Radix sort

number x written in base b
as seq. of digits $a_e \ldots a_3 a_2 a_1 a_0$

most significant ⟵ ⟶ least signif.

means $x = b^e a_e + \cdots + b^3 a_3 + b^2 a_2 + b^1 a_1 + a_0$

$$= \sum b^i a_i$$

b^i: base to power i; a_i: ith digit (from right)

e.g. in decimal numbers

$2015 = 2 \times 10^3 + 0 \times 10^2 + 1 \times 10^1 + 5 \times 10^0$

in binary

$11011110 =$

$1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

Radix sort:

given n data items as a list L
whose keys are numbers in range $0 \le key < K$
written as base-b numbers
(#digits $= \lceil \log_b K \rceil = \left\lceil \frac{\log K}{\log b} \right\rceil$)

for $i = 0, 1, 2, \ldots$ (#digits $- 1$):
(each digit position starting with least significant, ending with most signif.)

L = stable bucket sort L by digit i of keys

after sorting by digit i (i+1) list is sorted mod base

input: 314, 321, 911, 214, 376, 508, 524

sort by 1's digit: 321, 911, 314, 214, 524, 376, 508
(bucket 1: 321, 911; bucket 4: 314, 214, 524)

sort by 10's digit: 508, 911, 314, 214, 321, 524, 376

sort by 100's digit: 214, 314, 321, 376, 508, 524, 911

Analysis:

loop $\left\lceil \frac{\log K}{\log b} \right\rceil$ times

$O(n+b)$ per loop

total: $O\left((n+b)\left\lceil \frac{\log K}{\log b} \right\rceil\right)$

If you use binary ($b=2$) or decimal ($b=10$) you can simplify O-notation to $O(n \log K)$

Never a good choice!

If $K \leq n$

bucket sort is $O(n)$

this is not $O(n)$

bucket sort is better

If $K \geq n$

comparison sorts are $O(n \log n)$

$n \log n \leq n \log K$

so they are better

b is a free parameter —
we can choose value
that gives best possible runtime
right choice: $b \approx n$
(may be const $\times n$, may be n itself)

then time becomes $O\left(n \left\lceil \frac{\log K}{\log n} \right\rceil\right)$

linear when $K = O(n^2)$ $O(n^3)$
or $K = O(n^{\text{any const. power}})$

better than comparison sort
for $n < k < n^{\log n}$

best choice for b because:
smaller values of b use
more #digits, more iterations
without saving time/iteration
larger values of b waste too much time
on many empty buckets

To convert a number x to base b (giving d digits):

repeat d times:

output $x \bmod b$

$x = \lfloor x / b \rfloor$

Issue: Uses slow division, modulus operations

Would like:
- to be able to pull out ith digit without looping through all earlier digits (so can find bucket sort keys when we need them rather than converting whole number ahead of time)
- to use faster operations (addition, bitwise ops)

Possible when base is power of 2!

given binary numbers

x	1100	0101	1011	0110
y	1001	1010	0010	1011
flip all bits ~x	0011	1010	0100	1001
bitwise and x & y	1000	0000	0010	0010
" or x | y	1101	1111	1111	1111
(add mod 2) xor x ^ y	0101	1111	1101	1101
shift left x <<4	0101	1011	0110	0000
shift right x >>4	0000	1100	0101	1011

groups of k digits = base 2^k notation

x = d 5 b 6 (hexadecimal)

digit i of x in base 2^k:

$$\left(x >> (i * k)\right) \&_1 \left((1 << k) - 1\right)$$

precompute shift amount at start of loop in radix sort

and mask

1 (binary): ...00000001

(1 << k): 0001 0000

k zeros

subtract 0001 0000

- 00000001

0000 1111

k ones