Introduction to Algorithms

Meng-Tsung Tsai

09/12/2019

Course Materials

Textbook

Introduction to Algorithms (I2A) 3rd ed. by Cormen, Leiserson, Rivest, and Stein.

Reference Book

Algorithms (JfA) 1st ed. by Erickson. An e-copy can be downloaded from author's website: http://jeffe.cs.illinois.edu/teaching/algorithms/

Websites

http://e3new.nctu.edu.tw for slides, written assignments, and solutions.

http://oj.nctu.me for programming assignments.

Office Hours

Lecturer's

On Wednesdays 16:30 - 17:20 at EC 336 (工程三館).

TA. Erh-Hsuan Lu (呂爾軒) and Tsung-Ta Wu (吳宗達)

On Mondays 10:10 - 11:00 at ES 724 (電資大樓).

TA. Yung-Ping Wang (王詠平) and Chien-An Yu (俞建安)

On Thursdays 11:10 - 12:00 at ES 724 (電資大樓).

Errata

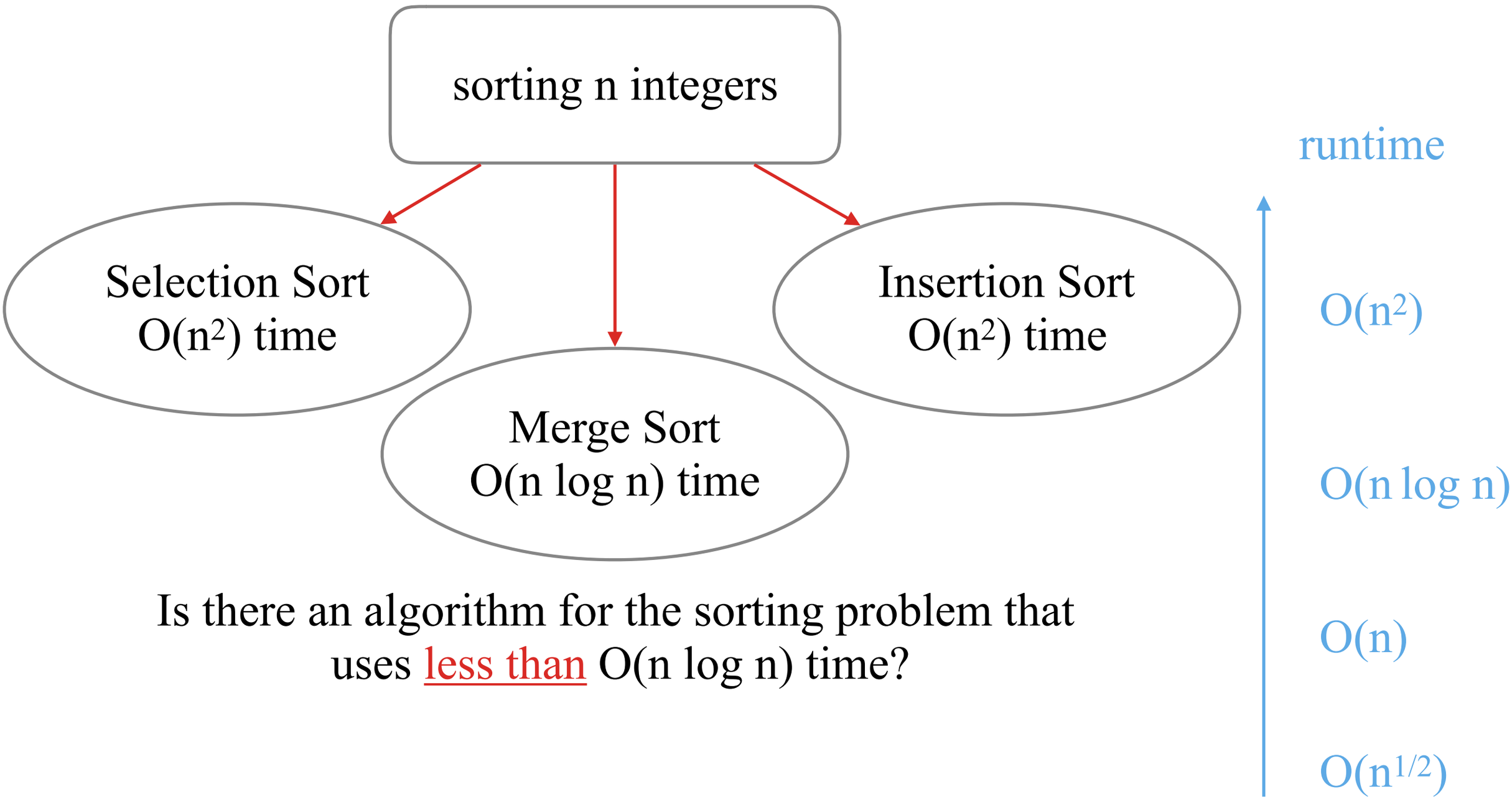
I get some error fixed based on your feedbacks. Thank you.

The fixed part has background-color yellow.

Lower Bounds

Lower Bounds

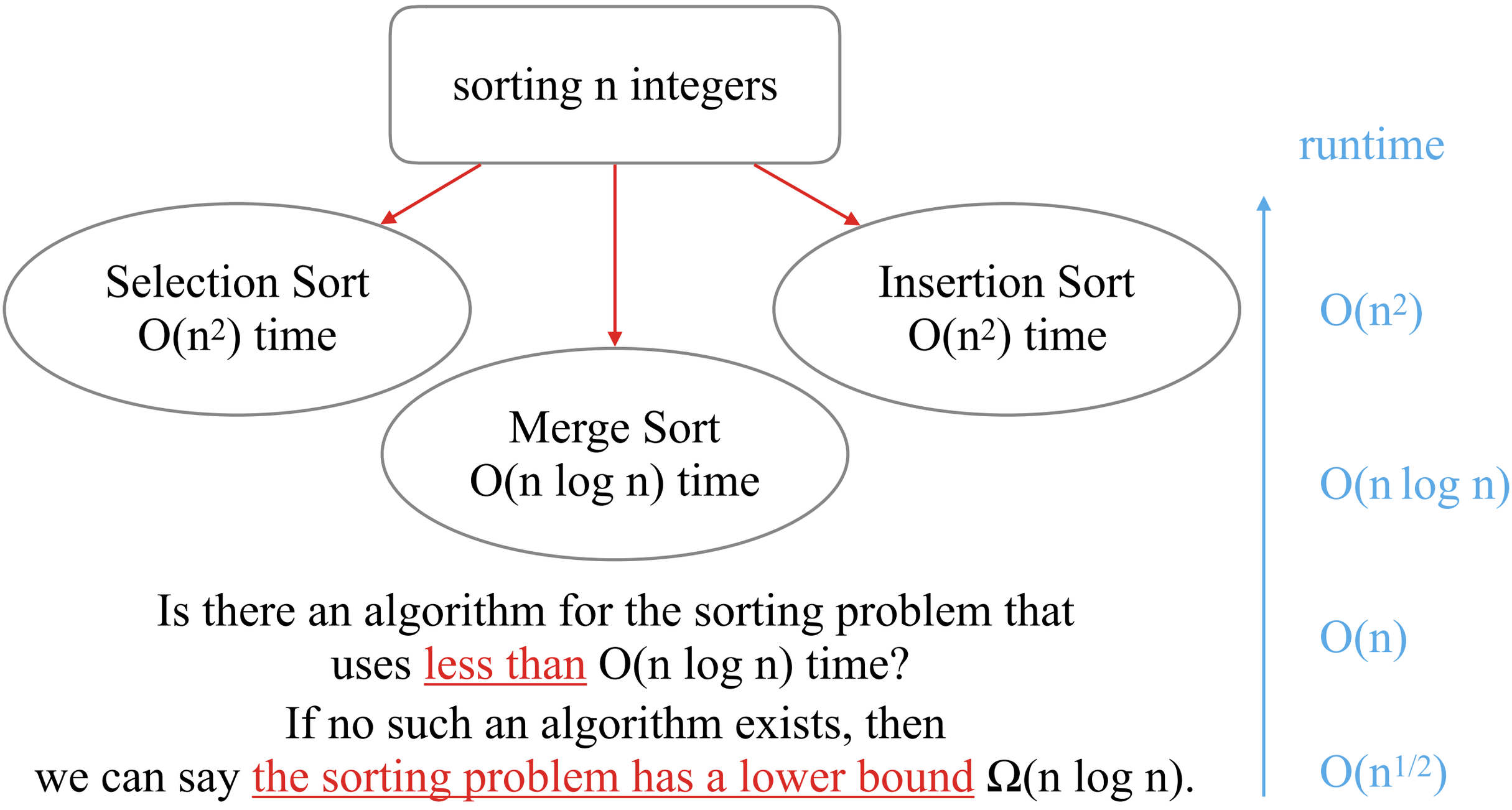
We have seen three different algorithms for the sorting problem.



