Sorting in O(n)
Announcements

Too fast?

Homework 1 example
Project

Previous class: Line tracking (Hough Transformation)
Project

Previous class: Line tracking (Hough Transformation)
Project

Previous class: Line tracking (Hough Transformation)
Project

Project turn in:
1. Paper (intro, related work, description, results, conclusion)
2. Code and data

Paper should not be under 4 pages
Outline

Sorting... again!
- Comparison sort
- Count sort
- Radix sort
- Bucket sort
Comparison sort

So far we have not put any restrictions on the sequence to sort.

We will show that the fastest an algorithm can sort this in is $O(n \log n)$
(i.e. all general sorts are $\Omega(n \log n)$).
Comparison sort

All $n!$ permutations must be leaves

Worst case is tree height
Comparison sort

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

$$2^h \geq n!$$

$$\lg(2^h) \geq \lg(n!) \quad \text{(Stirling's approx)}$$

$$h \geq (n \lg n)$$
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.
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

If you know the numbers are 1-8

loop over i ( \( s[ u[i] ] = u[i] \) )
Counting sort

If you know the numbers are 1-8

loop over i ( \( s[ u[i]] = u[i] \))

Does not work with repeated numbers
Counting sort

Offset by more if not repeated

2 3 5 6 6 8 9

4,5 7,8
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

\[ k = 5 \text{ (numbers are 2-7)} \]
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.
\[ C = \{1, 4, 5, 5, 7, 8\}, \text{ last in list } = 3 \]
\[ \begin{array}{cccccc}
2 & 3 & 4 & 5 & 6 & 7 \\
, & , & , & , & , & , \\
\end{array} \]
\[ C = \{1, 3, 5, 5, 7, 8\} \]
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\}

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
2,3,4,5,6,7 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
, & , & ,3, & , , ,6, & } & , & C=\{1,3,5,5,6,8\} \\
, & , & 3,3, & , , ,6, & } & , & C=\{1,2,5,5,6,8\} \\
, & , & 3,3, & , ,6,6, & } & , & C=\{1,2,5,5,5,8\} \\
, & , & 3,3, & , ,6,6, & } & , & C=\{1,1,5,5,5,8\} \\
, & , & 3,3, & , ,3,4,6,6, & } & , & C=\{1,1,4,5,5,8\} \\
, & , & 3,3, & , ,3,4,6,6,7 & }, & C=\{1,1,4,5,5,7\} \\
\end{array}
\]
Counting sort

Run time?
Counting sort

Run time?

Loop over C once, A twice

\[ k + 2n = O(n) \text{ as } k \text{ a constant} \]
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 ] = \sum_{j<i} C[ j ]

C' = \{0, 1, 4, 5, 5, 7\}
Add from start of original and increment
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)
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?

$O\left(\frac{b}{r} \left(n+2^r\right)\right)$

$b$-bits total, $r$ bits per 'digit'

digit $d = \frac{b}{r}$ digits

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

runs count sort $d$ times...

$O\left(d(n+2^r)\right) = O\left(\frac{b}{r} \left(n + 2^r\right)\right)$
Radix sort

Run time?

if $b < \lg(n)$, $\Theta(n)$
if $b \geq \lg(n)$, $\Theta(n \lg n)$
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 = 0 to A.length
    insert A[ i ] into B[ floor(n A[ i ]) ]
for i = 0 to B.length
    sort list B[ i ] with insertion sort
concatenate B[0] to B[1] to B[2]...
Bucket sort

Run time?
Bucket sort

Run time?

Θ(n)