

INPUT

10
52
5
209
19
44

1st PASS

10
52
44
5
209
19

2nd PASS

5
209
10
19
44
52

3rd PASS

5
10
19
44
52
209

Linear Sorting

# Upper and lower bounds on time complexity of a problem.

A problem has a **time upper bound**  $T_U(n)$  if there is an algorithm  $A$  such that **for any input**  $e$  of size  $n$ :  
 $A(e)$  gives the correct answer in  $\leq T_U(n)$  steps.

A problem has a **time lower bound**  $T_L(n)$  if **there is NO** algorithm which solves the problem if time  $< T_L(n)$ , **for any input**  $e, |e| = n$ .

It may be that an algorithm solves the problem faster than  $T_L(n)$  for a specific input.

**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:  $\exists A, \forall e \text{ time } A(e) \leq T_U(|e|),$
- ▶ Lower bound:  $\forall A, \exists e \text{ time } A(e) \geq T_L(|e|),$

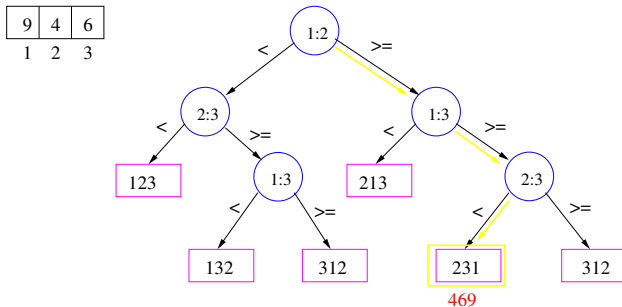
To prove an upper bound: produce an  $A$  which works for any  $e$ ,  
 $|e| = n$ .

To prove a lower bound, show that **for any possible algorithm**, the time on an input is greater than 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**  $(a_{\pi(1)}, a_{\pi(2)}, \dots, a_{\pi(n)})$ . Each of the  $n$  permutations must appear as one of the leaves of the tree



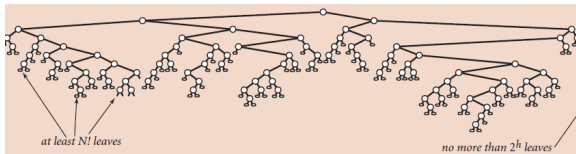
## Theorem

Any comparison sort that sorts  $n$  elements must 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,  
 $n! \leq 2^h \Rightarrow h \geq \lg(n!) > \lg\left(\frac{n}{e}\right)^n = \Omega(n \lg n)$ .



# Linear sorting: Counting sort

## CLRS Ch.8

Assume the input  $A[1 \dots n]$ , is an array of integers in  $[0, b]$ .

Need:  $B[1 \dots n]$  as the output and  $C[0 \dots b]$  as scratch.

**Counting** ( $A, b$ )

**for**  $i = 0$  to  $b$  **do**

do  $C[i] = 0$

**end for**

**for**  $i = 1$  to  $n$  **do**

do  $C[A[i]] = C[A[i]] + 1$

**end for**

**for**  $i = 0$  to  $b$  **do**

do  $C[i] = C[i] + C[i - 1]$

**end for**

**for**  $i = n$  downto  $1$  **do**

do  $B[C[A[i]]] = A[i]$

$C[A[i]] = C[A[i]] - 1$

**end for**

**Counting** ( $A, b$ )

**for**  $i = 0$  to  $b$  **do**

do  $C[i] = 0$  {  $O(b)$ }

**end for**

**for**  $i = 1$  to  $n$  **do**

do  $C[A[i]] = C[A[i]] + 1$  {  $O(n)$ }

**end for**

**for**  $i = 0$  to  $b$  **do**

do  $C[i] = C[i] + C[i - 1]$  {  $O(b)$ }

**end for**

**for**  $i = n$  down to  $1$  **do**

$B[C[A[i]]] = A[i]$  {  $O(n)$ }

$C[A[i]] = C[A[i]] - 1$

**end for**

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

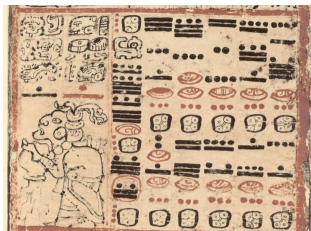
When using counting on short arrays,  $b = 10$ .

# What does it mean radix?

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 enter from binary  $\rightarrow$  decimal:

$$1011 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 12 \Rightarrow \mathbf{11}$$

- ▶ To convert an enter from decimal  $\rightarrow$  binary: Repeatedly dividing the enter by 2 will give a result plus a remainder:

$$19 \Rightarrow \underbrace{19/2}_{1} \underbrace{9/2}_{1} \underbrace{4/2}_{0} \underbrace{2/2}_{01} = \mathbf{10011}$$

- ▶ 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 need **a maximum** of  $\lg 16 \times 4 = 16$  bits

In the above example,  $(4CF5)_{16}$  in binary is 11001111

## More examples

radix 10	radix 2	radix 16
7134785012	110101001010001000010110111110100	1a9442df4
4561343780	100001111111000001001010100100100	10fe09524
0051889437	000000011000101111100010100011101	0317c51d

Ex. from past exams: 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  $a$  would have is  $\Theta(d^2)$ ?

NO. When converting radix-256 to binary, we increase  $d$  to  $(\lg 256) \times d = 9d = \Theta(d)$ .

## Linear sorting: 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 or Quicksort is not stable.

Given an array  $A$  with  $n$  keys, each one with  $d$  digits, the Radix (Least Significant Digit),

**RADIX LSD** ( $A, d$ )

**for**  $i = 1$  to  $d$  **do**

    Use stable sorting to sort the  $i$ -th digit of  $A$ .