Is there an algorithm for the sorting problem that uses less than O(n log n) time?

Lower Bounds

We have seen three different algorithms for the sorting problem.

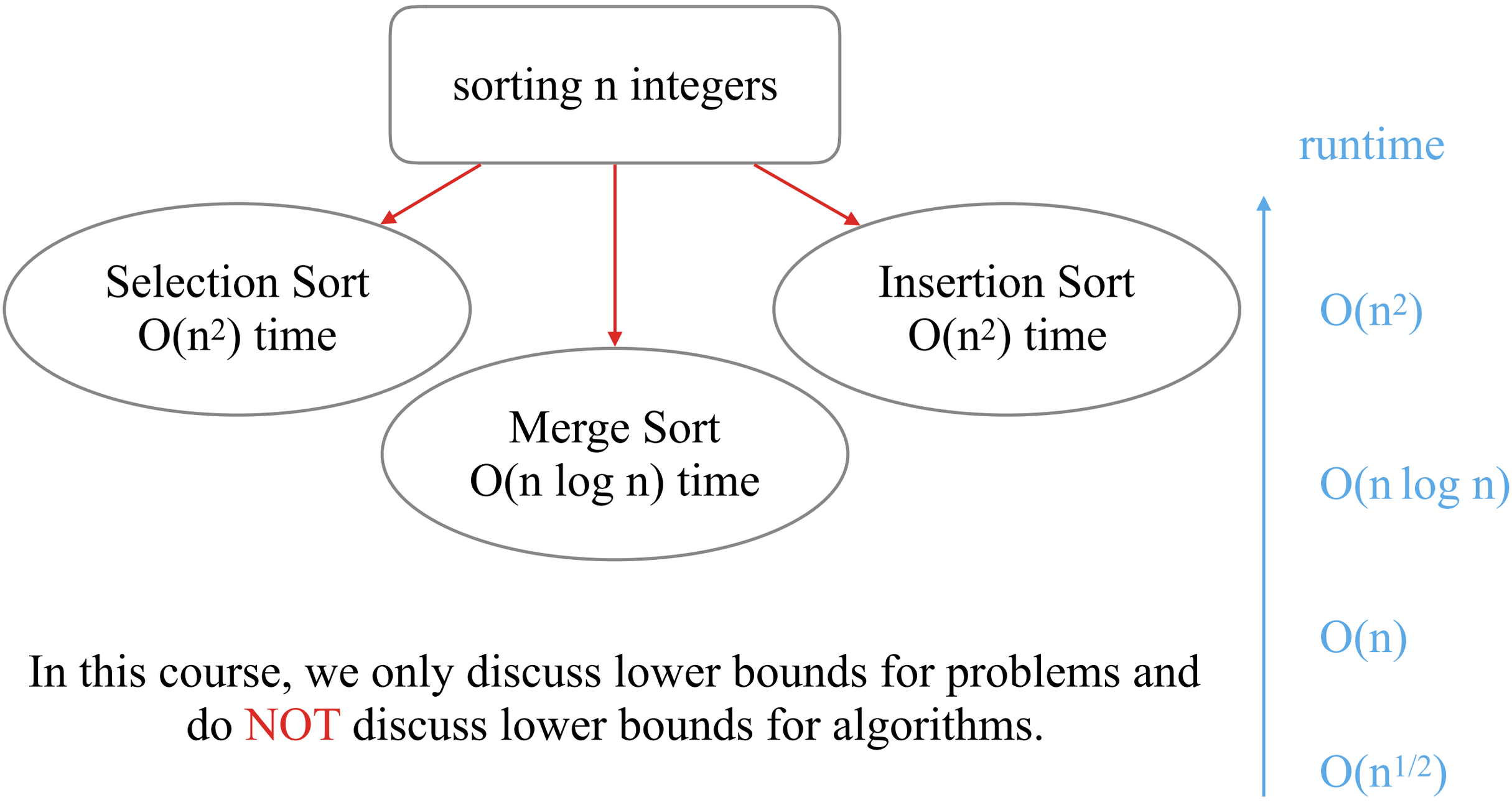


Is there an algorithm for the sorting problem that uses less than O(n log n) time?

If no such an algorithm exists, then we can say the sorting problem has a lower bound $\Omega(n \log n)$.

Lower Bounds

We have seen three different algorithms for the sorting problem.



In this course, we only discuss lower bounds for problems and do NOT discuss lower bounds for algorithms.

Membership Query

Membership Query Problem

Input: a sorted array A of length n in nondecremental order and a query x.

Output: "Yes", if x = A[k] for some k in [1, n], or otherwise "No".

Instance.

$x = 5$	$x = 14$
Yes	No

Membership Query Problem

Input: a sorted array A of length n in nondecremental order and a query x.

Output: "Yes", if x = A[k] for some k in [1, n], or otherwise "No".

Algorithm 1: Linear scan.

Clearly, it runs in O(n) time.

Membership Query Problem

Input: a sorted array A of length n in nondecremental order and a query x.

Output: "Yes", if x = A[k] for some k in [1, n], or otherwise "No".

Algorithm 1: Linear scan.

Clearly, it runs in O(n) time.

Membership Query Problem

Input: a sorted array A of length n in nondecremental order and a query x.

