CSCI 241

Lecture ~~6~~ 7

~~Quicksort~~

Stability; Non-Comparison Sorts

Radix Sort

Announcements

- Quiz 1 grades and review video out soon

Goals:

- Know what it means for a sorting algorithm to be **stable**
- Understand the distinction between comparison and non-comparison sorts.
- Be prepared to implement radix sort.
- Know the definition of an in-place sorting algorithm.

Stability

Objects can be sorted on **keys** - **different** objects may have the same value.

A **stable** sort maintains the order of distinct elements with the same key.

- Example: sort the following list on the **tens** place only:

Sorted stably:

unstably:

Stability

Objects can be sorted on **keys** - **different** objects may have the same value.

A **stable** sort maintains the order of distinct elements with the same key.

- Example: sort the following list on the **tens** place only:

Sorted stably: **[21 23 35 48 61 63]**

unstably:

Stability

Objects can be sorted on **keys** - **different** objects may have the same value.

A **stable** sort maintains the order of distinct elements with the same key.

- Example: sort the following list on the **tens** place only:

[61 21 63 23 35 48]

Sorted stably: **[21 23 35 48 61 63]**

unstably:

Stability

Objects can be sorted on **keys** - **different** objects may have the same value.

A **stable** sort maintains the order of distinct elements with the same key.

- Example: sort the following list on the **tens** place only:

[61 21 63 23 35 48]

Sorted stably: **[21 23 35 48 61 63]**

unstably: **[23 21 35 48 61 63]**

Stability

Objects can be sorted on **keys** - **different** objects may have the same value.

A **stable** sort maintains the order of distinct elements with the same key.

- Example: sort the following list on the **tens** place only:

Sorted stably:

unstably:

Stability

Objects can be sorted on **keys** - **different** objects may have the same value.

A **stable** sort maintains the order of distinct elements with the same key.

- Example: sort the following list on the **tens** place only:

Sorted stably:

unstably: **[21 23 35 48 61 63]**

Stability

Objects can be sorted on **keys** - **different** objects may have the same value.

A **stable** sort maintains the order of distinct elements with the same key.

- Example: sort the following list on the **tens** place only:

[61 23 63 21 35 48]

Sorted stably:

unstably: [21 23 35 48 61 63]

Stability

Objects can be sorted on **keys** - **different** objects may have the same value.

A **stable** sort maintains the order of distinct elements with the same key.

- Example: sort the following list on the **tens** place only:

[61 23 63 21 35 48]

Sorted stably: [23 21 35 48 61 63]

unstably: [21 23 35 48 61 63]

Comparison sorts operate by comparing pairs of elements.

Examples: all four sorts we've seen so far!

...is there any other way to do it?

Comparison sorts operate by comparing pairs of elements.

Examples: all four sorts we've seen so far!

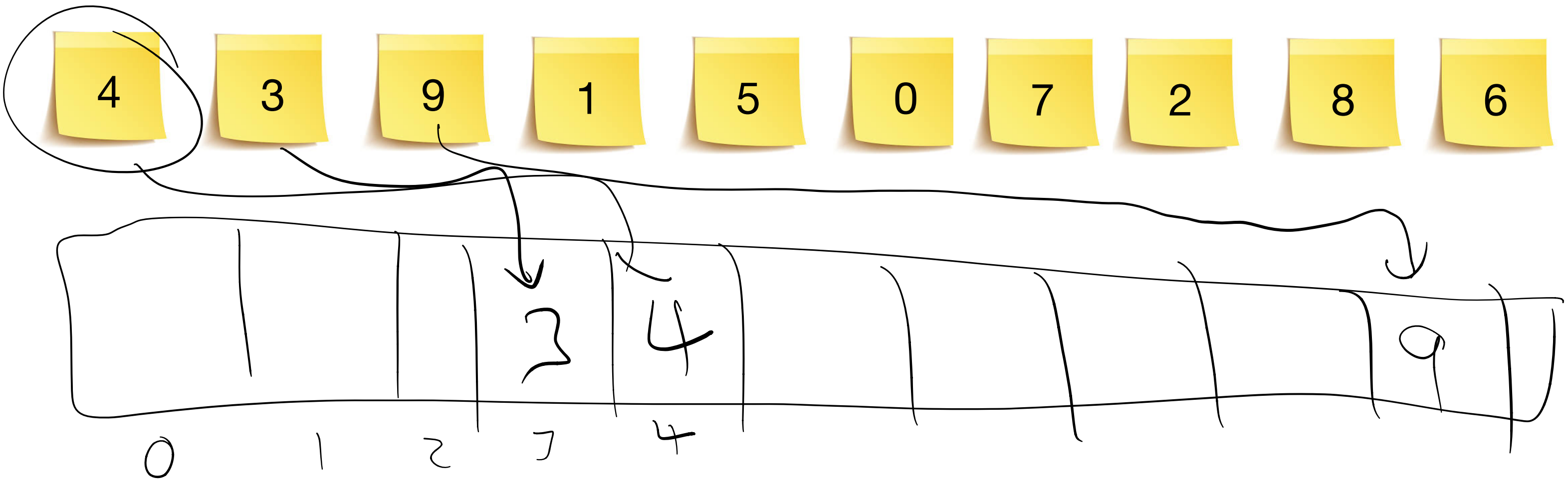
...is there any other way to do it?

