Sorting in Linear Time

Prepared by Suk Jin Lee

Sorting So Far

- Insertion sort:
  - Easy to code
  - Fast on small inputs (less than ~50 elements)
  - Fast on nearly-sorted inputs
  - $O(n^2)$ worst case
  - $O(n^2)$ average (equally-likely inputs) case
  - $O(n^2)$ reverse-sorted case

Sorting So Far

- Merge sort:
  - Divide-and-conquer:
    - Split array in half
    - Recursively sort subarrays
    - Linear-time merge step
  - O(n lg n) worst case
  - Doesn’t sort in place

Sorting So Far

- Heapsort:
  - Uses the very useful heap data structure
    - Complete binary tree
    - Heap property: parent key > children's keys
  - O(n lg n) worst case
  - Sorts in place
  - Fair amount of shuffling memory around

Sorting So Far

- Quicksort:
  - Divide-and-conquer:
    - Partition array into two subarrays, recursively sort
    - All of first subarray < all of second subarray
    - No merge step needed!
  - O(n lg n) average case
  - Fast in practice
  - O(n^2) worst case
    - Naïve implementation: worst case on sorted input
    - Address this with randomized quicksort

How Fast Can We Sort?

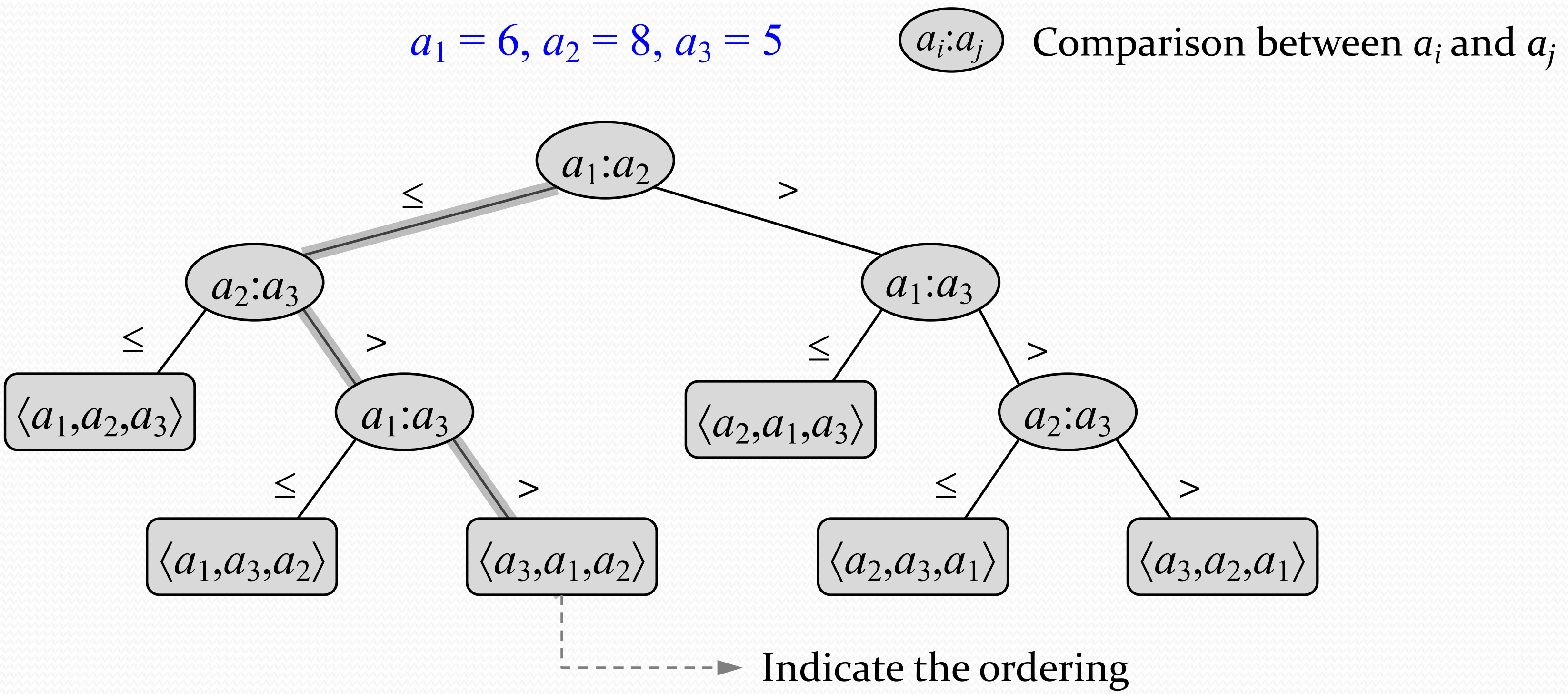
- We will provide a lower bound, then beat it by playing a different game
  - *How do you suppose we'll beat it?*
- First, an observation: all of the sorting algorithms so far are *comparison sorts*
  - The only operation used to gain ordering information about a sequence is the pairwise comparison of two elements
  - All sorts seen so far are comparison sorts: insertion sort, selection sort, merge sort, quicksort, heapsort

