Project Planning

You are investigating how the execution time of a sorting algorithm changes with the size of the input by instrumenting a Java program running on your laptop. You have available a function $currentTime()$ that reports the current time in microseconds; you are using this function to time executions of the sort for arrays of various sizes.

1. When the input array is of size less than about 1000, you notice that the difference between the $currentTime()$ at the start and the end of the sort is either 0 μs or 1 μs. Explain why this happens, and what you do to get more reliable numbers.

2. What sizes of arrays would you use to assess the sort algorithm? Why did you pick those sizes? How would you decide on the maximum size to measure?

3. How would you guard against the possibility of your sorting program containing a bug that might make your timings meaningless?

4. To see how reproducible your measurements are, you arrange for each of your test programs to sort 10 copies of the same random array; you find that for some programs, 8 or 9 of the runs yield timings within a few percent of each other, while the remaining 1 or 2 runs take vastly longer, sometimes by a factor of 3 or 5. Explain why this happens, and what you do with the data to more accurately assess the behavior of the sort.

CS 350 Algorithms and Complexity

Winter 2019

Lecture 11: Space & Time Tradeoffs.

Sorting by Counting, and String search

Andrew P. Black

Department of Computer Science

Portland State University

But first … the episode you missed

- Transform and conquer
  - instance simplification
  - representation change
  - problem reduction

Instance Simplification

- Transform a problem instance to another instance with a special property

- List Presorting
  - many algorithms are faster on sorted lists
- Gaussian Elimination
  - Transform the argument matrix into upper-triangular form
    - Can be used to compute a determinant
- Balancing AVL trees
  - "rotate" the tree to restore the balance criterion

Representation Change

- Find a better way to represent the data in your problem
  - binary search tree → 2-3 tree
  - Horner's rule
  - heaps, and heapsort
    - a heap is an <u>essentially complete binary tree</u> implemented as an array
  - Exponentiation by exploiting the binary representation of a number

Problem Reduction

- A *reduction* is transforming a problem that you don't know how to solve into one that you can
  - Find LCM by computing the GCD
  - Solve geometric problems by reducing to algebra
  - Optimization problems reduced to linear programming

Why did I miss Chapter 6?

- If you are mathematically inclined, you will already know about Gaussian Elimination
- You covered Heaps in your data-structures course
- 2-3 trees are a special case of B-trees, which we will discuss soon
- AVL-trees are too complicated! Hashing works as well or better.
  - Compare them for your project?

Space-for-time tradeoffs

- Two varieties of space-for-time algorithms:
- *input enhancement*: preprocess the input (all or part) to store some info to be used later in solving the problem
  - counting sorts
  - string searching algorithms
- *prestructuring* — preprocess the input to make accessing its elements easier
  - hashing
  - indexing schemes (*e.g.*, B-trees)

Question:

- Does saving space always cost time?

A. Yes

B. No

Examples:

- Traversing a compact representation may be faster than traversing a larger one
  - Sparse arrays
  - Sets as lists versus hash tables
  - Adjacency list vs Adjacency Matrix for graphs

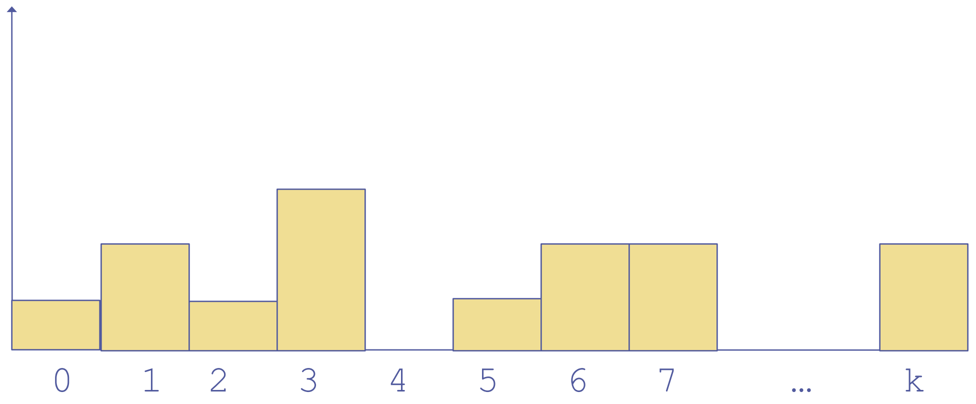
Question:

- How to choose the right data structure?

