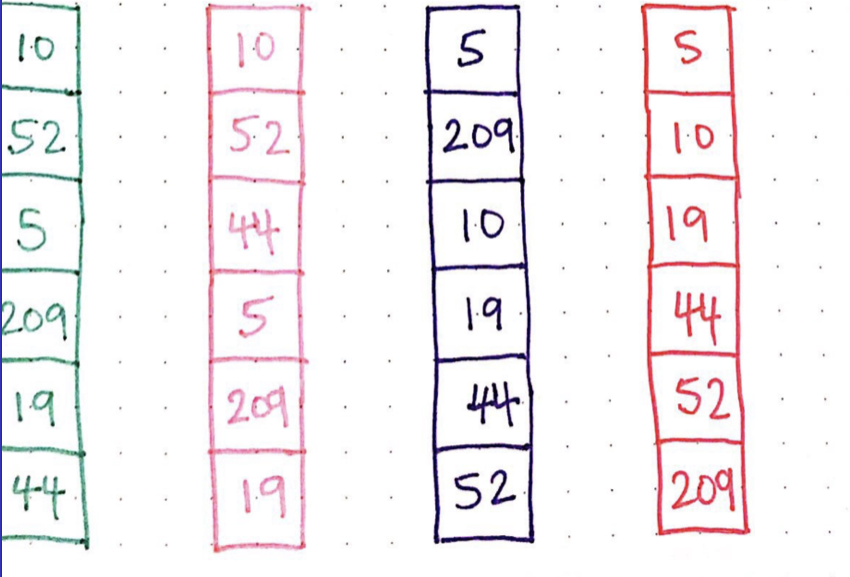
Fast Sorting Algorithms



Sorting algorithms on values in a known range

CLRS Ch.8

- Counting sort
- Radix sort
- Lower bounds for general sorting

- The algorithms will sort an array $A[n]$ of non-negative integers in the range $[0, r]$.
- The complexity of the algorithms depends on both n and r.
- For some values of r, the algorithms have cost $O(n)$ or$o(n \log n)$.

Counting sort

The counting sort algorithm,

- consider all possible values $i \in [0, r]$.
- For each of them, count how many elements in A are smaller or equal to i.
- Use this information to place the elements in the right order.

- The input $A[n]$, is an array of integers in the range $[0, r]$.
- Uses: $B[n]$ (output) and $C[r+1]$ (internal).

Counting sort: Algorithm

CountingSort (A, r)

for $i = 0$ to r **do**
  $C[i] = 0$
for $i = 0$ to $n-1$ **do**
  $C[A[i]] = C[A[i]] + 1$ $\{C[j] = |\{i \mid A[i] = j\}|\}$
for $i = 1$ to r **do**
  $C[i] = C[i] + C[i-1]$ $\{C[j] = |\{i \mid A[i] \leq j\}|\}$
for $i = n-1$ downto 0 **do**
  $B[C[A[i]]] = A[i]$;
  $C[A[i]] = C[A[i]] - 1$ $\{C$ holds the sorted elements$\}$

¿ B holds them ?

Counting sort: Cost

CountingSort (A, r)

for $i = 0$ to r **do**
  $C[i] = 0$ $\{O(r)\}$
for $i = 0$ to $n - 1$ **do**
  $C[A[i]] = C[A[i]] + 1$ $\{O(n)\}$
for $i = 0$ to r **do**
  do $C[i] = C[i] + C[i - 1]$ $\{O(r)\}$
for $i = n - 1$ downto 0 **do**
  $B[C[A[i]] - 1] = A[i]$;
  $C[A[i]] = C[A[i]] - 1$ $\{O(n)\}$

$T(n) = O(n + r)$, for $r = O(n)$, $T(n) = O(n)$.

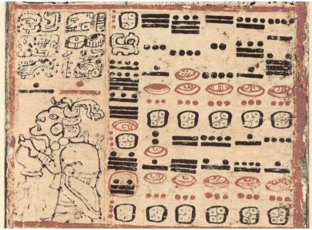
Counting sort: stability

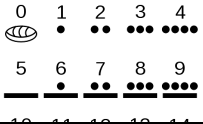
An important property of counting sort is that it is stable: numbers with the same value appear in the output in the same order as they do in the input.

Radix sort: What does radix mean?

