

# Linear-time Sorting

Note Title

9/27/2007

Last time: Any comparison-based sorting alg has worst-case complexity  $\Omega(n \log n)$

Can we sort in a way not based on comparing elements?

Consider sorting  $n$  elements that are all integers  $\{0, 1, \dots, k-1\}$

E.g.  $k=10$

Idea: keep 10 lists & put all elements of value  $i$  into list  $i$

Using an array: Count # of occurrences of each element & then you can put them in order

# Counting Sort

basic idea we've just described.

We'll need one more property

Stable sort - any two equivalent elements are kept in same relative order

d, a, b, c, b  $\xrightarrow{\text{Stable}}$  a, b, b, c, d

# Radix Counting Sort (input, output, k) {

for d = 0 to #digits - 1

int n = input.length;

for (i = 0; i < k; i++)

count[i] = 0;

for (j = 0; j < n; j++)

count[input[j]]++;

for (i = 1; i < k; i++)

count[i] += count[i-1];

for (j = n-1; j >= 0; j--)

output[--count[input[j]]] = input[j];

radix  
sort }

digitizer.getDigit(input[j], d) current digit

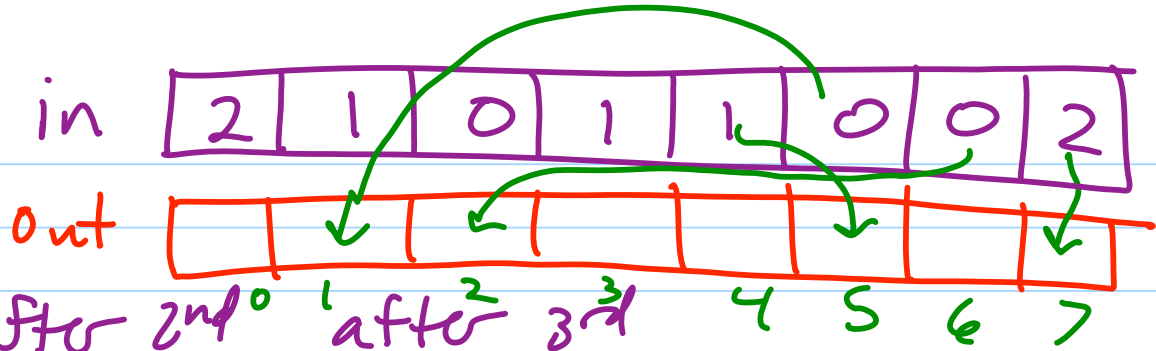
array of same length

at end

count[i] is # of elements that are  $\leq i$

count[i] is # of elements  $\leq i$

k=3



<u>Count</u>	<u>after 1<sup>st</sup> loop</u>	<u>after 2<sup>nd</sup> loop</u>	<u>after 3<sup>rd</sup> Loop</u>	
0	0	3	<del>3</del>	2 1
1	0	3	<del>6</del>	5
2	0	2	<del>8</del>	7

Time complexity

$$\Theta(n+k)$$

two loops over counters  $\Theta(k)$

two loops over elements  $\Theta(n)$

When  $k = O(n)$  then this

is a  $\Theta(n)$  sort.

↖ linear time

Suppose I want to sort social  
security number

9 digits

What would "k" be if we  
wanted to use counting  
sort?

$10^9$

As another example we might want to alphabetize names.

General purpose sorting method

- provide a comparator

- another option is to provide

Digitizer

# Digitizer Interface

int getBase(): # values  $b$  where  
digit values  $0, \dots, b-1$

like  $k$   
in counting  
sort  
↙

words,  $b = 26$

int numDigits(T x): # digits in element

$Cab$ , # digits = 3

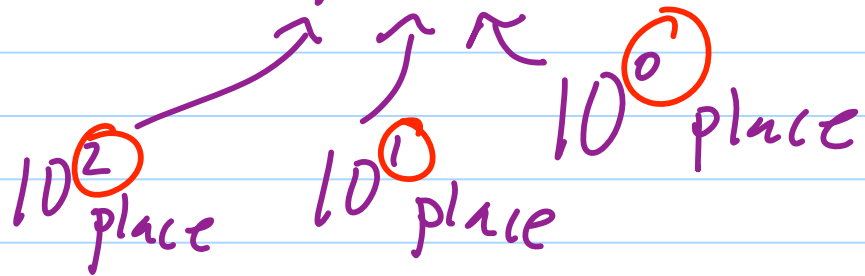
int getDigit(T x, int place): value of place  
for  $x$

$x = Cab$   
2 0 1

getDigit(x, 0) = 1

getDigit(x, 1) = 0  
...

5 7 9 8 1



Counting sort treats the element as just one digit

## Radix Sort

From least to most significant digit  
 $d = 0$  to  $b-1$

apply counting sort where we access digit  $d$

ex

329  
457  
657  
839  
436  
720  
355

→  
phase 0  
using  
counting  
sort on  
digit 0

720  
355  
436  
457  
657  
329  
839

phase  
1  
→  
place  
1

720  
329  
436  
839  
355  
457  
657

→  
look  
at  
place  
2

329  
355  
436  
457  
657  
720  
839

## Correctness Highlights.

Prove following holds using induction

After first  $p$  phases, numbers  
when looking at least significant  
 $p$  digits are sorted

base: true after 1 phase by <sup>correctness of</sup> counting sort

inductive step: combined inductive hyp,  
correctness of counting sort + stability of counting sort

What's the asymptotic time complexity?

$n$  # elements

$d$  max # of digits of elements  
(pad accordingly within digitizer)

$b$  base of each digit ( $0, 1, \dots, b-1$ )

$$\Theta(d(n+b))$$

Return to the ex.

$n = \#$  social security #s to sort

$d = 9$   
 $b = 10$  ) treat each base-10  
digit as a digit  
for radix sort

$$\Theta(\underline{9}(n+10)) = \Theta(n)$$

$$\Theta(9(n+10))$$

$\underbrace{XXX}_{\text{base 1000 number}} \underbrace{XXX}_{\text{digit}} \underbrace{XXX}_{\text{digit}}$

0, ..., 999

roughly  
3 times  
faster  
for "large" n

Time complexity with the digitizer

$$\Theta(3(n+1000))$$

Consider sorting #s that begin  
in binary (base 2)

$n$  #s

$b$  bits ( $b=32$ , 32-bit #)

group  $r$  bits into a digit

$$\# \text{ digits} = \frac{b}{r}$$

$$\text{base per digit} = 2^r$$

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

time complexity

$$\frac{C \cdot b}{r}(n+2^r)$$

} could even work out constants more carefully

min with respect to  $r$ .

roughly optimizes when  $r = \Theta(\log_2 n)$