Output: "Yes", if x = A[k] for some k in [1, n], or otherwise "No".

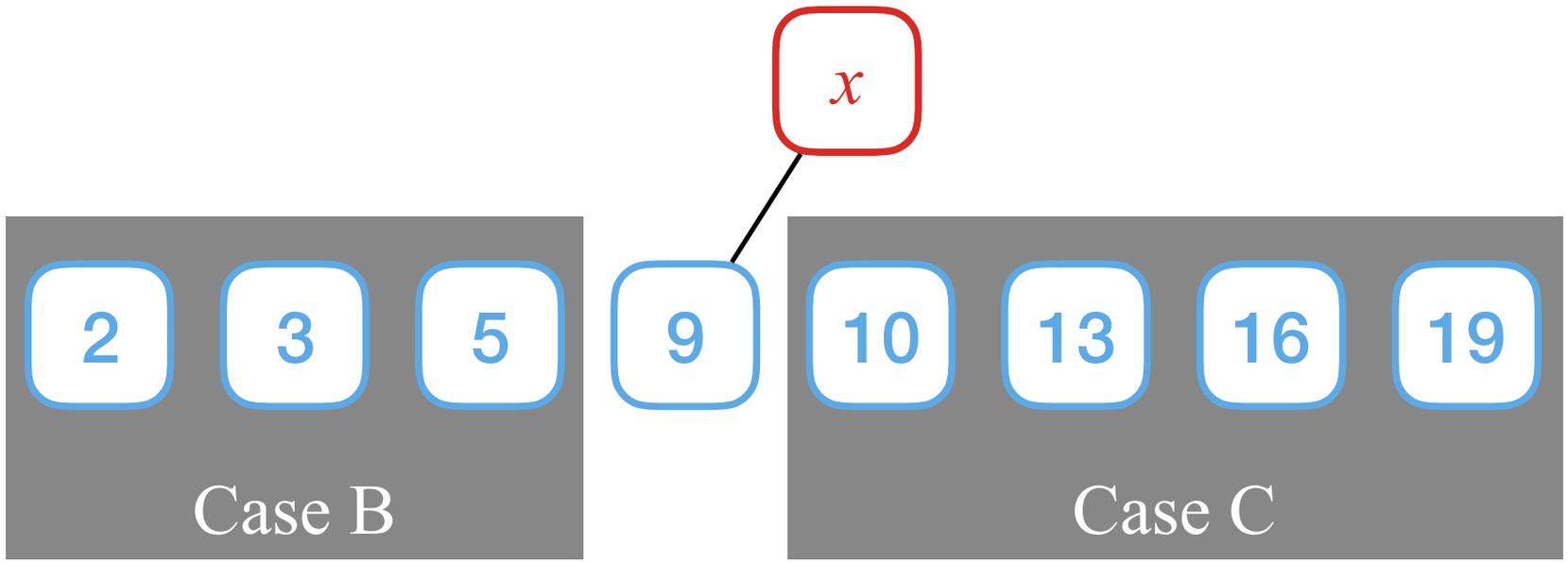
Algorithm 1: Linear scan.

Clearly, it runs in O(n) time.

Can we do better?

Membership Query Problem

Algorithm 2: Binary search.

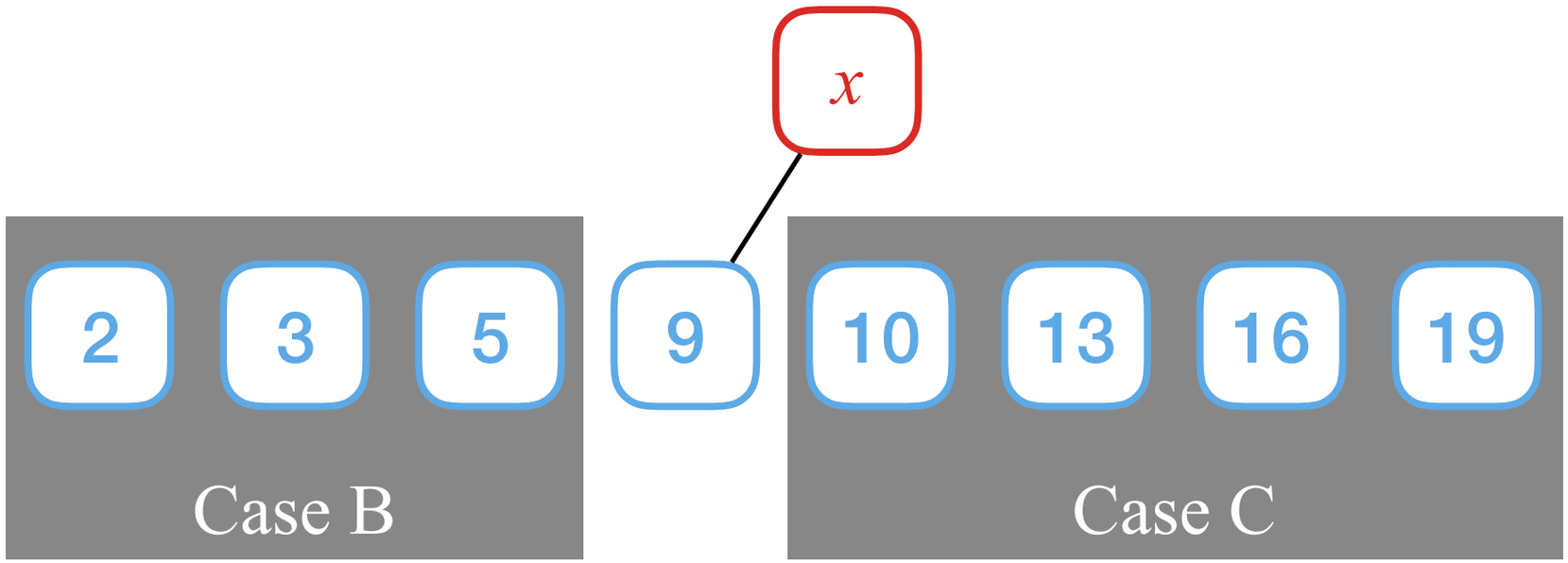


Case A. If $x = 9$, output "Yes."
Case B. If $x < 9$, recurse on the subproblem $(x, \{2, 3, 5\})$
Case C. If $x > 9$, recurse on the subproblem $(x, \{10, 13, 16, 19\})$

Case D. For subproblem $(x, \{\})$, output "No."

Membership Query Problem

Algorithm 2: Binary search.



Case A. If $x = 9$, output "Yes."
Case B. If $x < 9$, recurse on the subproblem $(x, \{2, 3, 5\})$
Case C. If $x > 9$, recurse on the subproblem $(x, \{10, 13, 16, 19\})$

Case D. For subproblem $(x, \{\})$, output "No."

It runs in O(log n) time. Why?

Membership Query Problem

Algorithm 2: Binary search.

Case A. If $x = 9$, output "Yes."
Case B. If $x < 9$, recurse on the subproblem $(x, \{2, 3, 5\})$
Case C. If $x > 9$, recurse on the subproblem $(x, \{10, 13, 16, 19\})$
Case D. For subproblem $(x, \{\})$, output "No."

