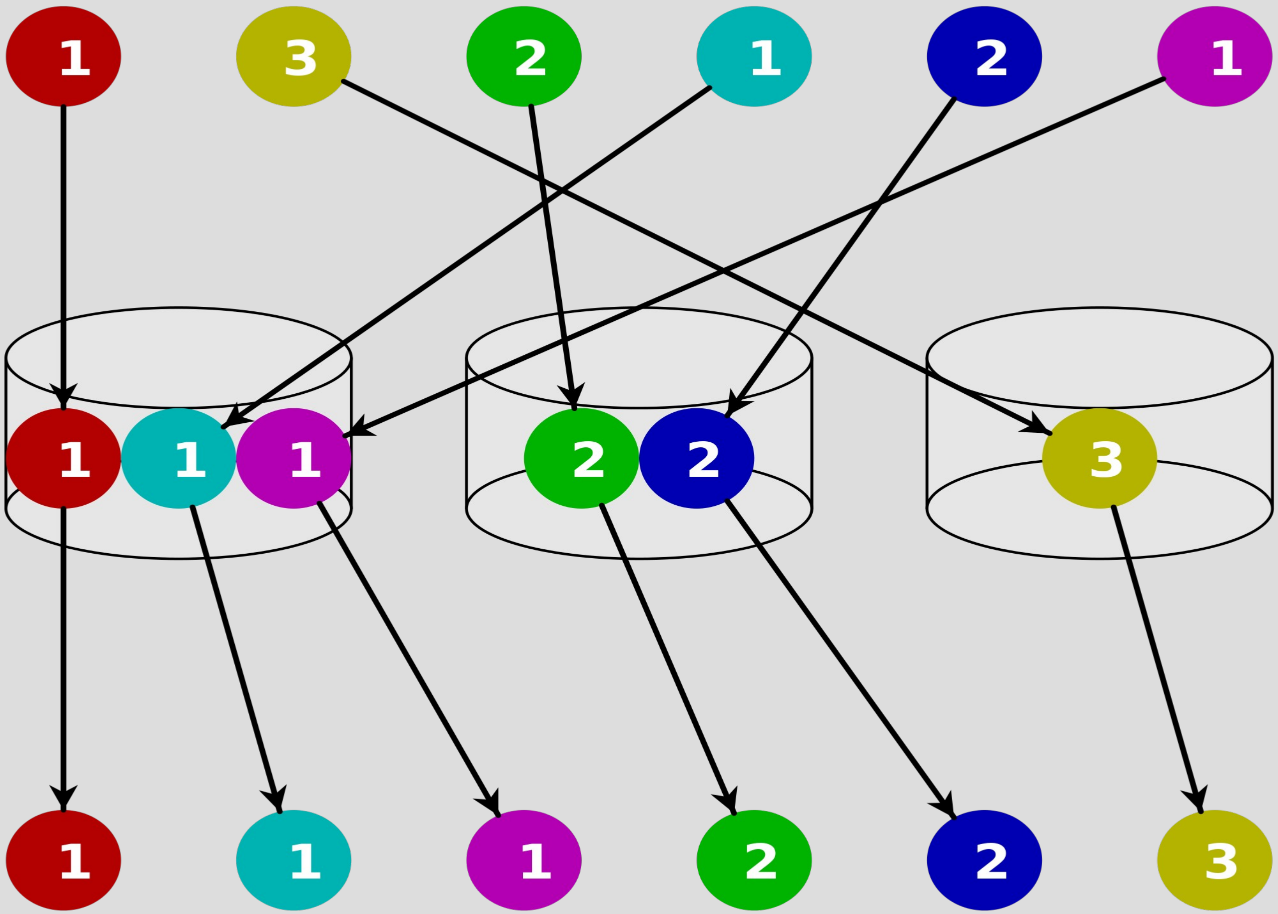
Sorting in O(n)