**end for**

## Example

329		720		720		329
475		475		329		355
657		355		436		436
839	$\Rightarrow$	436	$\Rightarrow$	839	$\Rightarrow$	457
436		657		355		657
720		329		657		720
355		839		475		839

## Theorem (Correctness of RADIX)

*The previous algorithm sort correctly  $n$  keys.*

### Induction on $d$ .

If  $d = 1$  the stable sorting works. Assume it is true for  $d - 1$ ,  
to sort the  $d$ -th digit,

if  $a_d < b_d$  then  $a$  will be placed before  $b$ ,

if  $b_d < a_d$  then  $b$  will be placed before  $a$ ,

if  $b_d = a_d$  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  $\geq 0$ , each integer with at most  $d$  digits, and each digit in the range 0 to 9, if we use counting sorting:

$$T(n, d) = \Theta(d(n + 9)).$$

- ▶ Consider that each integer (base 10) has a value up to  $f(n)$ .
- ▶ Then the number of digits is  $d = \lfloor \log f(n) \rfloor + 1 \sim \log n$ , so  $T(n, d) = \Theta((\log f(n))(n + 9))$ ,
- ▶ if  $f(n) = \omega(1)$  then  $T(n) = \omega(n)$ . So in this case RADIX is not linear.

**Can we do it faster?:** Yes, change the radix to express the integers.

# RADIX

Consider each integers in radix  $b$ , and max value  $f(n)$ , then the number of "digits" is  $d = \log_b f(n)$ , and apply RADIX to the columns of new digits in base  $b$ .

Complexity: Given  $n$  positive integers, each integer with a maximal value of  $f(n)$  and with at most  $d$  digits needed to represent each integer using radix  $b$ . Then the complexity  $T(n)$  of using RADIX to sort the  $n$  integers is:

$$T(n) = O((n + b)d) = O(n + b) \log_b f(n).$$

# RADIX Change: Example

RADIX 10	RADIX 2	RADIX 16
7134785012	110101001010001000010110111110100	1a9442df4
4561343780	100001111111000001001010100100100	10fe09524
0051889437	000000011000101111100010100011101	0317c51d

	n	d	b
RADIX 10	3	10	9
RADIX 2	3	33	1
RADIX 16	3	8	15



# Implementation for bit integers

Given  $n$  integers given as binary strings with length  $d$ ,  
we want to choose an integer  $1 < r < d$  such minimizes  
 $(d/r)(n + 2^r)$ .

Then use RADIX on each of the new  $\hat{d} = \lceil d/r \rceil$  digits.

For ex., if we have words of  $b = 64$  bits, which we split in  $d = 6$   
 $r = 8$ -bit digits:

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

Each new digit is an integer with  $b = 2^r - 1$ .

So we can use counting sort with  $k = 2^r - 1$ .

Each pass of counting sort takes  $\Theta(n + k) = \Theta(n + 2^r)$ ,  
as there are  $d$  passes  $\Rightarrow T(n) = \Theta(d(n + 2^r)) = \Theta((b/r)(n + 2^r))$ .

# Comparing radix and counting:

For  $n$  integers, each integer with at most  $d$  digits, where each digit is in the range  $[0, 9]$ :

- ▶ Counting sort is  $\Theta(9dn)$ ,
- ▶ Radix with the choice of  $r = \log_9 n$ -digits can sort  $n$   $d$ -digit numbers in  $\frac{d}{\log_9 n} \Theta(9n)$ .

Consider 2000 integers of 32 bits each:

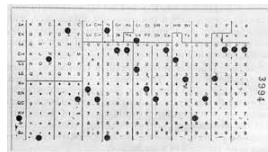
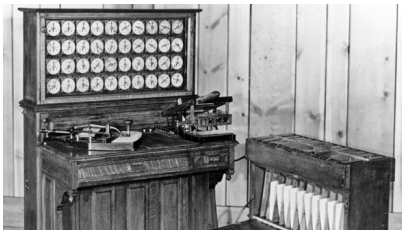
- ▶ Quicksort needs to do  $\lg 2000 = 11$  passes over the data,
- ▶ Radix sort with digits of 11-bits, takes 3 passes (at each one counting sort makes 2 passes).

Empirically, when dealing with natural numbers, radix is better than other sorting methods for values of  $n > 2000$ .

# A bit of history.

Radix and all counting sort are due to Herman Hollerith.

In 1890 he invented the card sorter that allowed to shorten the US census to 5 weeks, using punching cards.

A photograph of a single punch card from the 1890 US Census. The card is a grid of 12 columns and 12 rows. Each cell contains a number from 0 to 9. Some cells have a small hole punched through them. The card is labeled '3934' on the right side.

1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	0	1	2
3	4	5	6	7	8	9	0	1	2	3	4
5	6	7	8	9	0	1	2	3	4	5	6
7	8	9	0	1	2	3	4	5	6	7	8
9	0	1	2	3	4	5	6	7	8	9	0
0	1	2	3	4	5	6	7	8	9	0	1
1	2	3	4	5	6	7	8	9	0	1	2
2	3	4	5	6	7	8	9	0	1	2	3
3	4	5	6	7	8	9	0	1	2	3	4
4	5	6	7	8	9	0	1	2	3	4	5
5	6	7	8	9	0	1	2	3	4	5	6
6	7	8	9	0	1	2	3	4	5	6	7
7	8	9	0	1	2	3	4	5	6	7	8
8	9	0	1	2	3	4	5	6	7	8	9
9	0	1	2	3	4	5	6	7	8	9	0