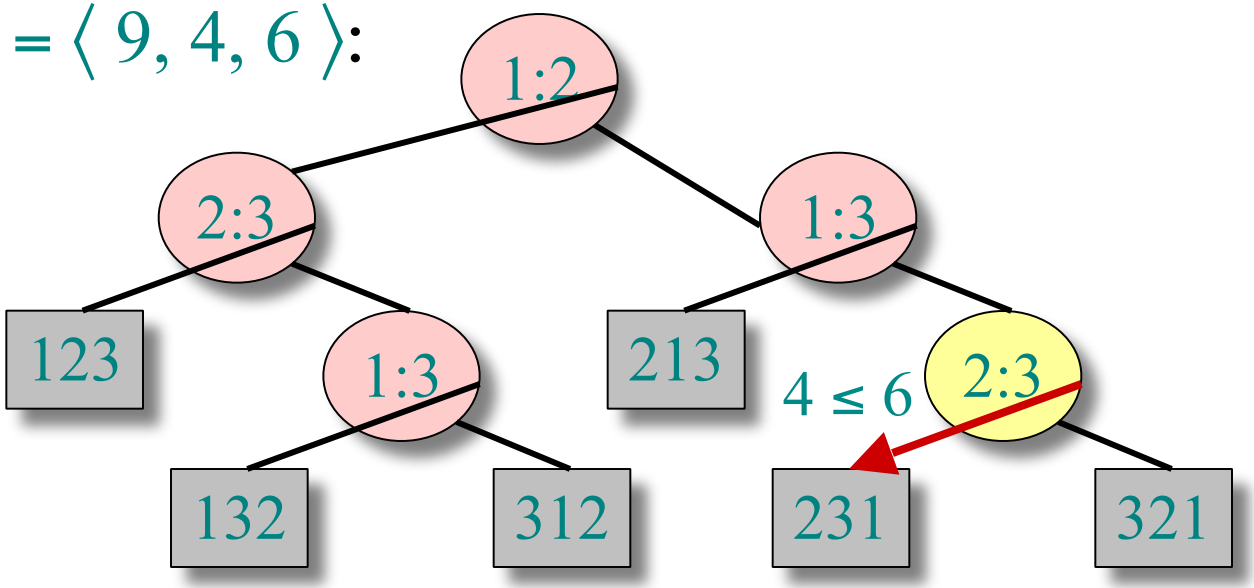


Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \leq a_j$.
- The right subtree shows subsequent comparisons if $a_i \geq a_j$.

Decision-tree example

Sort $\langle a_1, a_2, a_3 \rangle = \langle\ 9, 4, 6\ \rangle$:

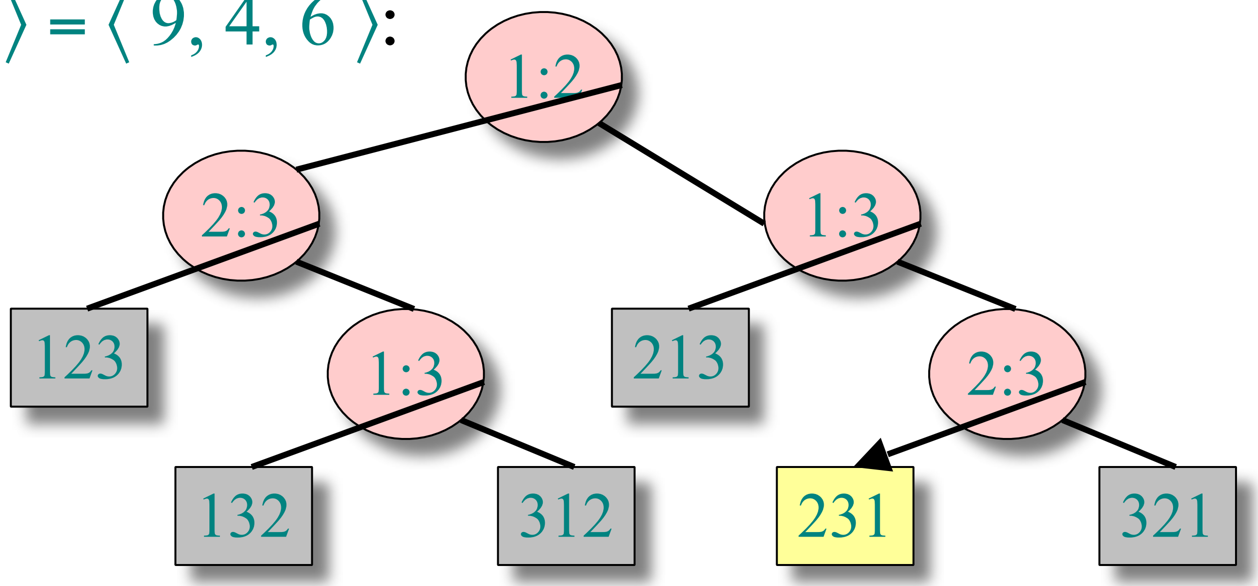


Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \leq a_j$.
- The right subtree shows subsequent comparisons if $a_i \geq a_j$.

Decision-tree example

Sort $\langle a_1, a_2, a_3 \rangle = \langle\, 9, 4, 6 \,\rangle$:



$4 \le 6 \le 9$

Each leaf contains a permutation $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ to indicate that the ordering $a_{\pi(1)} \le a_{\pi(2)} \le \boxed{?} \le a_{\pi(n)}$ has been established.

Decision Tree for Sorting Algorithms

- ✓ The logic of *any sorting algorithm* that uses comparisons can be represented by a decision tree
- ✓ In the worst case, the number of comparisons used by the algorithm equals the HEIGHT OF THE DECISION TREE
- ✓ In the average case, the number of comparisons is the average of the depths of all leaves
- ✓ There are N! different orderings of N elements

Lower bound for decision-tree sorting

Theorem. Any decision tree that can sort n elements must have height $\Omega(n \lg n)$.

Proof. The tree must contain $\geq n!$ leaves, since there are $n!$ possible permutations.

A height-h binary tree has $\leq 2^h$ leaves.

Thus, $n! \leq 2^h$.

$$
\begin{aligned}
\therefore h &\geq \lg(n!) && \text{(lg is mono. increasing)} \\
&\geq \lg((n/e)^n) && \text{(Stirling's formula)} \\
&= n \lg n - n \lg e \\
&= \Omega(n \lg n) .
\end{aligned}
$$

since n lg n – n lg e >= 1/2 n lg n for all n >= n_0 (take n_0 to be 100)

Problem Complexity

the time complexity of Problem P is $O(f(n))$ if

there exists an $O(f(n))$-time algorithm that solves Problem P

the time complexity of Problem P is $\Omega(f(n))$ if

any algorithm that solves Problem P requires $\Omega(f(n))$ time.

Optimal Algorithm

We say that Algorithm A is an *optimal algorithm* for Problem P in terms of worst-case time complexity if

- Algorithm A runs in time $O(f(n))$; and
- the time complexity of Problem P is $\Omega(f(n))$ in the worst case.

Corollary. Merge sort and Heap Sort are asymptotically optimal comparison sorting algorithms.

We say that the (worst-case) time complexity of Problem P is $\Theta(f(n))$ if

- the time complexity of Problem P is $O(f(n))$, i.e.

  there exists an $O(f(n))$-time algorithm that solves Problem P; and

- the time complexity of Problem P is $\Omega(f(n))$, i.e.

  any algorithm that solves Problem P requires $\Omega(f(n))$ time.

Comparing P and Q

We say that Problem P is no harder than Problem Q in terms of (worst-case) time complexity if there exists a function $f(n)$ such that

- the (worst-case) time complexity of Problem P is $O(f(n))$; and
- the (worst-case) time complexity of Problem Q is $\Omega(f(n))$.

Comparing P and Q

We say that Problem P is (strictly) *easier* than Problem Q in terms of (worst-case) time complexity if there exists a function $f(n)$ such that

- the (worst-case) time complexity of Problem P is $O(f(n))$; and
- the (worst-case) time complexity of Problem Q is $\omega(f(n))$

or

- the (worst-case) time complexity of Problem P is $o(f(n))$; and
- the (worst-case) time complexity of Problem Q is $\Omega(f(n))$.

innovate achieve lead

Thank You!!