How do you sort without comparing elements?

How do you sort things without comparing them?

Suppose I gave you 10 sticky notes with the digits 0 through 9.

What algorithm would you use to sort them?



How do you sort things without comparing them?

Suppose I gave you 10 sticky notes with the digits 0 through 9.

What algorithm would you use to sort them?

How many times did you need to look at each sticky note?

How do you sort things without comparing them?

Suppose I gave you 10 sticky notes with the digits 0 through 9.

What algorithm would you use to sort them?

How many times did you need to look at each sticky note?

What if there are duplicates?

LSD Radix Sort

```
/** least significant digit radix sort A */
LSDRadixSort(A):
max_digits = max # digits in any element of A
for d in 0..max_digits:
   stably sort A on the dth least significant
   digit

// A is now sorted(!)
```

1. ones place, then
2. tens place, then
3. hundreds place,

and so on

Do you believe me?

```
/** least significant digit radix sort A */
LSDRadixSort(A):
2= max_digits = max # digits in any element of A
  for d in 0..max_digits:
    stably sort A on the dth least significant
    digit

  // A is now sorted(!)
```

[45, 26, 42, 32] 07

42 32 45 26

26 32 42 45

Do you believe me?

```
/** least significant digit radix sort A */
LSDRadixSort(A):
max_digits = max # digits in any element of A
for d in 0..max_digits:
  stably sort A on the dth least significant
  digit

// A is now sorted(!)
```

Do you believe me?

```
/** least significant digit radix sort A */
LSDRadixSort(A):
max_digits = max # digits in any element of A
for d in 0..max_digits:
  stably sort A on the dth least significant
  digit

// A is now sorted(!)
```

Still don’t believe me? https://visualgo.net/en/sorting

LSD Radix Sort using queue buckets

Pseudocode from visualgo.net:

```
LSDRadixSort(A):
 create 10 buckets (queues) for each digit (0 to 9)
 for each digit (least- to most-significant):
   for each element in A:
     move element into its bucket based on digit
   for each bucket, starting from smallest digit
     while bucket is non-empty
       restore element to list
```

LSD Radix Sort using queue buckets

```
Pseudocode from visualgo.net:

LSDRadixSort(A):
 create 10 buckets (queues) for each digit (0 to 9)
 for each digit (least- to most-significant):
   for each element in A:
     move element into its bucket based on digit
   for each bucket, starting from smallest digit
     while bucket is non-empty
       restore element to list
```

LSD Intuition: sort on most-significant digit **last**; if tied, yield to the next most significant digit, and so on.
Only works because **stability** preserves orderings from less significant digits (previously sorted).

Exercise: Radix sort this

[7, 19, 21, 11, 14, 54, 1, 8]

Hint: [07, 19, 21, 11, 14, 54, 01, 08]

```
LSDRadixSort(A):
 create 10 buckets (queues) for each digit (0 to 9)
 for each digit (least- to most-significant):
   for each element in A:
     move element into its bucket based on digit
   for each bucket, starting from smallest digit
     while bucket is non-empty
       restore element to list
```

Exercise: Radix sort this

[07, 19, 61, 11, 14, 54, 01, 08]

n^2

$(n \cdot k) d$

Buckets on 1's place:

0	1	2	3	4	5	6	7	8	9
	01 11 61			54 14			07	08	19

Sorted on 1's place: 61 11 01 14 54 07 08 19

Buckets on 10's place:

0	1	2	3	4	5	6	7	8	9
08 07 01	19 14 11				54	61			

Sorted on 10's place: 01 07 08 11 14 19 54 61

LSD Radix Sort
using counting sort

```
/** least significant digit radix sort A */
LSDRadixSort(A):
max_digits = max # digits in any element of A
for d in 0..max_digits:
   counting sort A on the dth least
   significant digit

// A is now sorted(!)
```

Counting Sort

Formalizes what you did with the 0-9 sticky notes:

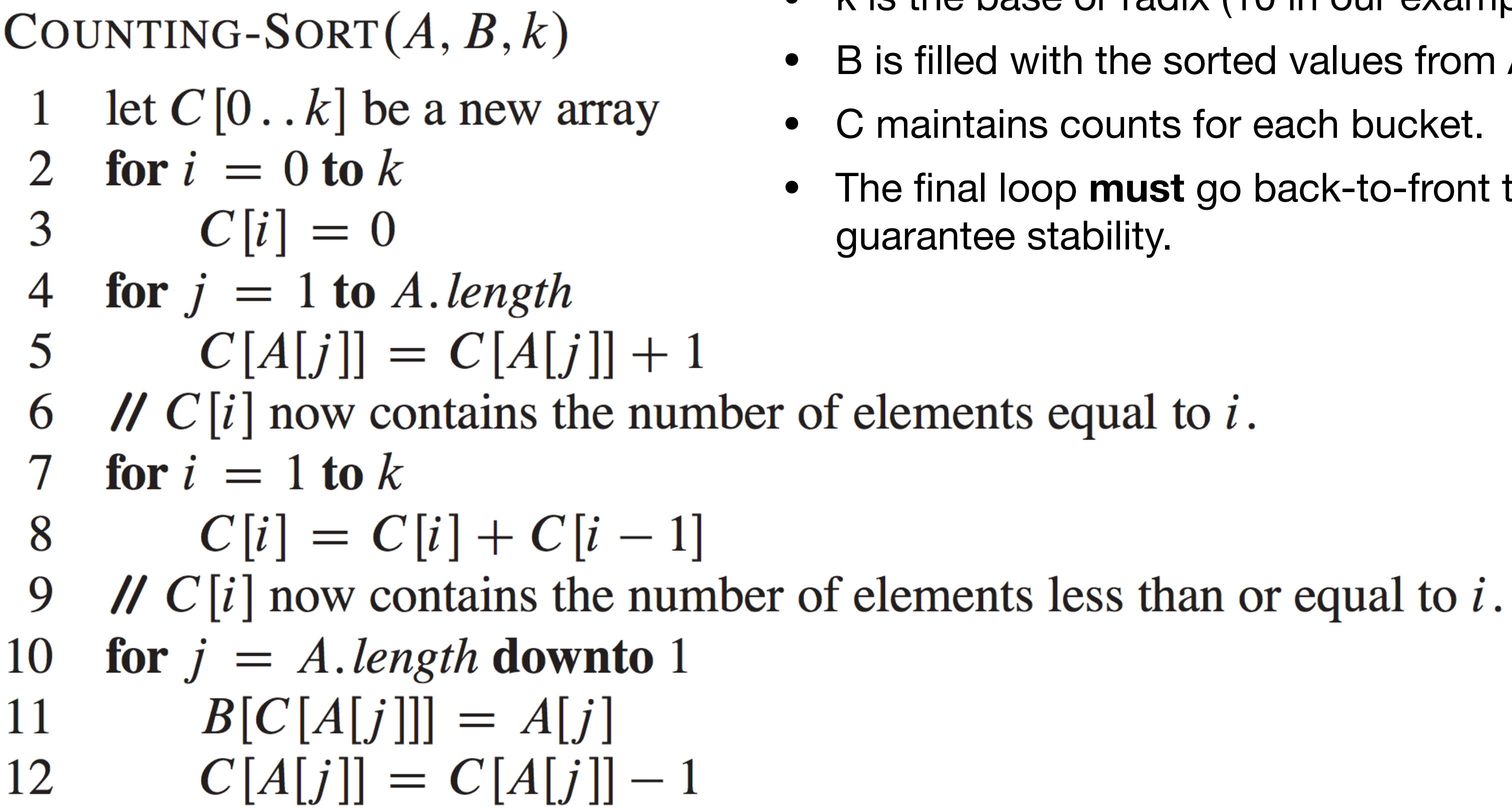
- Handles duplicates
- Stable sort
- Less memory overhead than queue buckets

Intuition:

http://www.cs.miami.edu/home/burt/learning/Csc517.091/workbook/countingsort.html

Pseudocode in CLRS (reproduced on the next slide).

Counting Sort - from CLRS



```
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
    $B[C[A[j]]] = A[j]$
    $C[A[j]] = C[A[j]] - 1$
```

Notes:

- k is the base or radix (10 in our examples)
- B is filled with the sorted values from A.
- C maintains counts for each bucket.
- The final loop **must** go back-to-front to guarantee stability.

in-place sorts

One more property of sorting algorithms aside from runtime.

A sorting algorithm is considered in-place if:

the extra storage used doesn't depend on the size of the input.

in-place sorts

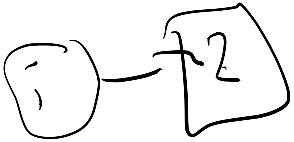
One more property of sorting algorithms aside from runtime.

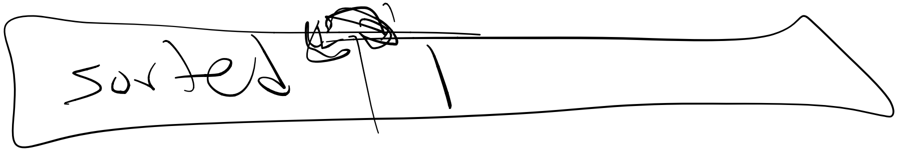
A sorting algorithm is considered in-place if:

the extra storage used doesn't depend on the size of the input.

↑ i.e., not part of the input array

↑ i.e., the array being sorted





	in-place?
insertion	Y
selection	Y
merge	?
quick	?
radix	?