Lower Bound for Sorting

COMS10017 - (Object-Oriented Programming and) Algorithms

Dr Christian Konrad

Can we sort faster than $O(n \log n)$ time?

Recall: Fastest runtime of any sorting algorithm seen is $O(n \log n)$

Can we sort faster than $O(n \log n)$ time?

Recall: Fastest runtime of any sorting algorithm seen is $O(n \log n)$

Can we sort faster?

Can we sort faster than $O(n \log n)$ time?

Recall: Fastest runtime of any sorting algorithm seen is $O(n \log n)$

Can we sort faster?

Yes! sometimes

Can we sort faster than $O(n \log n)$ time?

Recall: Fastest runtime of any sorting algorithm seen is $O(n \log n)$

Can we sort faster?

Yes! sometimes, but not all algorithms can

Can we sort faster than $O(n \log n)$ time?

Recall: Fastest runtime of any sorting algorithm seen is $O(n \log n)$

Can we sort faster?

Yes! sometimes, but not all algorithms can, and we generally don't know how to . . .

Can we sort faster than $O(n \log n)$ time?

Recall: Fastest runtime of any sorting algorithm seen is $O(n \log n)$

Can we sort faster?

Yes! sometimes, but not all algorithms can, and we generally don't know how to ...

Example: Sort an array $A \in \{0, 1\}^n$ in time $O(n)$?

Can we sort faster than $O(n \log n)$ time?

Recall: Fastest runtime of any sorting algorithm seen is $O(n \log n)$

Can we sort faster?

Yes! sometimes, but not all algorithms can, and we generally don't know how to ...

Example: Sort an array $A \in \{0,1\}^n$ in time $O(n)$?

- Count number of 0s n_0
- Write n_0 0s followed by $n - n_0$ 1s
- Both operations take time $O(n)$

Comparison-based Sorting

Comparison-based Sorting

Comparison-based Sorting

Comparison-based Sorting

- Order is determined solely by comparing input elements

Comparison-based Sorting

Comparison-based Sorting

- Order is determined solely by comparing input elements
- All information obtained is by asking "Is $A[i] \leq A[j]$?", for some i, j, in particular, we may not inspect the elements

Comparison-based Sorting

Comparison-based Sorting

- Order is determined solely by comparing input elements
- All information obtained is by asking "Is $A[i] \le A[j]$?", for some i, j, in particular, we may not inspect the elements
- Quicksort, Mergesort, Insertionsort, Heapsort are comparison-based sorting algorithms

Comparison-based Sorting

Comparison-based Sorting

- Order is determined solely by comparing input elements
- All information obtained is by asking "Is $A[i] \le A[j]$?", for some i, j, in particular, we may not inspect the elements
- Quicksort, Mergesort, Insertionsort, Heapsort are comparison-based sorting algorithms

Lower Bound for Comparison-based Sorting

Comparison-based Sorting

Comparison-based Sorting

- Order is determined solely by comparing input elements
- All information obtained is by asking "Is $A[i] \leq A[j]$?", for some i, j, in particular, we may not inspect the elements
- Quicksort, Mergesort, Insertionsort, Heapsort are comparison-based sorting algorithms

Lower Bound for Comparison-based Sorting

- We will prove that every comparison-based sorting algorithm requires $\Omega(n \log n)$ comparisons

Comparison-based Sorting

- Order is determined solely by comparing input elements
- All information obtained is by asking "Is $A[i] \leq A[j]$?", for some i, j, in particular, we may not inspect the elements
- Quicksort, Mergesort, Insertionsort, Heapsort are comparison-based sorting algorithms

Lower Bound for Comparison-based Sorting

- We will prove that every comparison-based sorting algorithm requires $\Omega(n \log n)$ comparisons
- This implies that $O(n \log n)$ is an optimal runtime for comparison-based sorting

Lower Bound for Comparison-based Sorting

Problem

Lower Bound for Comparison-based Sorting

Problem

- A : array of length n, all elements are different

Lower Bound for Comparison-based Sorting

Problem

- A : array of length n, all elements are different
- We are only allowed to ask: Is $A[i] < A[j]$, for any $i, j \in [n]$

Lower Bound for Comparison-based Sorting

Problem

- A : array of length n, all elements are different
- We are only allowed to ask: Is $A[i] < A[j]$, for any $i, j \in [n]$
- How many questions are needed until we can determine the order of all elements?