Decision Trees

- *Decision trees* provides an abstraction of comparison sorts
  - A decision tree represents the comparisons made by a comparison sort. Every thing else ignored
  - *What do each internal node represent?*
  - *What do the leaves represent?*
  - *How many leaves must there be?*

Decision Trees

- *Decision tree* for insertion sort operating on three elements



Each leaf must be reachable from the root by a downward path

Decision Trees

- Decision trees can model comparison sorts.
  For a given algorithm:
  - One tree for each *n*
  - Tree paths are all possible execution traces
  - What's the longest path in a decision tree for insertion sort? For merge sort?
- *What is the asymptotic height of any decision tree for sorting n elements?*
  - Answer: $\Omega(n \lg n)$

Lower bounds for Sorting

- ***Theorem***. Any decision tree to sort n elements requires $\Omega(n \lg n)$ comparisons in the worst case
- *What's the minimum # of leaves?*
- *What's the maximum # of leaves of a binary tree of height h?*

Lower bounds for Sorting

- ***Theorem***. Any decision tree to sort n elements requires $\Omega(n \lg n)$ comparisons in the worst case
- *What's the minimum # of leaves?*
  - Answer: $n!$
- *What's the maximum # of leaves of a binary tree of height h?*

Lower bounds for Sorting

- ***Theorem***. Any decision tree to sort n elements requires $\Omega(n \lg n)$ comparisons in the worst case
- *What's the minimum # of leaves?*
  - Answer: $n!$
- *What's the maximum # of leaves of a binary tree of height h?*
  - Answer: 2^h
- Clearly the minimum # of leaves is less than or equal to the maximum # of leaves

Lower bounds for Sorting

- ***Theorem***. Any decision tree to sort n elements requires $\Omega(n \lg n)$ comparisons in the worst case
- *Proof*. The tree must contain $\geq n!$ leaves, since there are $n!$ possible permutations. A height-h binary tree has no more than 2^h leaves. Thus, $n! \leq 2^h$

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

Thus the minimum height of a decision tree is $\Omega(n \lg n)$

Lower bounds for Sorting

- Thus the time to comparison sort n elements is $\Omega(n \lg n)$

Lower bounds for Sorting

- Thus the time to comparison sort *n* elements is $\Omega(n \lg n)$
- ***Corollary***. Heapsort and merge sort are asymptotically optimal comparison sorts.
- *Proof*. The $O(n \lg n)$ upper bounds on the running times for heapsort and merge sort match the $\Omega(n \lg n)$ worst-case lower bound from Theorem □
- *How can we do better than* $\Omega(n \lg n)$ *?*

Counting Sort

Prepared by Suk Jin Lee

Sorting in linear time

- Counting sort:
  - No comparisons between elements
  - Input: $A[1 \,..\, n]$, where $A[j] \in \{1, 2, \ldots, k\}$
  - Output: $B[1 \,..\, n]$, sorted
  - Auxiliary storage: $C[1 \,..\, k]$

Counting Sort

- COUNTING-SORT(A, B, k)

```
Let C[0 . . k] be a new array
for i = 0 to k
    C[i] ← 0
for j = 1 to A.length
    C[A[j]] ← C[A[j]] + 1
// C[i] now contains the number of elements equal to i
for i = 1 to k
    C[i] ← C[i] + C[i – 1]
// C[i] now contains the number of elements less than or equal to i
for j = A.length downto 1
    B[C[A[j]]] ← A[j]        // C[A[j]] is the correct final position of A[j]
    C[A[j]] ← C[A[j]] – 1
```