A. Guess
B. Read the book
C. Understand the tradeoffs between the various operations
D. Understand the frequency of the various operations
E. All of the above
F. B, C & D
G. None of the above

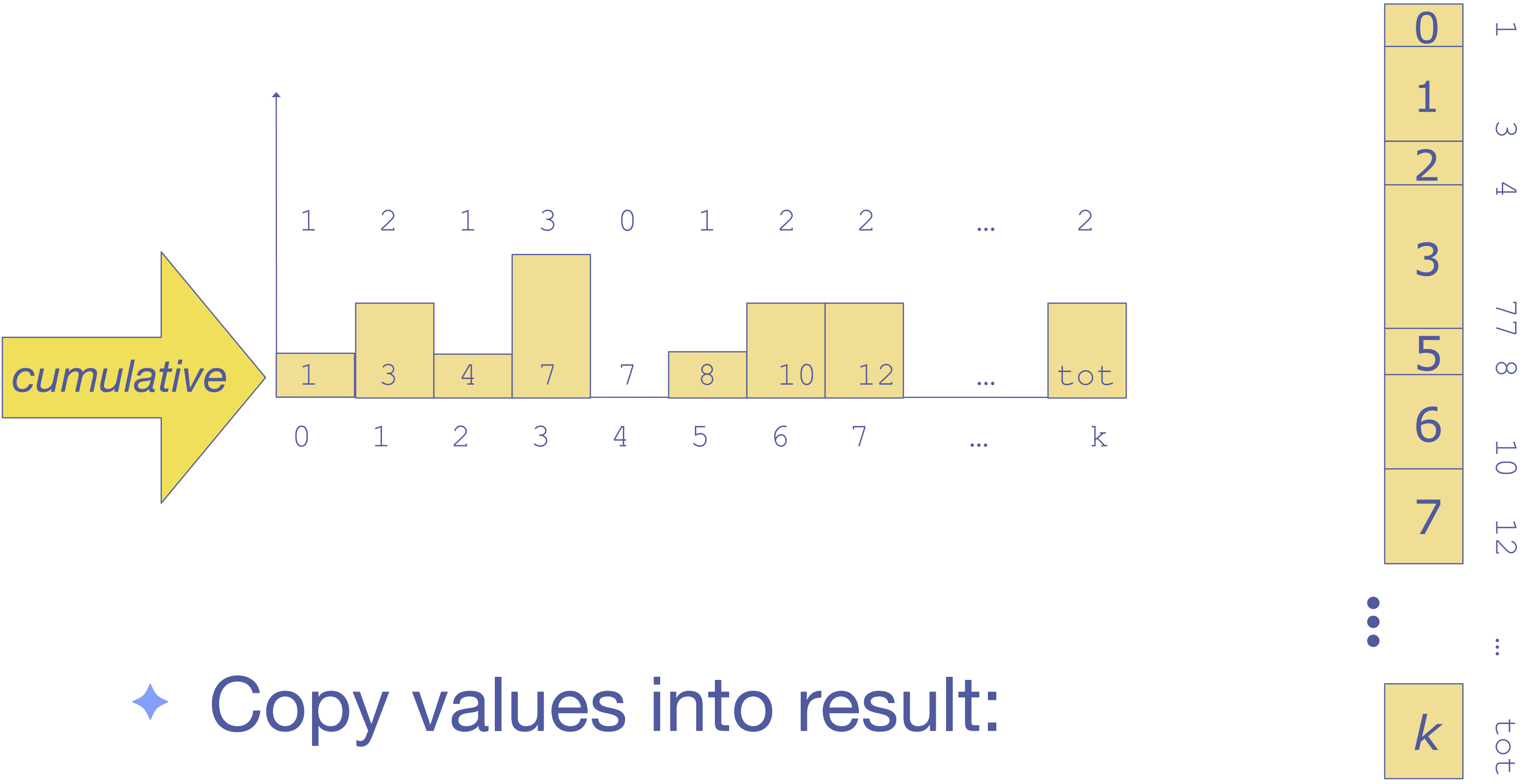
Distribution Counting Sort:

- Suppose that the values to be sorted are all in the range $0..k$, where k is some (small) integer.
- Pre-process the input:
  - Make a histogram to count how many times each key occurs:



