Lecture 12: Lower Bound for Sorting, Countingsort, Radixsort

COMS10007 - Algorithms

Dr. Christian Konrad

12.03.2019

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?

- For example in $O(n \log \log n)$ time?
- Or even $O(n)$ time?

Yes! we can sometimes sort faster
But in general, **no**, we cannot

Example: Sort an array of length n of bits, i.e., every array element is either 0 or 1, 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

- Order is determined solely by comparing input elements
- All information we obtain 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
- Algorithm on last slide can be turned into a comparison-based algorithm. How? (restricted domain)

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

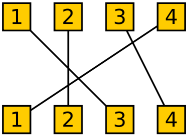
Lower Bound for Comparison-based Sorting

Problem

- 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?
Let Π be the set of all permutations on n elements

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

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

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

$$A[\pi(1)] < A[\pi(2)] < \dots < A[\pi(n-1)] < A[\pi(n)]$$

Decision-tree Model

Example:

Sort 3 elements by asking queries: $A[i] < A[j]$, for $i, j \in [3]$

How many Queries are needed? (worst case)

Lemma

At least 3 queries are needed to sort 3 elements.

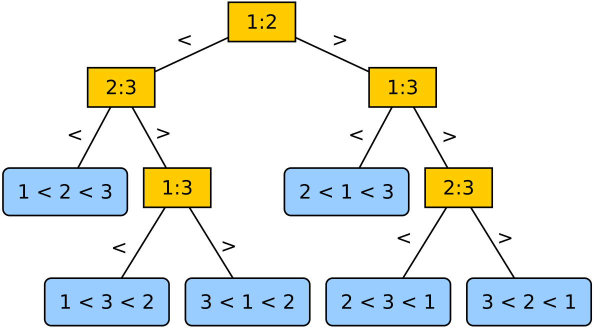
Proof. Let the three elements be a, b, c. Suppose that the first query is $a < b$ and suppose that the answer is yes. (if it is not then relabel the elements a, b, c). We are left with 3 scenarios:

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

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. □

Decision-tree Model (2)

Every Guessing Strategy is a Decision-tree

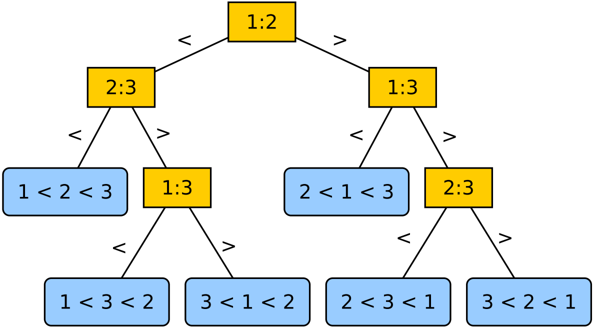


Observe:

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

Decision-tree Model (2)

Every Guessing Strategy is a Decision-tree

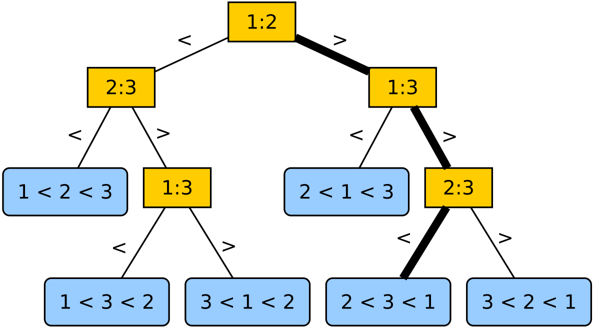


Observe:

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

Decision-tree Model (2)

Every Guessing Strategy is a Decision-tree



Observe:

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

Sorting Lower Bound

Lemma

Any comparision-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:

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

□

Comment: Stirling's approximation for $n!$ can be used for proving $\log(n!) = \Omega(n \log n)$

Counting Sort: Sorting Integers fast

Counting Sort
Input is an array A of integers from $\{0, 1, 2, \dots, k\}$, for some integer k

Idea

- For each element x, count number of elements $< x$
- Put x directly into its position
- **Difficulty:** Multiple elements have the same value