Counting Sort

- COUNTING-SORT(A, B, k)

Let $C[0\,..\,k]$ be a new array
for $i = 0$ to k $\qquad \Theta(k)$
    $C[i] \leftarrow 0$
for $j = 1$ to $A.length$ $\qquad \Theta(n)$
    $C[A[j]] \leftarrow C[A[j]] + 1$
// $C[i]$ now contains the number of elements equal to i
for $i = 1$ to k $\qquad \Theta(k)$
    $C[i] \leftarrow C[i] + C[i - 1]$
// $C[i]$ now contains the number of elements less than or equal to i
for $j = A.length$ **downto** 1 $\qquad \Theta(n)$
    $B[C[A[j]]] \leftarrow A[j]$ $\qquad$ // $C[A[j]]$ is the correct final position of $A[j]$
    $C[A[j]] \leftarrow C[A[j]] - 1$

Counting Sort - Example

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

	0	1	2	3	4	5
C:						

B:								

Counting Sort - Example

- Loop 1

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

i:	0	1	2	3	4	5
C:	0	0	0	0	0	0

B:								

for $i = 0$ to k

$C[i] \leftarrow 0$

Counting Sort - Example

- Loop 2

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	3	0	3

i:	0	1	2	3	4	5
C:	0	0	1	0	0	0

B:								

for $j = 1$ to $A.length$

$C[A[j]] \leftarrow C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

Counting Sort - Example

- Loop 2

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	3	0	3

i:	0	1	2	3	4	5
C:	0	0	1	0	0	1

B:								

for $j = 1$ to $A.length$

$C[A[j]] \leftarrow C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

Counting Sort - Example

- Loop 2

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	3	0	3
B:								

i:	0	1	2	3	4	5
C:	0	0	1	1	0	1

for $j = 1$ to $A.length$

$C[A[j]] \leftarrow C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

Counting Sort - Example

- Loop 2

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	3	0	3

i:	0	1	2	3	4	5
C:	1	0	1	1	0	1

B:								

for $j = 1$ to $A.length$

$C[A[j]] \leftarrow C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

Counting Sort - Example

- Loop 2

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	3	0	3

i:	0	1	2	3	4	5
C:	1	0	2	1	0	1

B:								

for $j = 1$ to $A.length$

$C[A[j]] \leftarrow C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

Counting Sort - Example

- Loop 2

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	3	0	3

i:	0	1	2	3	4	5
C:	1	0	2	2	0	1

B:								

for $j = 1$ to $A.length$

$C[A[j]] \leftarrow C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

Counting Sort - Example

- Loop 2

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	3	0	3

i:	0	1	2	3	4	5
C:	2	0	2	2	0	1

B:								

for $j = 1$ to $A.length$

$C[A[j]] \leftarrow C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

Counting Sort - Example

- Loop 2

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	3	0	3

i:	0	1	2	3	4	5
C:	2	0	2	3	0	1

B:								

for $j = 1$ to $A.length$

$C[A[j]] \leftarrow C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

Counting Sort - Example

- Loop 3

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	3	0	3
B:								

i:	0	1	2	3	4	5
C:	2	0	2	3	0	1

i:	0	1	2	3	4	5
C':	2	2	2	3	0	1

for $i = 1$ to k

$C[i] \leftarrow C[i] + C[i-1]$

// $C[i]$ now contains the number of elements less than or equal to i

Counting Sort - Example

- Loop 3

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	3	0	3
B:								

i:	0	1	2	3	4	5
C:	2	0	2	3	0	1

i:	0	1	2	3	4	5
C':	2	2	$\boxed{4}$	3	0	1

for $i = 1$ to k

$$C[i] \leftarrow C[i] + C[i-1]$$

// $C[i]$ now contains the number of elements less than or equal to i

Counting Sort - Example

- Loop 3

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	3	0	3
B:								