Problem

- A : array of length n, all elements are different
- We are only allowed to ask: Is $A[i] < A[j]$, for any $i, j \in [n]$
- How many questions are needed until we can determine the order of all elements?

Permutations

Lower Bound for Comparison-based Sorting

Problem

- A : array of length n, all elements are different
- We are only allowed to ask: Is $A[i] < A[j]$, for any $i, j \in [n]$
- How many questions are needed until we can determine the order of all elements?

Permutations

- A *bijective* function $\pi : [n] \rightarrow [n]$ is called a permutation

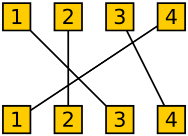
Lower Bound for Comparison-based Sorting

Problem

- A : array of length n, all elements are different
- We are only allowed to ask: Is $A[i] < A[j]$, for any $i, j \in [n]$
- How many questions are needed until we can determine the order of all elements?

Permutations

- A *bijective* function $\pi : [n] \rightarrow [n]$ is called a permutation



$$\begin{aligned} \pi(1) &= 3 \\ \pi(2) &= 2 \\ \pi(3) &= 4 \\ \pi(4) &= 1 \end{aligned}$$

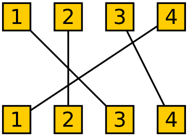
Lower Bound for Comparison-based Sorting

Problem

- A : array of length n, all elements are different
- We are only allowed to ask: Is $A[i] < A[j]$, for any $i, j \in [n]$
- How many questions are needed until we can determine the order of all elements?

Permutations

- A *bijective* function $\pi : [n] \rightarrow [n]$ is called a permutation



$$\begin{aligned} \pi(1) &= 3 \\ \pi(2) &= 2 \\ \pi(3) &= 4 \\ \pi(4) &= 1 \end{aligned}$$

- A reordering of $[n]$

Lower Bound for Comparison-based Sorting (2)

How many permutations are there?

Lower Bound for Comparison-based Sorting (2)

How many permutations are there?

Let Π be the set of all permutations on n elements

Lower Bound for Comparison-based Sorting (2)

How many permutations are there?

Let Π be the set of all permutations on n elements

Lemma

$|\Pi| = n! = n \cdot (n-1) \dots 3 \cdot 2 \cdot 1$

Lower Bound for Comparison-based Sorting (2)

How many permutations are there?
Let Π be the set of all permutations on n elements

Lemma

$|\Pi| = n! = n \cdot (n-1) \ldots 3 \cdot 2 \cdot 1$

Proof.

Lower Bound for Comparison-based Sorting (2)

How many permutations are there?
Let Π be the set of all permutations on n elements

Lemma
$|\Pi| = n! = n \cdot (n-1) \ldots 3 \cdot 2 \cdot 1$

Proof. The first element can be mapped to n potential elements.

Lower Bound for Comparison-based Sorting (2)

How many permutations are there?
Let Π be the set of all permutations on n elements

Lemma
$|\Pi| = n! = n \cdot (n-1) \ldots 3 \cdot 2 \cdot 1$

Proof. The first element can be mapped to n potential elements. The second can only be mapped to $(n-1)$ elements.

Lower Bound for Comparison-based Sorting (2)

How many permutations are there?
Let Π be the set of all permutations on n elements

Lemma
$|\Pi| = n! = n \cdot (n-1) \ldots 3 \cdot 2 \cdot 1$

Proof. The first element can be mapped to n potential elements. The second can only be mapped to $(n-1)$ elements. etc. $\square$

Lower Bound for Comparison-based Sorting (2)

How many permutations are there?

Let Π be the set of all permutations on n elements

Lemma

$|\Pi| = n! = n \cdot (n-1) \ldots 3 \cdot 2 \cdot 1$

Proof. The first element can be mapped to n potential elements. The second can only be mapped to $(n-1)$ elements. etc. $\square$

Rephrasing our Task:

Lower Bound for Comparison-based Sorting (2)

How many permutations are there?