Algorithm

Require: Array A of n integers from $\{0, 1, 2, \ldots, k\}$, for some integer k
Let $C[0 \ldots k]$ be a new array with all entries equal to 0
Store output in array $B[0 \ldots n-1]$

for $i = 0, \ldots, n-1$ **do** {Count how often each element appears}
    $C[A[i]] \leftarrow C[A[i]] + 1$
for $i = 1, \ldots, k$ **do** {Count how many smaller elements appear}
    $C[i] \leftarrow C[i] + C[i-1]$
for $i = n-1, \ldots, 0$ **do**
    $B[C[A[i]] - 1] \leftarrow A[i]$
    $C[A[i]] \leftarrow C[A[i]] - 1$
return m

- Last loop processes A from right to left
- $C[A[i]]$: Number of *smaller* elements than $A[i]$
- Decrementing $C[A[i]]$: Next element of value $A[i]$ should be left of the current one

Counting Sort: Example

Example: $n = 8$, $k = 5$

	0	1	2	3	4	5	6	7
A	2	5	3	0	2	3	0	3

Counting Sort: Example

Example: $n = 8$, $k = 5$

	0	1	2	3	4	5	6	7
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

Counting Sort: Example

Example: $n = 8$, $k = 5$

	0	1	2	3	4	5	6	7
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

	0	1	2	3	4	5
C	2	2	4	7	7	8

	0	1	2	3	4	5	6	7
B								

for $i = n-1, \dots, 0$ **do**
  $B[C[A[i]] - 1] \leftarrow A[i]$
  $C[A[i]] \leftarrow C[A[i]] - 1$

Counting Sort: Example

Example: $n = 8$, $k = 5$

	0	1	2	3	4	5	6	7
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

	0	1	2	3	4	5
C	2	2	4	7	7	8

	0	1	2	3	4	5	6	7
B							3	

for $i = n-1, \dots, 0$ **do**
$B[C[A[i]] - 1] \leftarrow A[i]$
$C[A[i]] \leftarrow C[A[i]] - 1$

Counting Sort: Example

Example: $n = 8$, $k = 5$

	0	1	2	3	4	5	6	7
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

	0	1	2	3	4	5
C	2	2	4	6	7	8

	0	1	2	3	4	5	6	7
B							3	

for $i = n - 1, \dots, 0$ **do**
  $B[C[A[i]] - 1] \leftarrow A[i]$
  $C[A[i]] \leftarrow C[A[i]] - 1$

Example: $n = 8$, $k = 5$

	0	1	2	3	4	5	6	7
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

	0	1	2	3	4	5
C	2	2	4	6	7	8

	0	1	2	3	4	5	6	7
B		0					3	

for $i = n-1, \dots, 0$ **do**
  $B[C[A[i]] - 1] \leftarrow A[i]$
  $C[A[i]] \leftarrow C[A[i]] - 1$

Counting Sort: Example

Example: $n = 8$, $k = 5$

A

0	1	2	3	4	5	6	7
2	5	3	0	2	3	0	3

C

0	1	2	3	4	5
2	0	2	3	0	1

C

0	1	2	3	4	5
1	2	4	6	7	8

B

0	1	2	3	4	5	6	7
	0					3	

for $i = n-1, \ldots, 0$ **do**
$B[C[A[i]] - 1] \leftarrow A[i]$
$C[A[i]] \leftarrow C[A[i]] - 1$

Counting Sort: Example

Example: $n = 8$, $k = 5$

	0	1	2	3	4	5	6	7
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

	0	1	2	3	4	5
C	1	2	4	6	7	8

	0	1	2	3	4	5	6	7
B		0				3	3	

for $i = n-1, \dots, 0$ **do**
  $B[C[A[i]] - 1] \leftarrow A[i]$
  $C[A[i]] \leftarrow C[A[i]] - 1$

Counting Sort: Example

Example: $n = 8$, $k = 5$

	0	1	2	3	4	5	6	7
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

	0	1	2	3	4	5
C	1	2	4	5	7	8

	0	1	2	3	4	5	6	7
B		0				3	3	

for $i = n-1, \dots, 0$ **do**
$B[C[A[i]] - 1] \leftarrow A[i]$
$C[A[i]] \leftarrow C[A[i]] - 1$