i:	0	1	2	3	4	5
C:	2	0	2	3	0	1

i:	0	1	2	3	4	5
C':	2	2	4	7	0	1

for $i = 1$ to k

$C[i] \leftarrow C[i] + C[i-1]$

// $C[i]$ now contains the number of elements less than or equal to i

Counting Sort - Example

- Loop 3

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	3	0	3

B:								

i:	0	1	2	3	4	5
C:	2	0	2	3	0	1

i:	0	1	2	3	4	5
C′:	2	2	4	7	$\boxed{7}$	1

for $i = 1$ to k

$$C[i] \leftarrow C[i] + C[i-1]$$

// $C[i]$ now contains the number of elements less than or equal to i

Counting Sort - Example

- Loop 3

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	3	0	3

i:	0	1	2	3	4	5
C:	2	0	2	3	0	1

i:	0	1	2	3	4	5
C':	2	2	4	7	7	8

B:								

for $i = 1$ to k

$$C[i] \leftarrow C[i] + C[i-1]$$

// $C[i]$ now contains the number of elements less than or equal to i

Counting Sort - Example

- Loop 4

$A[j]$

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	3	0	3

i:	0	1	2	3	4	5
C:	2	0	2	3	0	1

	1	2	3	4	5	6	7	8
B:							3	

i:	0	1	2	3	4	5
C':	2	2	4	6	7	8

$C[A[j]]$

for $j = A.length$ **downto** 1

$B[C[A[j]]] \leftarrow A[j]$ // $C[A[j]]$ is the correct final position of $A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

Counting Sort - Example

- Loop 4

$A[j]$

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	3	0	3

i:	0	1	2	3	4	5
C:	2	0	2	3	0	1

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

i:	0	1	2	3	4	5
C':	1	2	4	6	7	8

$C[A[j]]$

for $j = A.length$ **downto** 1

$B[C[A[j]]] \leftarrow A[j]$ // $C[A[j]]$ is the correct final position of $A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

Counting Sort - Example

- Loop 4

$A[j]$

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	$\boxed{3}$	0	3

	1	2	3	4	5	6	7	8
B:		0				$\boxed{3}$	3	

i:	0	1	2	3	4	5
C:	2	0	2	3	0	1

i:	0	1	2	3	4	5
C':	1	2	4	$\boxed{5}$	7	8

$C[A[j]]$

for $j = A.length$ **downto** 1

$B[C[A[j]]] \leftarrow A[j]$ // $C[A[j]]$ is the correct final position of $A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

Counting Sort - Example

- Loop 4

$A[j]$ (at $j = 5$)

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	$\boxed{2}$	3	0	3

i:	0	1	2	3	4	5
C:	2	0	2	3	0	1

	1	2	3	4	5	6	7	8
B:		0		$\boxed{2}$		3	3	

i:	0	1	2	3	4	5
C':	1	2	$\boxed{3}$	5	7	8

$C[A[j]]$ (at $i = 2$)

for $j = A.length$ **downto** 1

$B[C[A[j]]] \leftarrow A[j]$ // $C[A[j]]$ is the correct final position of $A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

Counting Sort - Example

- Loop 4

$A[j]$

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	3	0	3

i:	0	1	2	3	4	5
C:	2	0	2	3	0	1

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

i:	0	1	2	3	4	5
C':	0	2	3	5	7	8

$C[A[j]]$

for $j = A.length$ **downto** 1

$B[C[A[j]]] \leftarrow A[j]$ // $C[A[j]]$ is the correct final position of $A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

Counting Sort - Example

- Loop 4

$A[j]$

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	3	0	3

i:	0	1	2	3	4	5
C:	2	0	2	3	0	1

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

i:	0	1	2	3	4	5
C':	0	2	3	4	7	8

$C[A[j]]$

for $j = A.length$ **downto** 1

$B[C[A[j]]] \leftarrow A[j]$ // $C[A[j]]$ is the correct final position of $A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

Counting Sort - Example

- Loop 4

$A[j]$

j:	1	2	3	4	5	6	7	8
A:	2	5	3	0	2	3	0	3

i:	0	1	2	3	4	5
C:	2	0	2	3	0	1

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

i:	0	1	2	3	4	5
C':	0	2	3	4	7	7

$C[A[j]]$

for $j = A.length$ **downto** 1

$B[C[A[j]]] \leftarrow A[j]$ // $C[A[j]]$ is the correct final position of $A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