Let Π be the set of all permutations on n elements

Lemma

$|\Pi| = n! = n \cdot (n-1) \ldots 3 \cdot 2 \cdot 1$

Proof. The first element can be mapped to n potential elements. The second can only be mapped to $(n-1)$ elements. etc. $\square$

Rephrasing our Task: Find permutation $\pi \in \Pi$ such that:

$$A[\pi^{-1}(1)] < A[\pi^{-1}(2)] < \cdots < A[\pi^{-1}(n-1)] < A[\pi^{-1}(n)]$$

Decision-tree Model

Example:

Decision-tree Model

Example:

Sort 3 elements by asking queries: $A[i] < A[j]$, for $i, j \in \{0, 1, 2\}$

Decision-tree Model

Example:

Sort 3 elements by asking queries: $A[i] < A[j]$, for $i, j \in \{0, 1, 2\}$

How many Queries are needed? (worst case)

Decision-tree Model

Example:

Sort 3 elements by asking queries: $A[i] < A[j]$, for $i, j \in \{0, 1, 2\}$

How many Queries are needed? (worst case)

Lemma

At least 3 queries are needed to sort 3 elements.

Decision-tree Model

Example:

Sort 3 elements by asking queries: $A[i] < A[j]$, for $i, j \in \{0, 1, 2\}$

How many Queries are needed? (worst case)

Lemma

At least 3 *queries are needed to sort* 3 *elements.*

Proof.

Decision-tree Model

Example:

Sort 3 elements by asking queries: $A[i] < A[j]$, for $i, j \in \{0, 1, 2\}$

How many Queries are needed? (worst case)

Lemma

At least 3 queries are needed to sort 3 elements.

Proof. Let the three elements be a, b, c.

Decision-tree Model

Example:

Sort 3 elements by asking queries: $A[i] < A[j]$, for $i, j \in \{0, 1, 2\}$

How many Queries are needed? (worst case)

Lemma

At least 3 queries are needed to sort 3 elements.

Proof. Let the three elements be a, b, c. Suppose that the first query is $a < b$ and suppose that the answer is yes.

Decision-tree Model

Example:

Sort 3 elements by asking queries: $A[i] < A[j]$, for $i, j \in \{0, 1, 2\}$

How many Queries are needed? (worst case)

Lemma

At least 3 queries are needed to sort 3 elements.

Proof. Let the three elements be a, b, c. Suppose that the first query is $a < b$ and suppose that the answer is yes. (if it is not then relabel the elements a, b, c).

Decision-tree Model

Example:

Sort 3 elements by asking queries: $A[i] < A[j]$, for $i, j \in \{0, 1, 2\}$

How many Queries are needed? (worst case)

Lemma

At least 3 queries are needed to sort 3 elements.

Proof. Let the three elements be a, b, c. Suppose that the first query is $a < b$ and suppose that the answer is yes. (if it is not then relabel the elements a, b, c). We are left with 3 scenarios:

Decision-tree Model

Example:

Sort 3 elements by asking queries: $A[i] < A[j]$, for $i, j \in \{0, 1, 2\}$

How many Queries are needed? (worst case)

Lemma

At least 3 queries are needed to sort 3 elements.

Proof. Let the three elements be a, b, c. Suppose that the first query is $a < b$ and suppose that the answer is yes. (if it is not then relabel the elements a, b, c). We are left with 3 scenarios:

$$1. a < b < c \qquad 2. a < c < b \qquad 3. c < a < b$$

Decision-tree Model

Example:

Sort 3 elements by asking queries: $A[i] < A[j]$, for $i, j \in \{0, 1, 2\}$

How many Queries are needed? (worst case)

Lemma

At least 3 queries are needed to sort 3 elements.

Proof. Let the three elements be a, b, c. Suppose that the first query is $a < b$ and suppose that the answer is yes. (if it is not then relabel the elements a, b, c). We are left with 3 scenarios:

$$1. a < b < c \qquad 2. a < c < b \qquad 3. c < a < b$$

Next we either ask $a < c$ or $b < c$.

Decision-tree Model

Example:

Sort 3 elements by asking queries: $A[i] < A[j]$, for $i, j \in \{0, 1, 2\}$

How many Queries are needed? (worst case)

Lemma

At least 3 queries are needed to sort 3 elements.