Counting Sort: Example

Example: $n = 8$, $k = 5$

	0	1	2	3	4	5	6	7
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

	0	1	2	3	4	5
C	1	2	4	5	7	8

	0	1	2	3	4	5	6	7
B		0		2		3	3	

for $i = n-1, \dots, 0$ **do**
  $B[C[A[i]] - 1] \leftarrow A[i]$
  $C[A[i]] \leftarrow C[A[i]] - 1$

Counting Sort: Example

Example: $n = 8$, $k = 5$

	0	1	2	3	4	5	6	7
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

	0	1	2	3	4	5
C	1	2	3	5	7	8

	0	1	2	3	4	5	6	7
B		0		2		3	3	

for $i = n-1, \dots, 0$ **do**
  $B[C[A[i]]-1] \leftarrow A[i]$
  $C[A[i]] \leftarrow C[A[i]] - 1$

Counting Sort: Example

Example: $n = 8$, $k = 5$

	0	1	2	3	4	5	6	7
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

	0	1	2	3	4	5
C	1	2	3	5	7	8

	0	1	2	3	4	5	6	7
B	0	0		2		3	3	

for $i = n - 1, \dots, 0$ **do**
$B[C[A[i]] - 1] \leftarrow A[i]$
$C[A[i]] \leftarrow C[A[i]] - 1$

Counting Sort: Example

Example: $n = 8$, $k = 5$

	0	1	2	3	4	5	6	7
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

	0	1	2	3	4	5
C	0	2	3	5	7	8

	0	1	2	3	4	5	6	7
B	0	0		2		3	3	

for $i = n-1, \dots, 0$ **do**
$B[C[A[i]] - 1] \leftarrow A[i]$
$C[A[i]] \leftarrow C[A[i]] - 1$

Counting Sort: Example

Example: $n = 8$, $k = 5$

	0	1	2	3	4	5	6	7
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

	0	1	2	3	4	5
C	0	2	2	4	7	7

	0	1	2	3	4	5	6	7
B	0	0	2	2	3	3	3	5

for $i = n-1, \dots, 0$ **do**
  $B[C[A[i]] - 1] \leftarrow A[i]$
  $C[A[i]] \leftarrow C[A[i]] - 1$

Analysis: Counting Sort

Runtime:

$O(n) + O(k) + O(n) = O(n + k)$

- Counting Sort has runtime $O(n)$ if $k = O(n)$
- This beats the lower bound for comparison-based sorting

```
for i = 0, ..., n − 1 do
    C[A[i]] ← C[A[i]] + 1
for i = 1, ..., k do
    C[i] ← C[i] + C[i − 1]
for i = n − 1, ..., 0 do
    B[C[A[i]] − 1] ← A[i]
    C[A[i]] ← C[A[i]] − 1
```

Stable? In-place? Yes, it is stable (important!) No, not in-place

Correctness Loop Invariant

Radix Sort

Input is an array A of d digits integers, each digit is from the set $\{0, 1, \ldots, b-1\}$

Examples

- $b = 2$, $d = 5$. E.g. 01101 (binary numbers)
- $b = 10$, $d = 4$. E.g. 9714

Idea

- Iterate through the d digits
- Sort according to the current digit

Radix Sort (2)

Radix Sort Algorithm

> **for** $i = 1, \dots, d$ **do**
> Use a stable sort algorithm to sort array A on digit i

(least significant digit is digit 1)

Example

329		72**0**		7**2**0		**3**29
457		35**5**		3**2**9		**3**55
657		43**6**		4**3**6		**4**36
839	$\rightarrow$	45**7**	$\rightarrow$	8**3**9	$\rightarrow$	**4**57
436		65**7**		3**5**5		**6**57
720		32**9**		4**5**7		**7**20
355		83**9**		6**5**7		**8**39

Radix Sort (3)

Analysis

Lemma

Given n d-digit number in which each digit can take on up to b possible values. Radix-sort correctly sorts these numbers in $O(d(n+b))$ time if the stable sort it uses takes $O(n+b)$ time.

Proof Runtime is obvious. Correctness follows by induction on the columns being sorted. □

Observe: If $d = O(1)$ and $b = O(n)$ then the runtime is $O(n)$!