Counting Sort - Example

- Loop 4

$A[j]$

j:	1	2	3	4	5	6	7	8
A:	$\boxed{2}$	5	3	0	2	3	0	3

i:	0	1	2	3	4	5
C:	2	0	2	3	0	1

	1	2	3	4	5	6	7	8
B:	0	0	$\boxed{2}$	2	3	3	3	5

i:	0	1	2	3	4	5
C':	0	2	$\boxed{2}$	4	7	7

$C[A[j]]$

for $j = A.length$ **downto** 1

$B[C[A[j]]] \leftarrow A[j]$ // $C[A[j]]$ is the correct final position of $A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

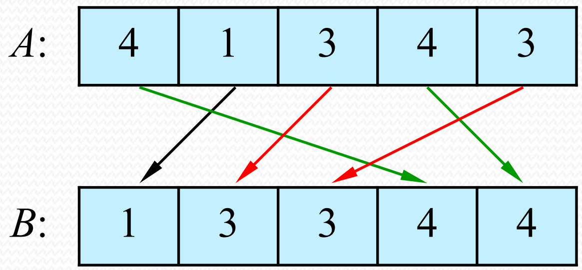
Analysis

- COUNTING-SORT(A, B, k)

for $i = 0$ to k	$\Theta(k)$
$C[i] \leftarrow 0$	
for $j = 1$ to $A.length$	$\Theta(n)$
$C[A[j]] \leftarrow C[A[j]] + 1$	
for $i = 1$ to k	$\Theta(k)$
$C[i] \leftarrow C[i] + C[i - 1]$	
for $j = A.length$ **downto** 1	$\Theta(n)$
$B[C[A[j]]] \leftarrow A[j]$	
$C[A[j]] \leftarrow C[A[j]] - 1$	
	$\Theta(n + k)$

Running time

- If $k = O(n)$, then counting sort takes $\Theta(n)$ time.
  - Counting sort beats the lower bound of $\Theta(n \lg n)$ comparison sort
  - Counting sort is not a comparison sort
- Stable sorting
  - Counting sort is a ***stable*** sort: it preserves the input order among equal elements.



Counting Sort

- ***Cool!***
- *Why don't we always use counting sort?*
  - Because it depends on range *k* of elements
- *Could we use counting sort to sort 32 bit integers? Why or why not?*
  - Answer: no, *k* too large (2^{32} = 4,294,967,296)

Quiz

- Using the previous figure model, illustrate the operation of COUNT-SORT on the array $A = \langle 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2 \rangle$

Quiz

- Using the previous figure model, illustrate the operation of COUNT-SORT on the array $A = \langle 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2 \rangle$

j:	1	2	3	4	5	6	7	8	9	10	11
A:	6	0	2	0	1	3	4	6	1	3	2

Quiz

- Using the previous figure model, illustrate the operation of COUNT-SORT on the array $A = \langle 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2 \rangle$

j:	1	2	3	4	5	6	7	8	9	10	11
A:	6	0	2	0	1	3	4	6	1	3	2

i:	0	1	2	3	4	5	6
C:	2	2	2	2	1	0	2

Quiz

- Using the previous figure model, illustrate the operation of COUNT-SORT on the array $A = \langle 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2 \rangle$

j:	1	2	3	4	5	6	7	8	9	10	11
A:	6	0	2	0	1	3	4	6	1	3	2

i:	0	1	2	3	4	5	6
C:	2	2	2	2	1	0	2

i:	0	1	2	3	4	5	6
C';	2	4	6	8	9	9	11

Quiz

- Using the previous figure model, illustrate the operation of COUNT-SORT on the array $A = \langle 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2 \rangle$

j:	1	2	3	4	5	6	7	8	9	10	11
A:	6	0	2	0	1	3	4	6	1	3	2

i:	0	1	2	3	4	5	6
C:	2	2	2	2	1	0	2

	1	2	3	4	5	6	7	8	9	10	11
B:	0	0	1	1	2	2	3	3	4	6	6

i:	0	1	2	3	4	5	6
C';	2	4	6	8	9	9	11