Proof. Let the three elements be a, b, c. Suppose that the first query is $a < b$ and suppose that the answer is yes. (if it is not then relabel the elements a, b, c). We are left with 3 scenarios:

$$1. a < b < c \qquad 2. a < c < b \qquad 3. c < a < b$$

Next we either ask $a < c$ or $b < c$. Suppose that we ask $a < c$.

Decision-tree Model

Example:

Sort 3 elements by asking queries: $A[i] < A[j]$, for $i, j \in \{0, 1, 2\}$

How many Queries are needed? (worst case)

Lemma

At least 3 queries are needed to sort 3 elements.

Proof. Let the three elements be a, b, c. Suppose that the first query is $a < b$ and suppose that the answer is yes. (if it is not then relabel the elements a, b, c). We are left with 3 scenarios:

$$1. a < b < c \qquad 2. a < c < b \qquad 3. c < a < b$$

Next we either ask $a < c$ or $b < c$. Suppose that we ask $a < c$. Then, if the answer is yes then we are left with cases 1 and 2 and we need an additional query.

Decision-tree Model

Example:

Sort 3 elements by asking queries: $A[i] < A[j]$, for $i, j \in \{0, 1, 2\}$

How many Queries are needed? (worst case)

Lemma

At least 3 queries are needed to sort 3 elements.

Proof. Let the three elements be a, b, c. Suppose that the first query is $a < b$ and suppose that the answer is yes. (if it is not then relabel the elements a, b, c). We are left with 3 scenarios:

$$1.a < b < c \qquad 2.a < c < b \qquad 3.c < a < b$$

Next we either ask $a < c$ or $b < c$. Suppose that we ask $a < c$. Then, if the answer is yes then we are left with cases 1 and 2 and we need an additional query. Suppose that we ask $b < c$.

Decision-tree Model

Example:

Sort 3 elements by asking queries: $A[i] < A[j]$, for $i, j \in \{0, 1, 2\}$

How many Queries are needed? (worst case)

Lemma

At least 3 queries are needed to sort 3 elements.

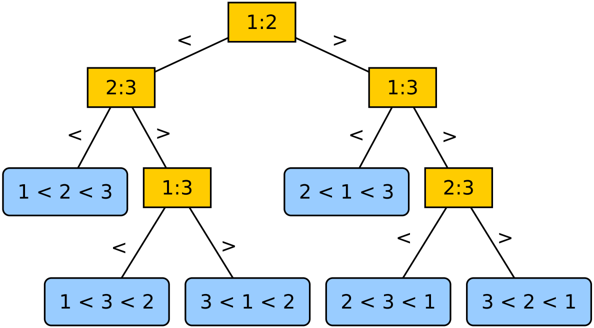
Proof. Let the three elements be a, b, c. Suppose that the first query is $a < b$ and suppose that the answer is yes. (if it is not then relabel the elements a, b, c). We are left with 3 scenarios:

$$1. a < b < c \qquad 2. a < c < b \qquad 3. c < a < b$$

Next we either ask $a < c$ or $b < c$. Suppose that we ask $a < c$. Then, if the answer is yes then we are left with cases 1 and 2 and we need an additional query. Suppose that we ask $b < c$. Then, if the answer is no then we are left with cases 2 and 3 and we need an additional query. $\square$

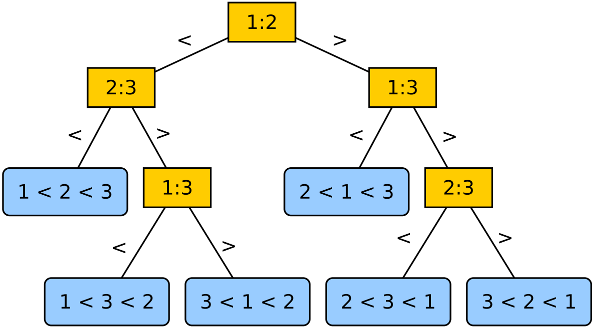
Decision-tree Model (2)

Every Guessing Strategy (and Sorting Algorithm) is a Decision-tree



Decision-tree Model (2)