Counting sort, continued:

- Calculate *cumulative* totals:



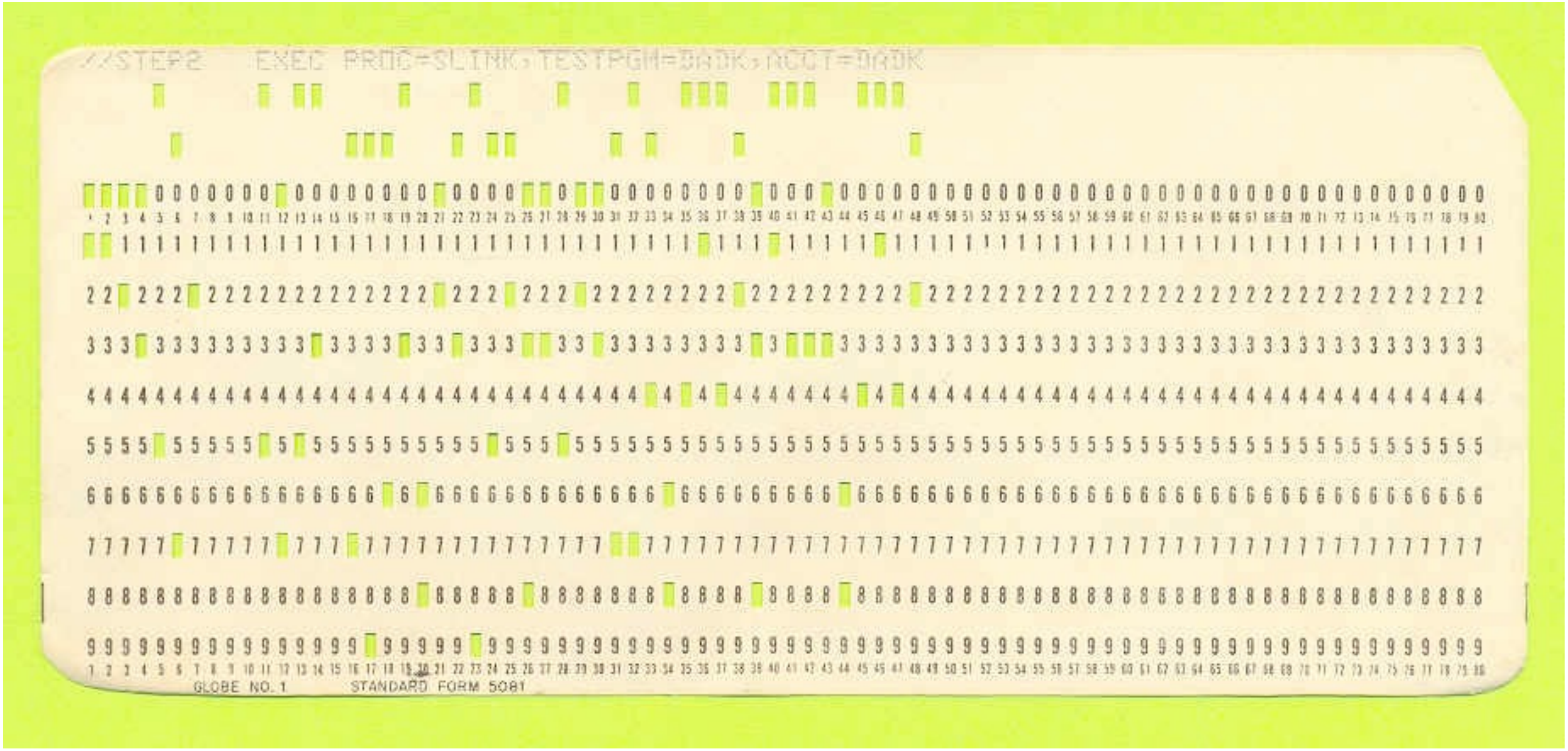
- Copy values into result:

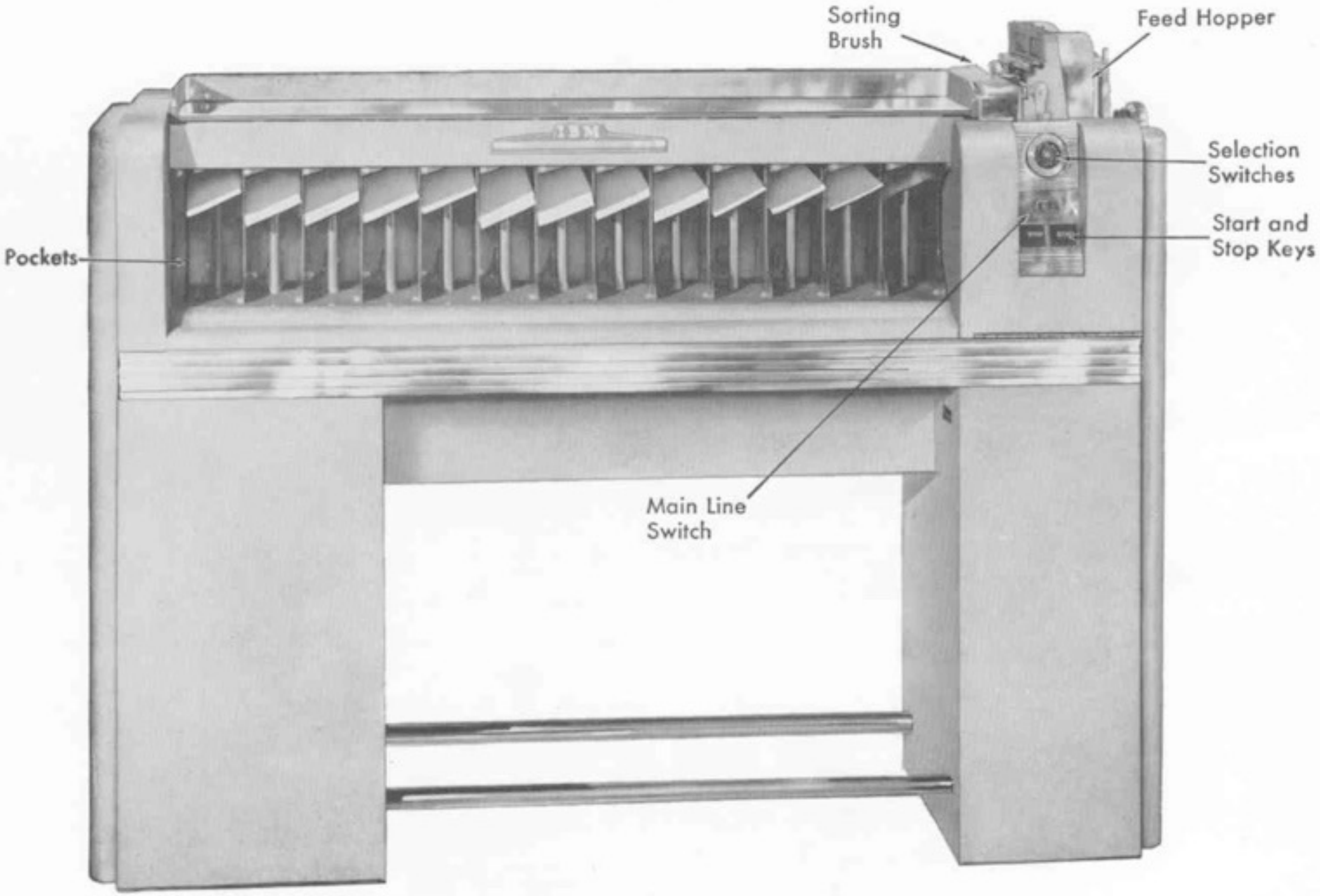
Implementation:

```
for (int i=0; i<=k; i++)      // set counts to zero
    count[i] = 0;

for (int i=0; i<n; i++)       // build histogram
    count[a[i]]++;

for (int i=1; i<=k; i++)      // cumulative totals
    count[i] += count[i-1];

for (int i=n-1; i>=0; i--) {  // copy data to target
    target[count[a[i]]] = a[i];
    count[a[i]]--;
}
```

All steps O(n)

