CSCI 241

Lecture 7
Comparison Sorts
Radix Sort

Announcements

- Don’t leave A1/Lab 2 to the last minute. Radix sort is a bit tricky.

Goals:

- Understand the distinction between comparison and non-comparison sorts.
- Know the runtime and other tradeoffs between $O(n \log n)$ comparison sorts and radix sort.
- Be prepared to implement radix sort.

MergeSort is $O(n \log_2(n))$

- Side note: the base is just a constant factor!

$$\begin{aligned}\log_{10}(n) &= \frac{\log_2(n)}{\log_{10}(2)}\\ &= \frac{1}{\log_{10}(2)}\log_2 n\\ &= 0.30102999566\,\log_2(n)\end{aligned}$$

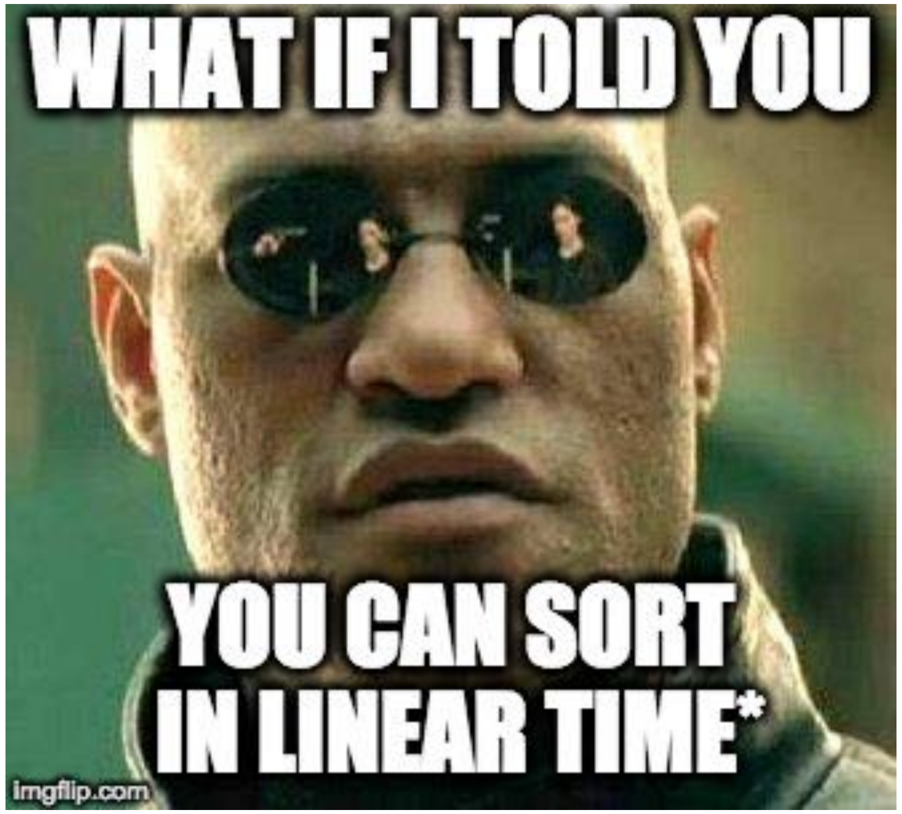
- Consequence: $\log_a(n)$ is $O(\log_b(n))$ for any constants a, b.

- We usually just write $O(\log n)$

Can we do any better?

- Fact: O(n log n) is provably optimal*.

  *for **comparison** sorts, which operate by comparing pairs of elements.



*if your values have a **constant** (O(1)) number of digits

Comparison sorts operate by comparing pairs of elements.

How do you sort without comparing elements?

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

What algorithm would you use to sort them?

What's the runtime?

What if there are duplicates?

Refresher: Stacks and Queues

(LIFO) (FIFO)

```
Stack s;
Queue q;

for i in 1..5:
  s.add(i) // push
  q.add(i) // enqueue
for i in 1..5:
  print s.remove() // pop
  print q.remove() // dequeue
```

ABCD: What is printed?

A. 1 1 2 2 3 3 4 4
B. 1 4 2 3 3 2 4 1
C. 4 1 3 2 2 3 1 4
D. 4 4 3 3 2 2 1 1

Stability

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

- e.g., sorting on 10's place only

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

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:
  do a stable sort of A on the dth least
  significant digit

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

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:
  do a stable sort of A on the dth least
  significant digit

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

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 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]

Buckets on 1’s place:

0	1	2	3	4	5	6	7	8	9

Sorted on 1’s place:

Buckets on 10’s place:

0	1	2	3	4	5	6	7	8	9

Sorted on 10’s place:

Exercise: Radix sort this

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

Buckets on 1's place:

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

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

LSD Radix Sort using queue buckets

What's the runtime?

```
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
```

n = a.length
d = max # digits?

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
```

n = a.length
d = max # digits?

ABCD: What's the runtime?

A. O(n) B. O(dn) C. O(d log n) D. O(n^2)

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:
   n   move element into its bucket based on digit
d    for each bucket, starting from smallest digit
       while bucket is non-empty
   n     restore element to list
```

n = a.length
d = max # digits?

ABCD: What's the runtime?

A. O(n) B. O(dn) C. O(d log n) D. O(n^2)

When would we prefer comparison sorts O(n log n) over radix sort O(dn)?

When is O(dn) better than O(n log n)?

When is $n * d < n * \log n$?

$d < \log n$

log(maxVal) log(# values)

Some other considerations:

- Is Radix sort in-place? Might prefer QuickSort.
- "O(n)" often hides large constants.
- Radix sort requires "digits"; comparison sorts work on anything pairwise comparable.

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 1-9 sticky notes:

- Handles duplicates
- Stable sort

Intuition:

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

Pseudocode in CLRS (and reproduced on the next slide).

Counting Sort - from CLRS

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.

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
```