Announcements

HW will be posted tomorrow, due next Sunday 11:55pm

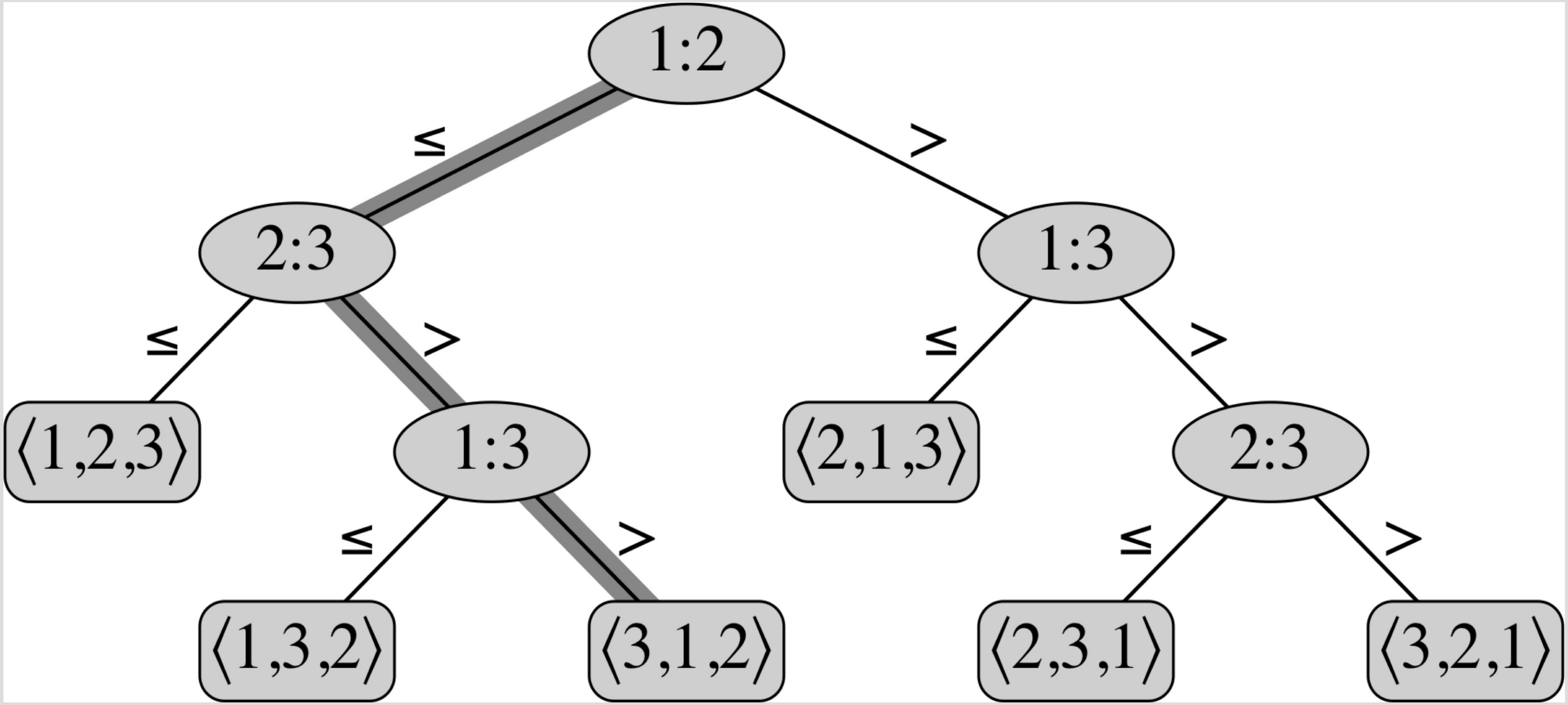
Sorting!

So far we have been looking at comparative sorts (where we only can compute < or >, but have no idea on range of numbers)

The minimum running time for this type of algorithm is $\Theta(n \lg n)$

Sorting!

All n! permutations must be leaves



Worst case is tree height

Sorting!

