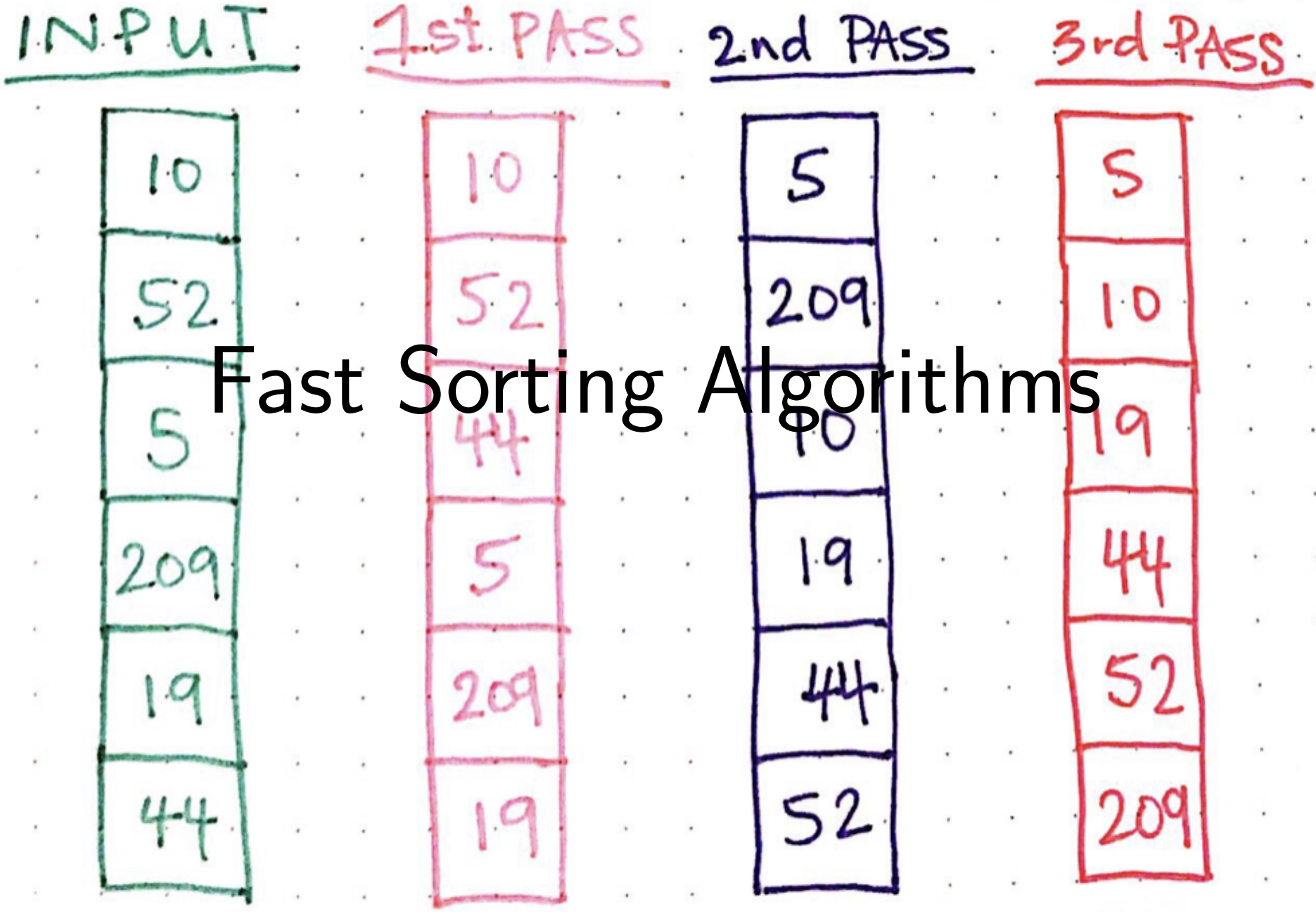
Fast Sorting Algorithms



Sorting algorithms on values in a small range

CLRS Ch.8

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

The algorithms will sort an array $A[n]$ of integers in the range $[0, r]$.

The complexity of the algorithms depends on both n and r.

For adequate values of r, the algorithms have cost $O(n)$ or below $O(n \log n)$.

Counting sort

The counting sort algorithm, consider all possible values $i \in [0, r]$.
For each of them it counts how many elements in A are smaller or equal to i.
Finally, this information is used to place the elements in the right order.