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

By the recursion-tree method, we have a guess $T(n) = O(\log n)$.

$T(n) = T(n/2) + 1 = T(n/4) + 1 + 1 = \ldots = \log_2 n + O(1)$.

Membership Query Problem

Algorithm 2: Binary search.

Case A. If $x = 9$, output "Yes."
Case B. If $x < 9$, recurse on the subproblem (x, {2, 3, 5})
Case C. If $x > 9$, recurse on the subproblem (x, {10, 13, 16, 19})
Case D. For subproblem (x, {}), output "No."

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

By the substitution method, we verify the guess T(n) = O(log n).

Membership Query Problem

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

By the substitution method, we verify the guess T(n) = O(log n).

Claim: T(n) $\leq$ d $\log_2$ n + d for some constant d > 0.

For n = 1, T(1) = O(1) $\leq$ d. The claim holds by picking a large constant d.
For n < k, we assume the claim holds.
For n = k ≥ 2, $T(k) = T(\lfloor k/2 \rfloor) + O(1)$

$\leq d \log_2 \lfloor k/2 \rfloor + d + O(1)$ by induction hypothesis.

$\leq d \log_2 (k/2) + d + O(1)$

$= d \log_2 k + O(1)$

$\leq d \log_2 k + d$ if we pick a sufficiently large constant d.

By induction on n, the claim holds for every integer n $\geq$ 1,
so T(n) $\leq$ d $\log_2$ n + d = O($\log_2$ n) by picking n_0 = 2 and C = 2d.

Exercise

In the following link, you may find a simple explanation for mathematical induction.

https://calculus.nipissingu.ca/tutorials/induction.html

Exercise

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

Prove that T(n) = O(log n).

Prove that $O(\log_2 n) = O(\ln n) = O(\log_c n)$ for any constant $c > 1$.

Consequently, we seldom spell out the base in a logarithm.

Membership Query Problem

Can we do better than the binary search? Algorithm 3?

Maybe no. Why?

Comparison-Based Model

If the relative order between x and elements in A can only be determined by comparisons, then the answer is NO.

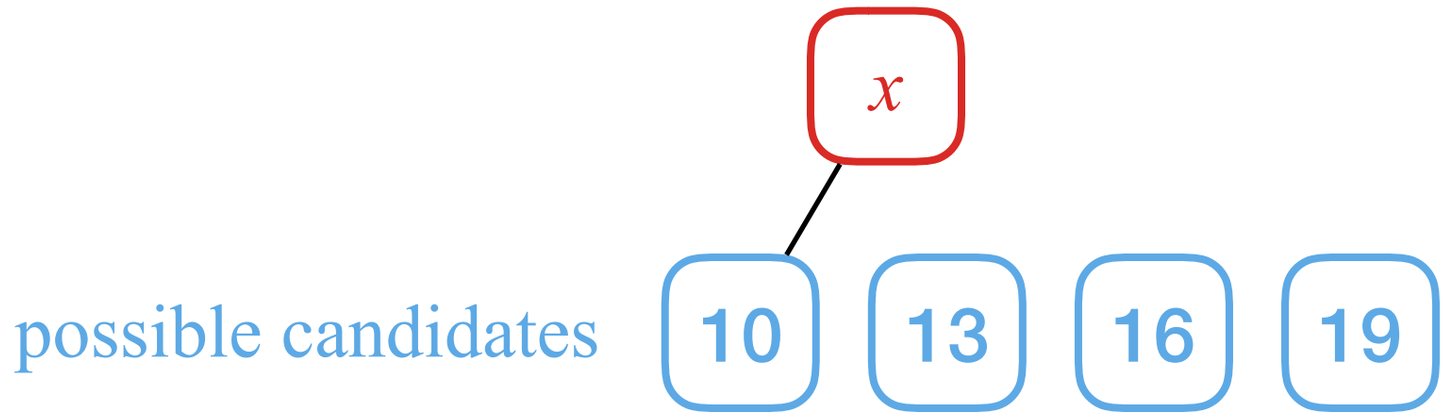
You may interpret the above restriction as that

only Alice knows what x and A are,

and you can request Alice to compare x with A[k] for some k in [1, n].
She will tell you the outcome whether A[k] = x, or A[k] < x, or A[k] > x.
The goal is to minimize the number of requests.

Adversary Game

Here is why.



Alice can play this adversary game to maximize the number of requests.

If you ask to compare x with 10 or 13, she always says x is larger.

If you ask to compare x with 16 or 19, she always says x is smaller.

In either case, you can eliminate at most a half of possible candidates.

Any algorithm needs to either find where x is or reduce the number of possible candiates to 0, so $\log_2 n$ comparisons is necessary.

Summary

(1) There exists an algorithm in the comparison-based model that can solve the membership query problem in $O(\log n)$ time.

(2) Any algorithm that can solve the membership query problem in the comparison-based model requires at least $\log_2 n = \Omega(\log n)$ time.

Explained later.

Exercise

Membership Query (restricted):

Input: a sorted array A of length n in nondecremental order and a query x where each element in A has value either 1 or 2.

Output: "Yes", if x = A[k] for some k in [1, n], or otherwise "No".

How many comparisons do you need?

Why can we break the lower bound $\Omega(\log n)$? Is our proof flawed?

Exercise

Membership Query (restricted):

Input: a sorted array A of length n in nondecremental order and a query x where each element in A has value in {1, 2, 3}.

Output: "Yes", if x = A[k] for some k in [1, n], or otherwise "No".

How many comparisons do you need?

More Asymptotic Notations

Ω-notation

Ω(g(n)) is pronounced as big-omega of g of n.

Ω(g(n)) is the set of functions that

$$\{f(n) : \text{there exist positive constants } n_0 \text{ and } C \text{ so that for every } n \geq n_0, 0 \leq C \cdot g(n) \leq f(n)\}.$$

Examples.

1) $2n^2 = \Omega(n^2)$
2) $7n - 1000 = \Omega(n)$
3) $2n^3 - 20n = \Omega(n^2)$

What are n_0 and C for these examples?

Θ-notation

Θ(g(n)) is pronounced as theta of g of n.

f(n) = Θ(g(n)) is equivalent to f(n) = O(g(n)) and f(n) = Ω(g(n)).