Radix means the base in which we express an integer

Radix 10=Decimal; Radix 2= Binary; Radix 16=Hexadecimal; Radix 20 (The Maya numerical system)





Binary	Hex	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Radix Change: Example

- To convert an integer from binary to decimal:
  $1011 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 11$
- To convert an integer from decimal to binary:
  Repeatedly dividing the enter by 2, will give a result plus a remainder:
  $19 \Rightarrow \underbrace{19/2}_{1}\ \underbrace{9/2}_{1}\ \underbrace{4/2}_{0}\ \underbrace{2/2}_{01} = 10011$
- To transform an integer radix 16 to decimal:
  $(4CF5)_{16} = (4 \times 16^3 + 12 \times 16^2 + 15 \times 16^1 + 5 \times 16^0) = 19701$

- To convert $(4CF5)_{16}$ into binary you have to expand each digit to its binary representation.
  In the above example, $(4CF5)_{16}$ in binary is
  0011110011110101
- To convert an integer in binary to radix 16:
  Make groups of 4 from left to right and replace by the corresponding digit
  11010100101000100001011011110100 in HEX is
  1A9442DF4

RADIX LSD algorithm

Given an array A with n numbers, each one with d digits in base b the Radix Least Significant Digit, algorithm is

RADIX LSD (A, d, b)
for $i = 1$ to d **do**
  Use a stable sorting algorithm to sort A according to the i-th digit values.

The values to sort are in the range $[0, 2^d)$.

Example: $b = 10$ and $d = 3$

329
475
657
839
436
720
355

Example: $b = 10$ and $d = 3$

$$
\begin{array}{ccc}
329 & & 72\color{red}{0} \\
475 & & 47\color{red}{5} \\
657 & & 35\color{red}{5} \\
839 & \Rightarrow & 43\color{red}{6} \\
436 & & 65\color{red}{7} \\
720 & & 32\color{red}{9} \\
355 & & 83\color{red}{9}
\end{array}
$$

Example: $b = 10$ and $d = 3$

$$
\begin{array}{ccccc}
329 & & 72\color{red}{0} & & 7\color{red}{2}0 \\
475 & & 47\color{red}{5} & & 3\color{red}{2}9 \\
657 & & 35\color{red}{5} & & 4\color{red}{3}6 \\
839 & \Rightarrow & 43\color{red}{6} & \Rightarrow & 8\color{red}{3}9 \\
436 & & 65\color{red}{7} & & 3\color{red}{5}5 \\
720 & & 32\color{red}{9} & & 6\color{red}{5}7 \\
355 & & 83\color{red}{9} & & 4\color{red}{7}5
\end{array}
$$

Example: $b = 10$ and $d = 3$

$$
\begin{array}{ccccccc}
329 & & 72\color{red}{0} & & 7\color{red}{2}0 & & \color{red}{3}29 \\
475 & & 47\color{red}{5} & & 3\color{red}{2}9 & & \color{red}{3}55 \\
657 & & 35\color{red}{5} & & 4\color{red}{3}6 & & \color{red}{4}36 \\
839 & \Rightarrow & 43\color{red}{6} & \Rightarrow & 8\color{red}{3}9 & \Rightarrow & \color{red}{4}75 \\
436 & & 65\color{red}{7} & & 3\color{red}{5}5 & & \color{red}{6}57 \\
720 & & 32\color{red}{9} & & 6\color{red}{5}7 & & \color{red}{7}20 \\
355 & & 83\color{red}{9} & & 4\color{red}{7}5 & & \color{red}{8}39
\end{array}
$$

Correctness

Theorem

RADIX LSD sorts correctly the n given numbers.

Induction on d.

Base: If $d = 1$ the stable sorting algorithm sorts correctly.
IH: Assume that it is true for $d - 1$ digits.
Looking at the the d-th digit, we have

- if $a_d < b_d$, $a < b$ and the algorithm places a before b,
- if $a_d = b_d$, as we are using a stable sorting, a and b remain in the same order as in the previous step.
  By IH, they are already the correct one.

□

Time complexity

Given n numbers, each number with at most d digits, and each digit in the range 0 to b, if we use counting sorting at each round of RADIX LSD:

$$T(n, d, b) = \Theta(d(n + b)).$$

- Consider that each number has a value up to $f(n)$.
- Then the number of digits is $d = \lceil \log_b f(n) \rceil$, so $T(n, b) = \Theta(\log_b f(n)(n + b))$,
- if $\log_b f(n) = \omega(1)$, $T(n, b) = \omega(n)$ and RADIX is not linear.

RADIX: selecting the base

Can we tune the parameters?