Counting sort

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

CountingSort (A, r)
for $i = 0$ to r **do**
  $C[i] = 0$
for $i = 0$ to $n-1$ **do**
  $C[A[i]] = C[A[i]] + 1$
for $i = 1$ to r **do**
  $C[i] = C[i] + C[i-1]$
for $i = n-1$ downto 0 **do**
  $B[C[A[i]]] = A[i]$; $C[A[i]] = C[A[i]] - 1$

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**
  $B[C[A[i]]] = A[i]$; $C[A[i]] = C[A[i]] - 1$ $\{O(n)\}$

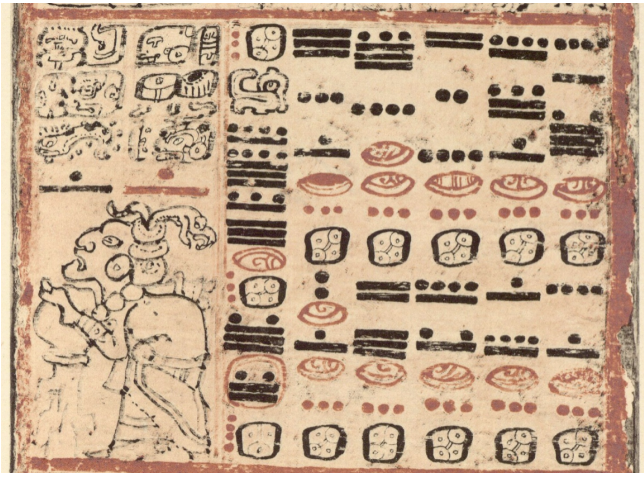
Complexity: $T(n) = O(n + r)$ if $r = O(n)$, then $T(n) = O(n)$.

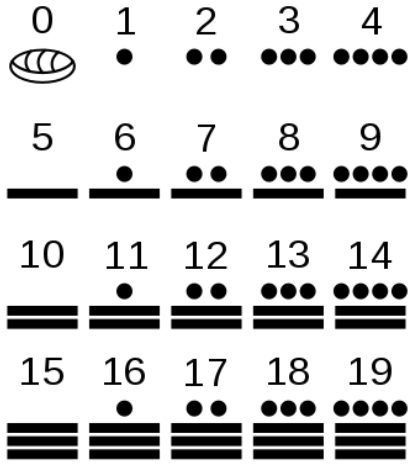
Radix sort: What does radix mean?

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 $\rightarrow$ decimal:
  $1011 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 11$
- To convert an enter from decimal $\rightarrow$ 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

More examples

radix 10		radix 2	radix 16
7134785012		11010100101000100001011011110100	1a9442df4
4561343780		10000111111100000100101010010100	10fe09524
0051889437	.	00000001100010111110001010001101	0317c51d

Question in a past exam: Is it true that if an integer a in radix 256 has d digits, and we change it to be expressed in radix 2, the number of bits that n would have is $\Theta(d^2)$?

NO. When converting radix-256 to binary, we increase d to $(\lg 256) \times d = 8d = \Theta(d)$.
Ex.: (251, 180) $\rightarrow$ 1000110110101101000

RADIX sort

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.
For instance, Heap sort and Quicksort are not stable. Merge sort can be implemented to become stable.

Given an array A with n keys, 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 to sort A according to the i-th digit values.

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

329		720		720		329
475		475		329		355
657		355		436		436
839	$\Rightarrow$	436	$\Rightarrow$	839	$\Rightarrow$	475
436		657		355		657
720		329		657		720
355		839		475		839

Theorem (Correctness of RADIX)

RADIX LSD sorts correctly n given keys.

Induction on d.

Base: If $d = 1$ the stable sorting algorithm sorts correctly.

HI: 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 $a_d = b_d$ then as we are using a stable sorting a and b will remain in the same order, which by hypothesis was already the correct one.

□

Complexity of RADIX

Given n integers ≥ 0, each integer with at most d digits, and each digit in the range 0 to b, if we use counting sorting:
$T(n, d) = \Theta(d(n + b))$.

- Consider that each integer (in base 10) has a value up to $f(n)$.
- Then the number of digits is $d = \lceil \log_b f(n) \rceil$, so $T(n, d) = O(\log f(n)(n + b))$,
- if $\log f(n) = \omega(1)$ then $T(n) = \omega(n)$. So in this case RADIX is not lineal.

Can we tune the parameters? Yes, select the best radix to express the integers.

RADIX: selecting the base

For numbers in binary, we can select basis that are powers of 2. This simplifies the computation as we have only to look to pieces of bits.

For ex., if we have integers of $d = 64$ bits, and take $r = 3$; $2^3 = 8$-bit integers, with $\hat{d} = 4$ new digits per integer.

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 $1 < r < d$ to use radix 2^r.

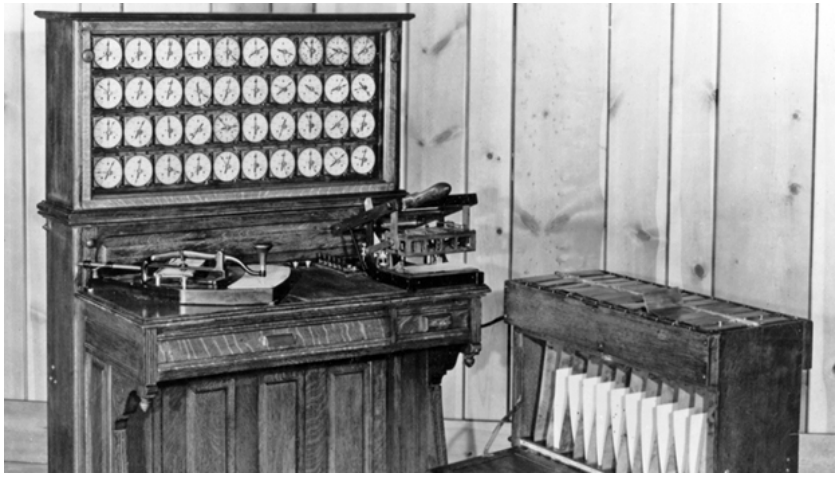
Running RADIX LSD with base 2^r the new $\hat{d} = \lceil d/r \rceil$ digits, and the bound is