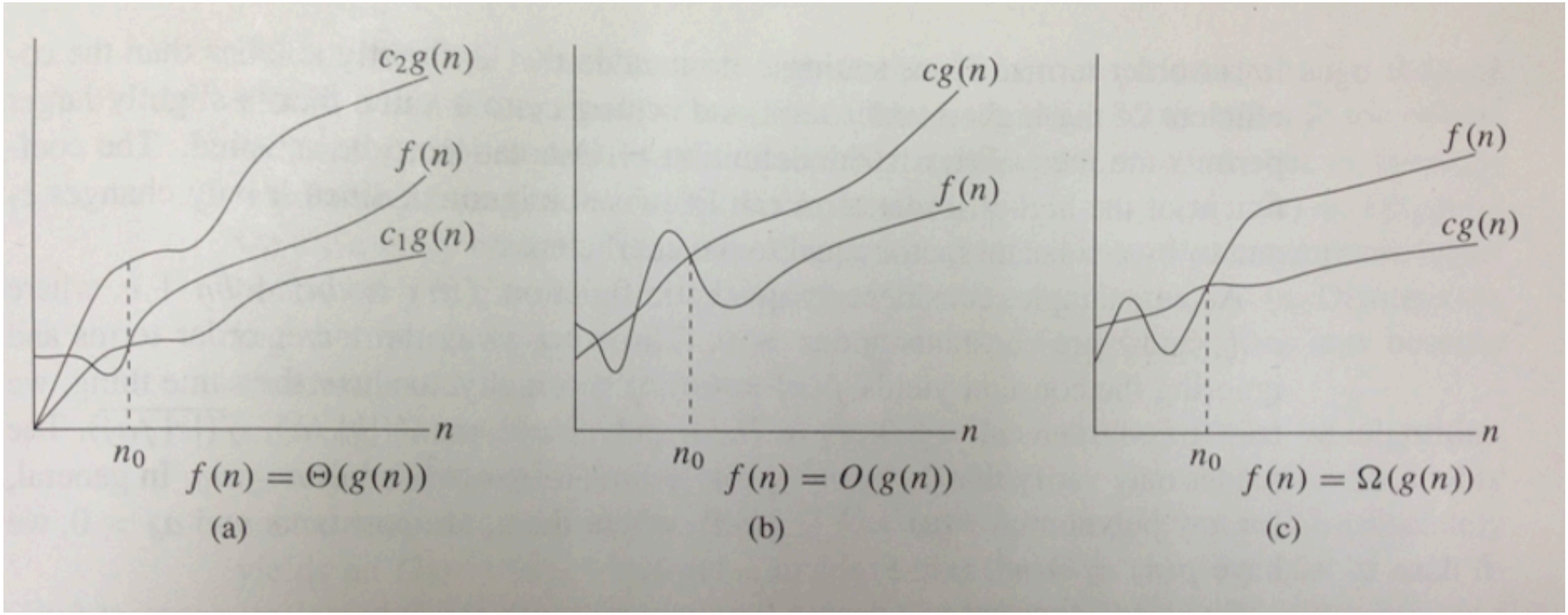
Θ(g(n)) is the set of functions that

$\{f(n)$: there exist positive constants n_0, C_1, and C_2 so that
for every $n \geq n_0$, $0 \leq C_1 \cdot g(n) \leq f(n) \leq C_2 \cdot g(n)\}$.

Examples.

1) $2n^2 = \Theta(n^2)$
2) $7n - 1000 = \Theta(n)$
3) $2n^3 - 20n \neq \Theta(n^2)$

Graphic examples of Θ, O, Ω-notation



Excerpts made from I2A.

o-notation

o(g(n)) is pronounced as little-o of g of n.

f(n) = o(g(n)) if and only if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

Examples.

1) $2n^3 = o(n^3 \log n)$
2) $n \neq o(n)$
3) $1/\log n = o(1)$

If f(n) = o(g(n)), then f(n) = O(g(n)).
The reverse direction is not true.

Exercise

Sometimes it is easier to prove f(n) = O(g(n)) by L'Hôpital's rule.

Prove that $\log^2 n = O(n)$.

$$\begin{aligned}\lim_{n\to\infty} \frac{\log^2 n}{n} &= \lim_{n\to\infty} \frac{2\log n}{n} \\ &= \lim_{n\to\infty} \frac{2}{n} \\ &= 0\end{aligned}$$

We get $\log^2 n = o(n)$, and therefore $\log^2 n = O(n)$.

Prove that $\log^4 n = O(n)$.

ω-notation

ω(g(n)) is pronounced as little-omega of g of n.

f(n) = ω(g(n)) if and only if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

Examples.

1) $2n^3 = \omega(n^2)$
2) $n \neq \omega(n)$
3) $\log n = \omega(1)$

If f(n) = ω(g(n)), then f(n) = Ω(g(n)).
The reverse direction is not true.

Informal Intepretation

$f(n) = O(g(n)) \rightarrow f(n) \leq g(n)$ up to a multiplicative constant factor

$f(n) = \Omega(g(n)) \rightarrow f(n) \geq g(n)$ ...

$f(n) = \Theta(g(n)) \rightarrow f(n) = g(n)$ ...

$f(n) = o(g(n)) \rightarrow f(n) < g(n)$ ...

$f(n) = \omega(g(n)) \rightarrow f(n) > g(n)$ ...

Asymptotically Optimal

If a problem P has a lower bound $\Omega(f(n))$ for some $f(n)$, and P can be solved by an algorithm that runs in $O(f(n))$ time, then we have:

1) the lower bound cannot be raised,
2) the upper bound cannot be reduced,
3) the algorithm is *asymptocally optimal*, and
4) we cannot improve the algorithm by any superconstant factor.

Superconstant means $\omega(1)$ and *subconstant* means $o(1)$.

Sorting Lower Bound

Sorting Problem

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

Merge sort runs in O(n log n) time.
Can we do better?

Sorting in the Comparison-Based Model

We will see why any sorting algorithm in the comparison-based model requires $\Omega(n \log n)$ runtime. This implies that Merge Sort is asymptotically optimal in the comparison-based model.

(Recap) In the comparison-based model, the relative order between any pair of elements in the given array can only be determined by comparisons.

Worst Case

While stating the runtime of an algorithm or the lower bound of a problem, we hope our statements hold for all possible input.

Thus, as long as there is a single case make an algorithm slow, we say the algorithm is slow. We will relax this concept in the next lectures.