Every Guessing Strategy (and Sorting Algorithm) is a Decision-tree

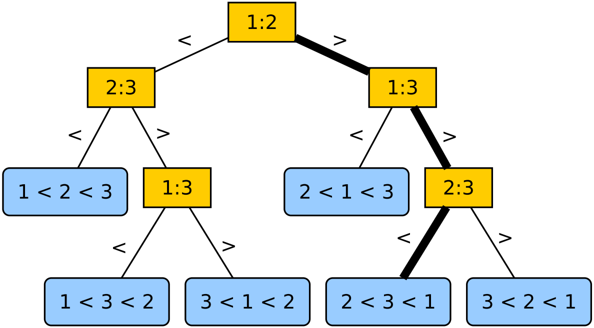


Observe:

- Every leaf is a permutation

Decision-tree Model (2)

Every Guessing Strategy (and Sorting Algorithm) is a Decision-tree



Observe:

- Every leaf is a permutation
- An execution is a root-to-leaf path

Sorting Lower Bound

Sorting Lower Bound

Lemma

Any comparison-based sorting algorithm requires $\Omega(n \log n)$ comparisons.

Sorting Lower Bound

Lemma

Any comparison-based sorting algorithm requires $\Omega(n \log n)$ comparisons.

Proof

Sorting Lower Bound

Lemma

Any comparison-based sorting algorithm requires $\Omega(n \log n)$ comparisons.

Proof Observe that decision-tree is a binary tree.

Sorting Lower Bound

Lemma

Any comparison-based sorting algorithm requires $\Omega(n \log n)$ comparisons.

Proof Observe that decision-tree is a binary tree. Every potential permutation is a leaf.

Sorting Lower Bound

Lemma

Any comparison-based sorting algorithm requires $\Omega(n \log n)$ comparisons.

Proof Observe that decision-tree is a binary tree. Every potential permutation is a leaf. There are $n!$ leaves.

Sorting Lower Bound

Lemma

Any comparison-based sorting algorithm requires $\Omega(n \log n)$ *comparisons.*

Proof Observe that decision-tree is a binary tree. Every potential permutation is a leaf. There are $n!$ leaves. A binary tree of height h has no more than 2^h leaves.

Sorting Lower Bound

Lemma

Any comparison-based sorting algorithm requires $\Omega(n \log n)$ comparisons.

Proof Observe that decision-tree is a binary tree. Every potential permutation is a leaf. There are $n!$ leaves. A binary tree of height h has no more than 2^h leaves. Hence:

$$2^h \geq n!$$

Sorting Lower Bound

Lemma

Any comparison-based sorting algorithm requires $\Omega(n \log n)$ comparisons.

Proof Observe that decision-tree is a binary tree. Every potential permutation is a leaf. There are $n!$ leaves. A binary tree of height h has no more than 2^h leaves. Hence:

$$\begin{aligned} 2^h &\geq n! \\ h &\geq \log(n!) \end{aligned}$$

Sorting Lower Bound

Lemma

Any comparison-based sorting algorithm requires $\Omega(n \log n)$ comparisons.

Proof Observe that decision-tree is a binary tree. Every potential permutation is a leaf. There are $n!$ leaves. A binary tree of height h has no more than 2^h leaves. Hence:

$$\begin{aligned} 2^h &\geq n! \\ h &\geq \log(n!) \geq \log\left((\frac{n}{e})^n\right) \end{aligned}$$

Sorting Lower Bound

Lemma

Any comparison-based sorting algorithm requires $\Omega(n \log n)$ comparisons.

Proof Observe that decision-tree is a binary tree. Every potential permutation is a leaf. There are $n!$ leaves. A binary tree of height h has no more than 2^h leaves. Hence:

$$
\begin{aligned}
2^h &\geq n! \\
h &\geq \log(n!) \geq \log\left((\frac{n}{e})^n\right) = n\log(\frac{n}{e})
\end{aligned}
$$

Sorting Lower Bound

Lemma

Any comparison-based sorting algorithm requires $\Omega(n \log n)$ comparisons.

Proof Observe that decision-tree is a binary tree. Every potential permutation is a leaf. There are $n!$ leaves. A binary tree of height h has no more than 2^h leaves. Hence:

$$\begin{aligned} 2^h &\geq n! \\ h &\geq \log(n!) \geq \log\left((\frac{n}{e})^n\right) = n \log(\frac{n}{e}) = \Omega(n \log n) \ . \end{aligned}$$

$\square$

Stirling's approximation: $n! \geq \left(\frac{n}{e}\right)^n$