- Yes, in some cases, we can select the best radix to express the input values.
- For numbers in binary, we can select as new radix $\hat{b}$ a power of 2. This simplifies the computation as we have only to look to pieces of bits to change from one representation to anoter.
- For ex., if we have numbers of $d = 64$ bits $(b = 2)$, and take the new radix to be $\hat{b} = 2^8$, we have $\hat{d} = 4$ new digits per number.

1 1 0 0 1 0 1 0 0 0 1 1 0 1 0 0 1 1 1 0 1 0 0 1 1 1 0 0 1 0 0 0

RADIX: selecting the base

Given n, d-bits integers, we want to choose an integer e, $1 < e < d$ to use as new radix $\hat{b} = 2^e$.

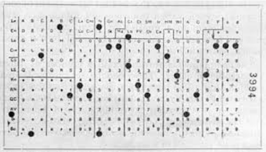
- In the new radix, the number of digits is $\hat{d} = \lceil d/e \rceil$ digits,
- Running RADIX LSD with base 2^e has cost the new $\hat{d} = \lceil d/e \rceil$ digits,

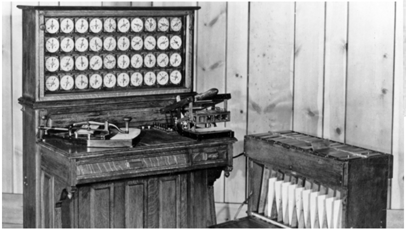
$$T(n) = \Theta(\hat{d}(n + 2^e)) = \Theta((d/e)(n + 2^e)).$$

- The best choice for e is roughly $\lceil \lg n \rceil$.
  - Then, $2^e = O(n)$.
  - So, the cost is, $O(\frac{d}{\lg n} n)$.
  - Which provides, linear cost if $\frac{d}{\lg n} = O(1)$.

A bit of history.

Radix and counting sort ideas are due to Herman Hollerith.
In 1890 he invented the card sorter that, for ex., allowed to process the US census in 5 weeks, using punching cards.





Upper and lower bounds on time complexity of a problem.

- A problem has a time upper bound $T(n)$ if there is an algorithm $\mathcal{A}$ such that, for any input x of size n, $A(x)$ gives the correct answer in $\leq T(n)$ steps.
- A problem has a time lower bound $L(n)$ if there is NO algorithm which solves the problem in time $< L(n)$, for any input e of size n.
- Lower bounds are hard to prove, as we have to consider every possible algorithm.

Upper and lower bounds on time complexity of a problem.

- Upper bound: $\exists \mathcal{A}, \forall x \; t_{\mathcal{A}}(x) \leq T(|x|)$,
- Lower bound: $\forall \mathcal{A}, \exists x \; t_{\mathcal{A}}(x) \geq L(|x|)$,

To prove an upper bound: produce an A so that the bound holds for any input x ($n = |x|$).

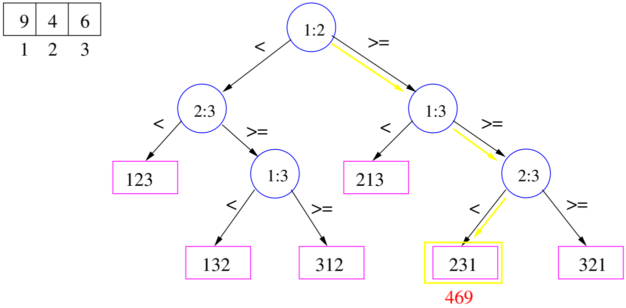
To prove a lower bound , show that for any possible algorithm, the time on one input is greater than or equal to the lower bound.

Lower bound for **comparison based** sorting algorithm.

To prove the lower bound, we consider binary decision trees a way to represent the comparisons made by a sorting algorithm to distinguish the possible inputs of size n.

- each leaf represents one of the $n!$ possible permutations $(a_{\pi(1)}, a_{\pi(2)}, \ldots, a_{\pi(n)})$. The tree has exactly $n!$ leaves.
- each internal node is labeled by a comparison $a_i : a_j$, the leaves in the left subtree verify $a_i < a_j$ and the ones in the right subtree verify $a_i \geq a_j$.

An example of binary decision tree for $n = 3$



Theorem

For any comparison sort algorithm that sorts n elements, there is an input in which it has to perform $\Omega(n \lg n)$ comparisons.

Proof.

- Equivalent to prove: Any decision tree that sorts n elements must have height $\Omega(n \lg n)$.
- Let h the height of a decision tree with $n!$ leaves,

$$n! \leq 2^h \Rightarrow h \geq \lg(n!) > \lg(\frac{n}{e})^n = \Omega(n \lg n).$$

$\square$