The (worst) case we are going to use for the sorting problem is that

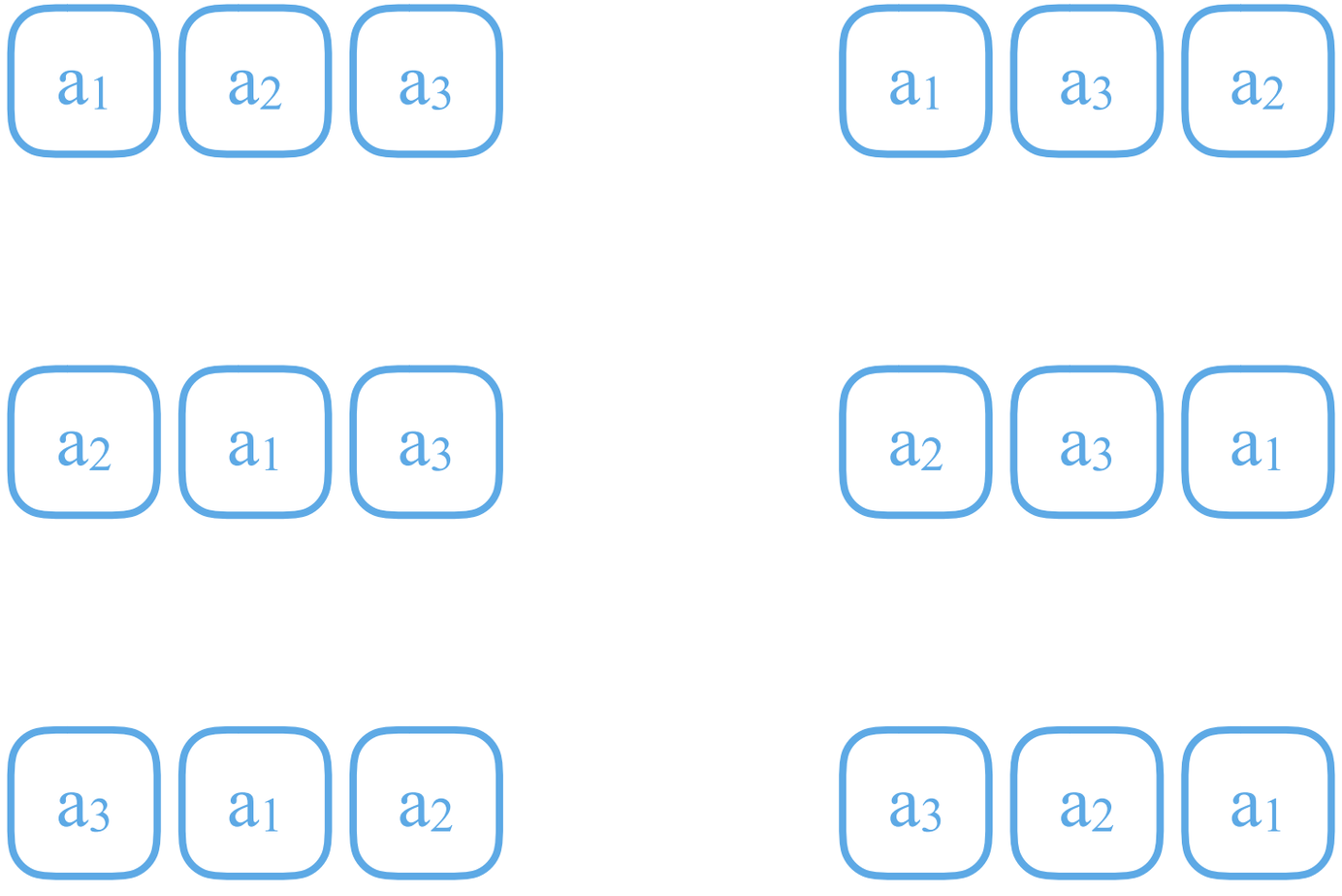
the n input integers are distinct.

Hence, exactly one of the n! permutations of the n integers is the correct output order.

Adversary Game

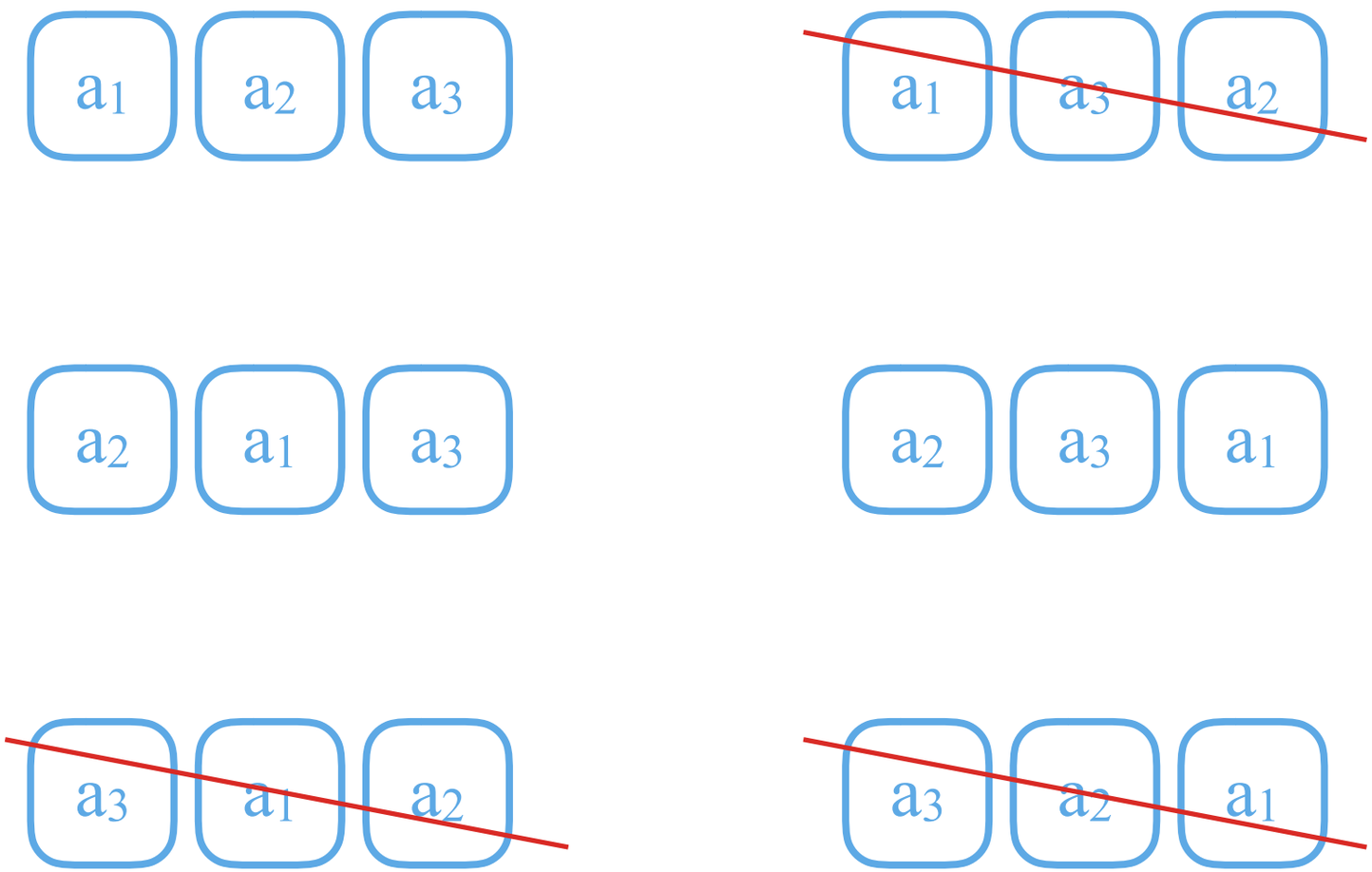
An example of n = 3.

Initially, any of the 3! permutations could be a valid output. We say a permutation *alive* if it is still be a valid output.