A binary tree (either $<$ or $\geq$) of height h has 2^h leaves:

$2^h \geq n!$

$\lg(2^h) \geq \lg(n!)$ (Stirling's approx)

$h \geq (n \lg n)$

Comparison sort

Today we will make assumptions about the input sequence to get O(n) running time sorts

This is typically accomplished by knowing the range of numbers

Outline

Sorting... again!

- -Comparison sort
- -Bucket sort
- -Count sort
- -Radix sort

Counting sort

1. Store in an array the number of times a number appears
2. Use above to find the last spot available for the number
3. Start from the last element, put it in the last spot (using 2.) decrease last spot array (2.)

Counting sort

A = input, B= output, C = count

```
for j = 1 to A.length
  C[ A[ j ]] = C[ A[ j ]] + 1
for i = 1 to k (range of numbers)
  C[ i ] = C[ i ] + C [ i – 1 ]
for j = A.length to 1
  B[ C[ A[ j ]]] = A[ j ]
  C[ A[ j ]] = C[ A[ j ]] - 1
```

Counting sort

k = 5 (numbers are 2-7)

Sort: {2, 7, 4, 3, 6, 3, 6, 3}

1. Find number of times each number appears

C = {1, 3, 1, 0, 2, 1}

2, 3, 4, 5, 6, 7

Counting sort

Sort: {2, 7, 4, 3, 6, 3, 6, 3}

2. Change C to find last place of each element (first index is 1)

C = {1, 3, 1, 0, 2, 1}

{1, 4, 1, 0, 2, 1}

{1, 4, 5, 0, 2, 1}{1, 4, 5, 5, 7, 1}

{1, 4, 5, 5, 2, 1}{1, 4, 5, 5, 7, 8}

Counting sort

Sort: {2, 7, 4, 3, 6, 3, 6, 3}

3. Go start to last, putting each element into the last spot avail.

C = {1, 4, 5, 5, 7, 8}, last in list = 3

2 3 4 5 6 7

{ , , ,3, , , , }, C =

1 2 3 4 5 6 7 8 {1, 3, 5, 5, 7, 8}

Counting sort

Sort: {2, 7, 4, 3, 6, 3, 6, 3}

3. Go start to last, putting each element into the last spot avail.

C = {1, 4, 5, 5, 7, 8}, last in list = 6

2 3 4 5 6 7

{ , , ,3, , ,6, }, C =

1 2 3 4 5 6 7 8 {1, 3, 5, 5, 6, 8}

Counting sort

Sort: {2, 7, 4, 3, 6, 3, 6, 3}

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

{ ,   ,   ,3,   ,   ,6,   }, C={1,3,5,5,6,8}

{ ,   ,3,3,   ,   ,6,   }, C={1,2,5,5,6,8}

{ ,   ,3,3,   ,6,6,   }, C={1,2,5,5,5,8}

{ , 3,3,3,   ,6,6,   }, C={1,1,5,5,5,8}

{ , 3,3,3,4,6,6,   }, C={1,1,4,5,5,8}

{ , 3,3,3,4,6,6,7}, C={1,1,4,5,5,7}

Counting sort

Run time?

Counting sort

Run time?

Loop over C once, A twice

$k + 2n = O(n)$ as k a constant

Counting sort

Does counting sort work if you find the first spot to put a number in rather than the last spot?

If yes, write an algorithm for this in loose pseudo-code

If no, explain why

Counting sort

Sort: {2, 7, 4, 3, 6, 3, 6, 3}

C = {1,3,1,0,2,1} -> {1,4,5,5,7,8}

instead $C[\,i\,] = \text{sum}_{j<i}\, C[\,j\,]$

C' = {0, 1, 4, 5, 5, 7}

Add from start of original and increment

Counting sort

A = input, B= output, C = count

for j = 1 to A.length

  C[A[j]] = C[A[j]] + 1

for i = 2 to k (range of numbers)

  C'[i] = C'[i-1] + C [i – 1]

for j = A.length to 1

  B[C[A[j]]] = A[j]

  C[A[j]] = C[A[j]] + 1

Counting sort

Counting sort is stable, which means the last element in the order of repeated numbers is preserved from input to output

(in example, first '3' in original list is first '3' in sorted list)