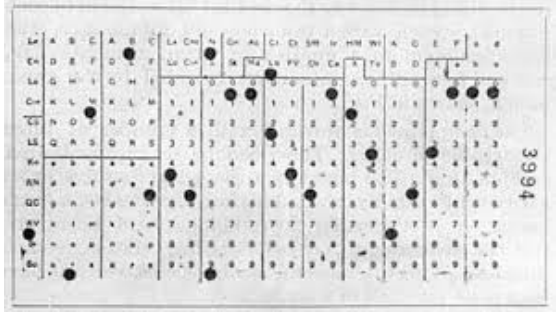
$\Rightarrow T(n) = \Theta(\hat{d}(n + 2^r)) = \Theta((d/r)(n + 2^r))$.

The best choice of r is roughly $c \lg n$, for $c < 1$, so that $2^r < n$, but take into account that with this selection the used counting sort has to operate on numbers with $O(\lg n)$ bits and we cannot assume that the operation takes constant time.

A bit of history.

Radix and all counting sort 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 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 if time $< L(n)$, for any input e,$|e| = n$.

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: $\exists A, \forall x$ time $A(x) \leq T(|x|)$,
- Lower bound: $\forall A, \exists x$ time $A(x) \geq L(|x|)$,

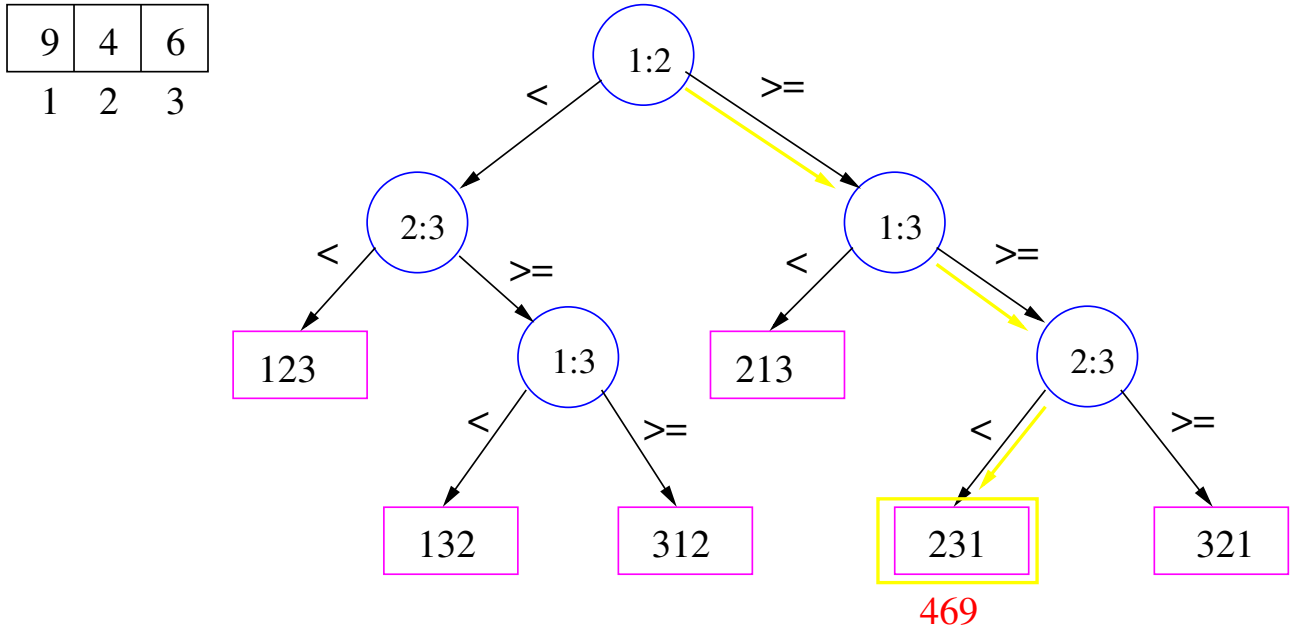
To prove an upper bound: produce an A so that the bound holds for any input x ($n = |x|$).

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.

Use a decision tree: A binary tree where,

- each internal node represents a comparison $a_i : a_j$, the left subtree represents the case $a_i \leq a_j$ and the right subtree represents the case $a_i > a_j$
- each leaf represents one of the $n!$ possible permutations $(a_{\pi(1)}, a_{\pi(2)}, \ldots, a_{\pi(n)})$. Each of the n permutations must appear as one of the leaves of the tree



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,

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