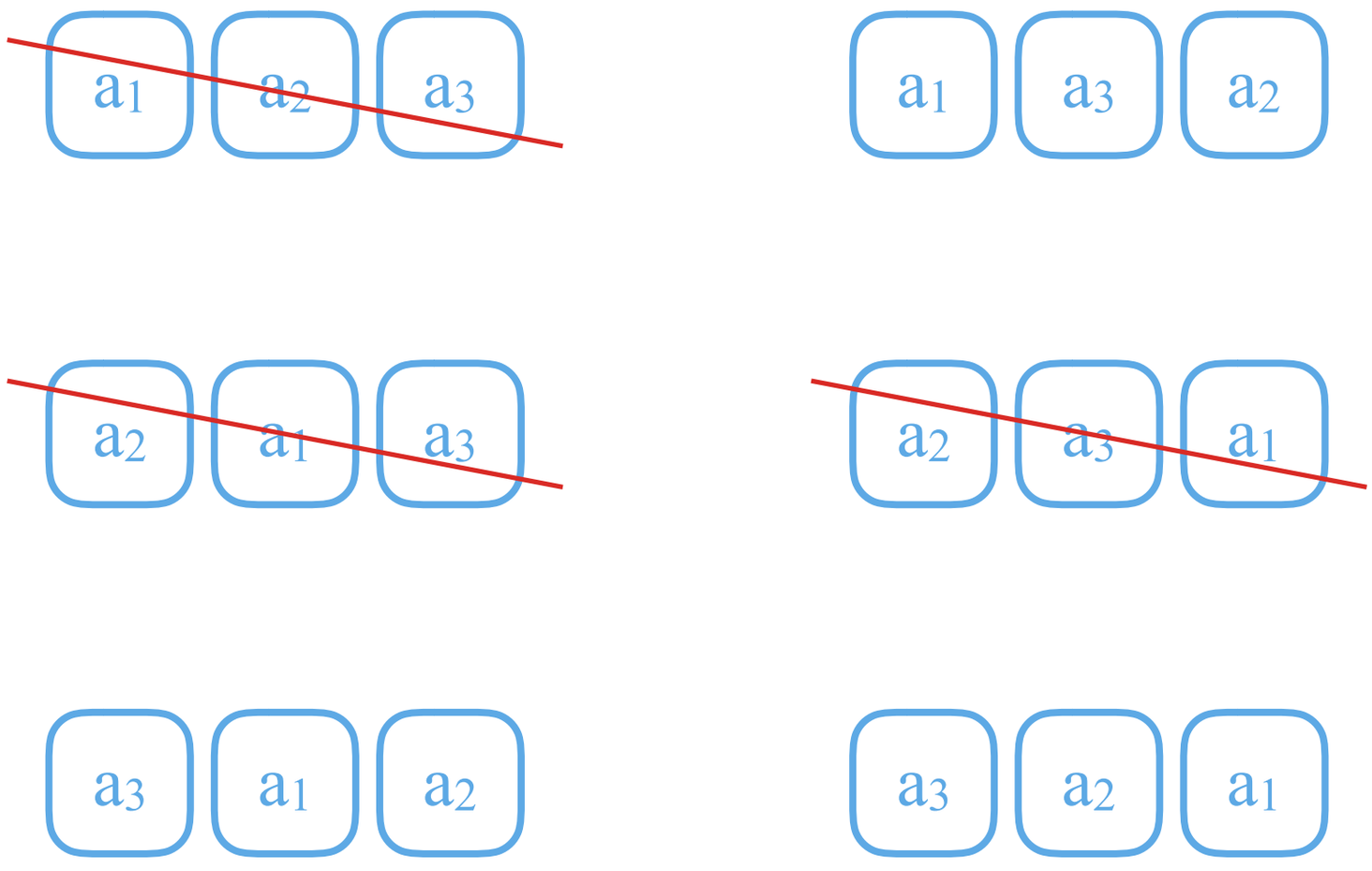
Adversary Game

Say an algorithm asks to compare a_2 with a_3. If Alice replies $a_2 < a_3$, then some permutations are no longer a valid output. We call such permutations *dead*.



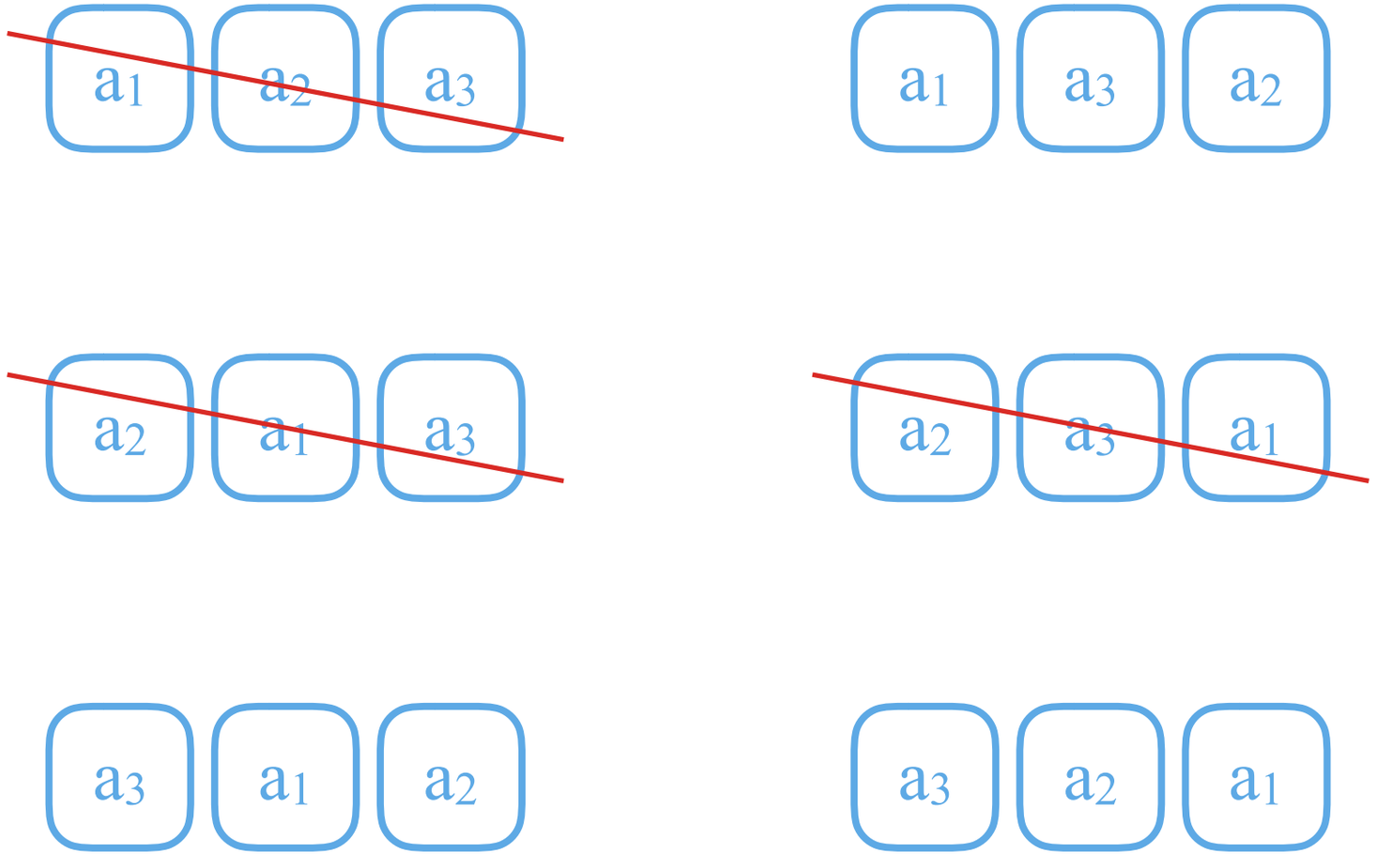
Adversary Game

Say an algorithm asks to compare a_2 with a_3. If Alice replies $a_2 > a_3$, then the rest permutations are dead.