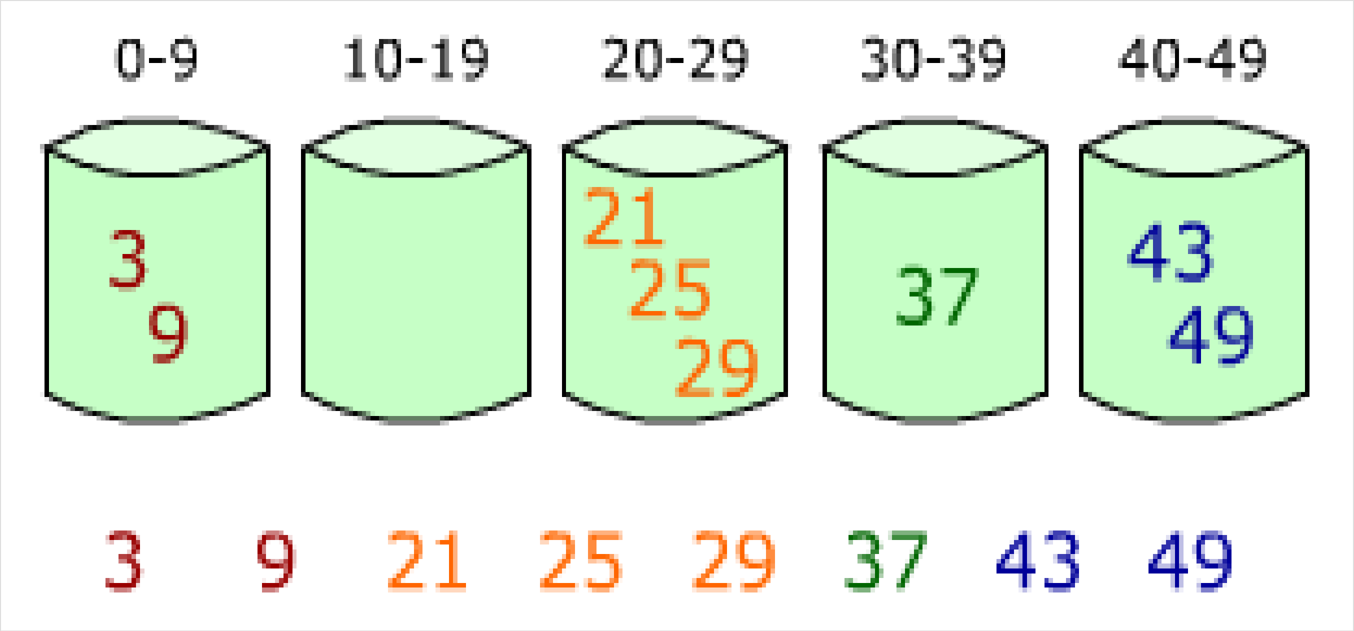
Bucket sort

1. Group similar items into a bucket

2. Sort each bucket individually

3. Merge buckets

Bucket sort



As a human, I recommend this sort if you have large n

Bucket sort

(specific to fractional numbers)
(also assumes n buckets for n numbers)

```
for i = 1 to n // n = A.length
  insert A[ i ] into B[floor(n A[ i ])+1]
for i = 1 to n // n = B.length
  sort list B[ i ] with insertion sort
concatenate B[1] to B[2] to B[3]...
```

Bucket sort

Run time?

Bucket sort

Run time?

$\Theta(n)$

Proof is gross... but with n buckets each bucket will have on average a constant number of elements

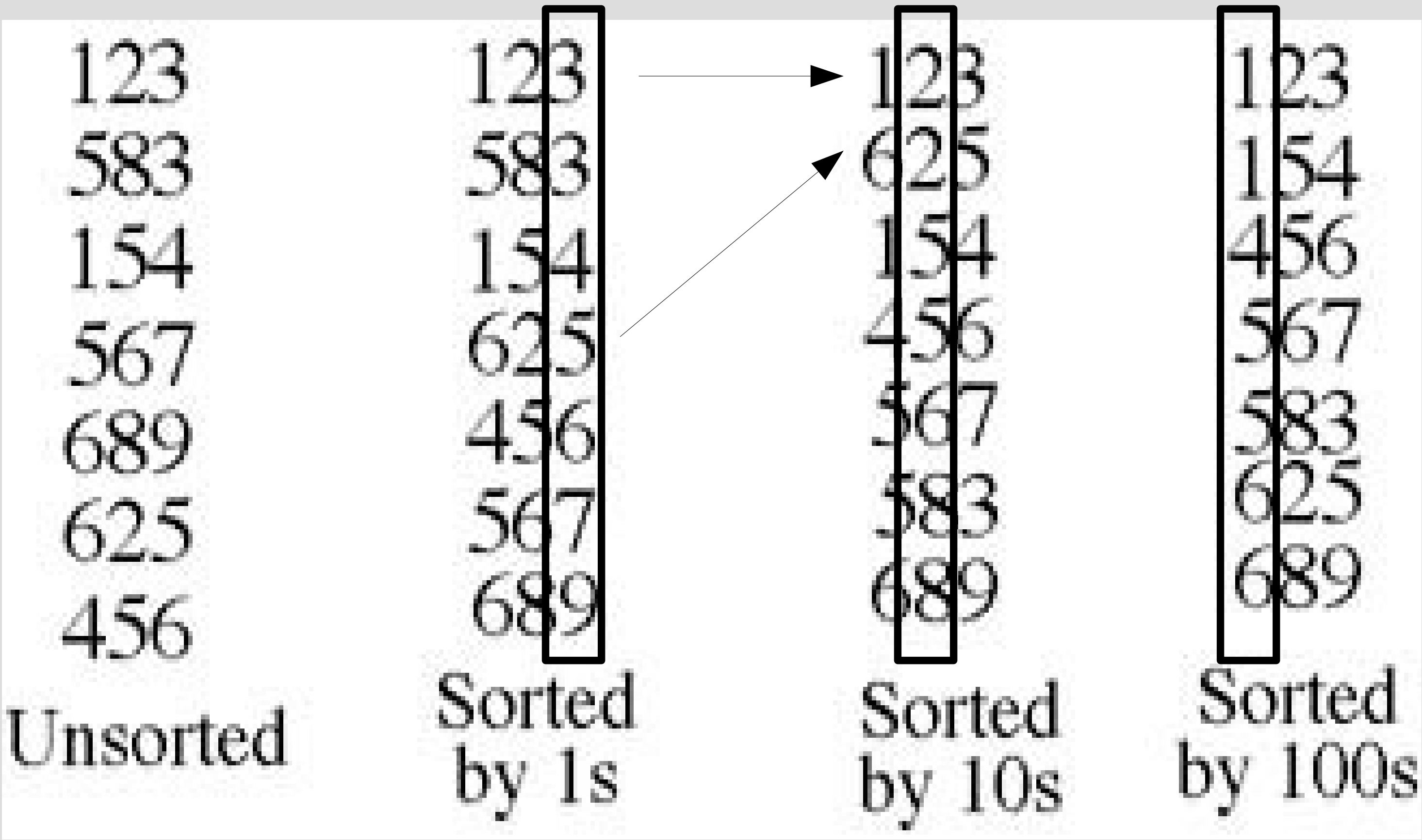
Radix sort

Use a **stable** sort to sort from the least significant digit to most

Psuedo code: (A=input)

```
for i = 1 to d
   stable sort of A on digit i
```

Radix sort



Stable means you can draw lines without crossing for a single digit

Radix sort

Run time?

Radix sort

Run time?

$O((b/r) (n+2^r))$

b-bits total, r bits per 'digit'

$d = b/r$ digits

Each count sort takes $O(n + 2^r)$

runs count sort d times...

$O(d(n+2^r)) = O(b/r (n + 2^r))$

Radix sort

Run time?

if $b < \lg(n)$, $\Theta(n)$

if $b \geq \lg(n)$, $\Theta(n \lg n)$