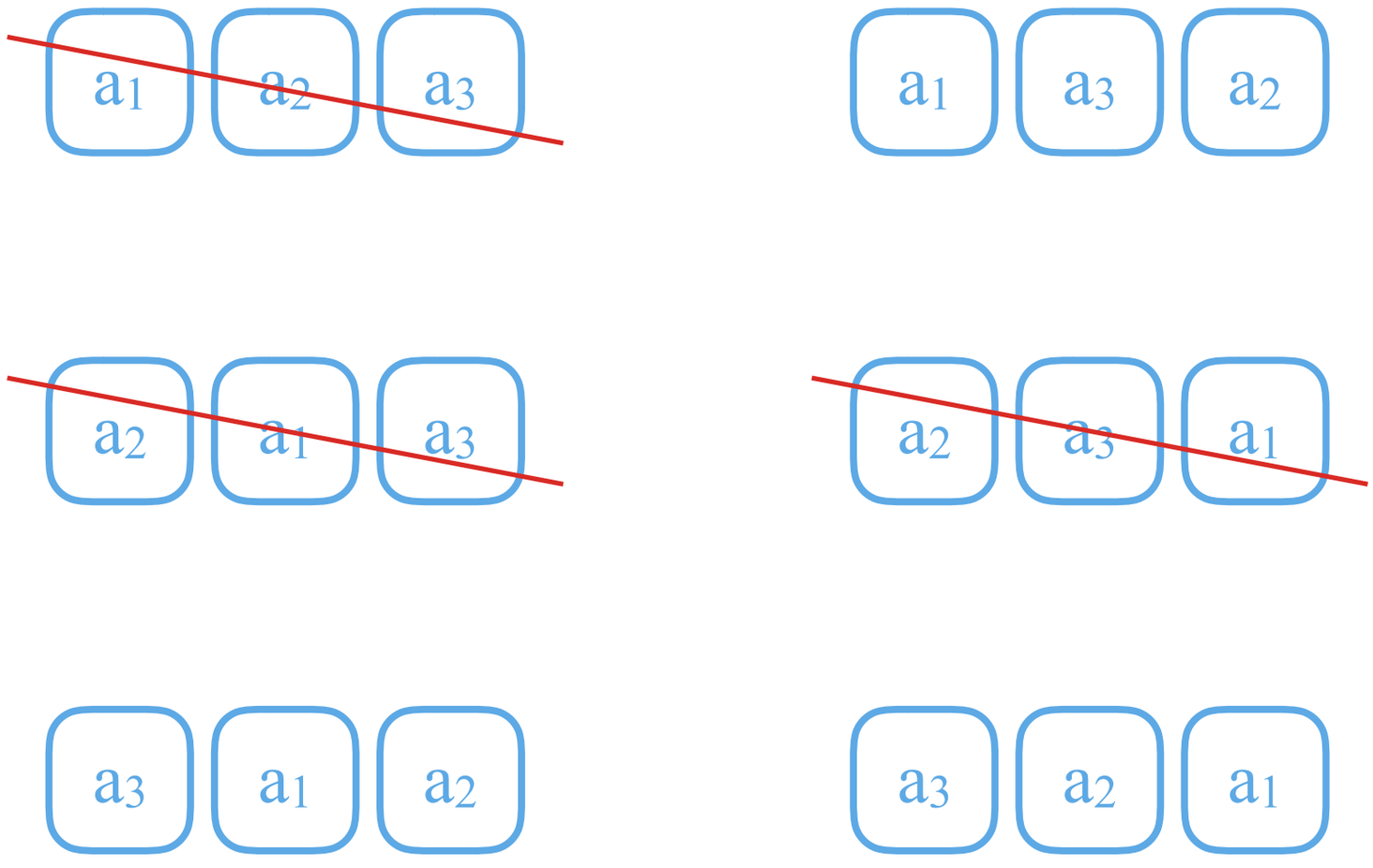
Adversary Game

Alice can decide the outcome of the comparison so that the number of alive permutation is maximized, which is at least a half.



Adversary Game

If more than one permutations are alive, then the sorting algorithm is not yet completed, because Alice can always pick an alive permutation different from the algorithm's output as the correct permutation. // There is exactly one correct permutation.



Adversary Game

Each comparison can reduce the number of alive permutations at most by half, and initially there are n! alive permutations. To reduce the number of alive permutations to 1, any algorithm requires

$$\log_2 n! = \Omega(n \log n) \text{ comparisons}$$

in the comparison-based model.

Merge Sort is asymptotically optimal in the comparison-based model.

Exercise

Prove that log n! = Θ(n log n).

$$\log n! \leq \log n^n = n \log n = O(n \log n)$$

$$\log n! \geq \log \left(\frac{n}{2}\right)^{n/2} = \frac{n}{2} \log \frac{n}{2} = \Omega(n \log n)$$

Summary

(1) There exists an algorithm in the comparison-based model that can solve the sorting problem in $O(n \log n)$ time.

(2) Any algorithm that can solve the sorting problem in the comparison-based model requires $\Omega(n \log n)$ time.

(3) In the comprison-based model, the time complexity of the sorting problem is $\Theta(n \log n)$ due to (1) and (2), and Merge Sort is asymptotically optimal.

Master Thoerem

To solve recurrence relations asymptotically

So far we learn (1) the substitution method and (2) the recursion-tree method.

A third one is *the master theorem*. By (1) and (2), one can prove the master theorem. A proof can be found in I2A.

Master Theorem (I2A pp. 94)

Let a ≥ 1 and b > 1 be constants, let f(n) be a function, and let T(n) be defined on the nonnegative integers by the recurrence

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

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then, T(n) has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ for all sufficiently large n, then $T(n) = \Theta(f(n))$.

Exercise

Solve the following recurrence relations by Master Theorem.

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

$$T(n) = \begin{cases} 3T(\lfloor n/2 \rfloor) + O(n^2) & \text{if } n > 1 \\ O(1) & \text{if } n = 1 \end{cases}$$

Exercise

Input: an array A of n integers and a query x. It is known that there is an index k so that $A[1] < A[2] < ... < A[k]$ and $A[k] > A[k+1] > ... > A[n]$.

Output: "Yes", if $x = A[t]$ for some t in [1, n], or otherwise "No".

Prove or disprove that this problem can be solved in o(n) time in the comparison-based model.

Exercise

Input: an array A of n integers and a query x. It is known that there is an index k so that $A[1] \leq A[2] \leq ... \leq A[k]$ and $A[k] \geq A[k+1] \geq ... \geq A[n]$.

Output: "Yes", if $x = A[t]$ for some t in [1, n], or otherwise "No".

Prove or disprove that this problem can be solved in o(n) time in the comparison-based model.