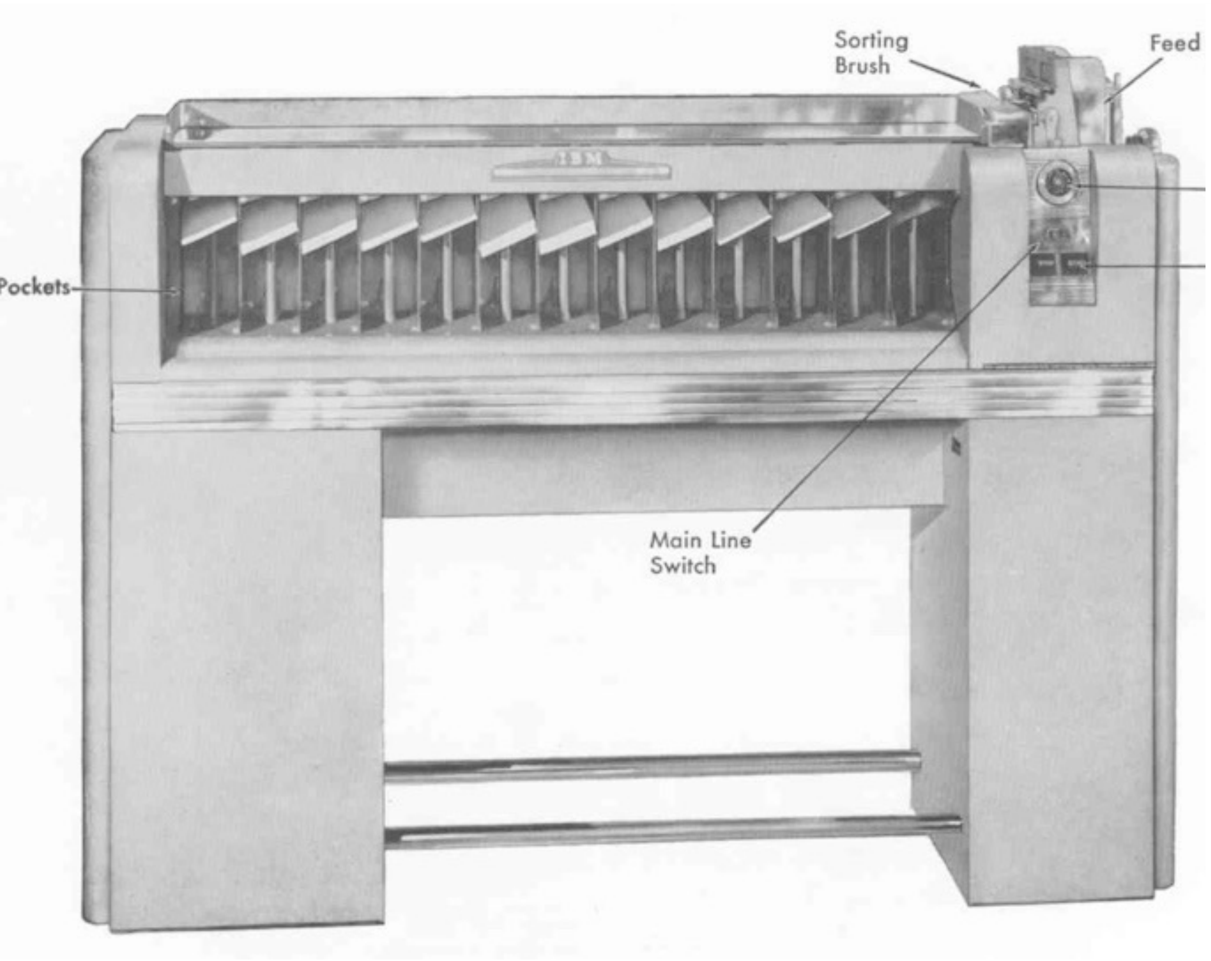
Punch-card computing




Sorting Brush
Feed Hopper
Selection Switches
Start and Stop Keys
Pockets
Main Line Switch

Each carton hods 2000 cards

IBM Model 82 Card Sorter (1949)



- Deals cards from a feed into 13 output pockets. (one pocket for rejects)
- 250-2000 cards per minute, depending on model.
- One column, selected by the Sorting Brush, sorted per pass.
- The Selector Switches determine which rows are included in the sort
- To fully sort a deck of cards required multiple passes through the sorter—one pass per column in the sort key.

Review: String searching by brute force

- pattern: a string of *m* characters to search for
- text: a (long) string of *n* characters to search in

- Brute force algorithm

  Step 1 Align pattern at beginning of text

  Step 2 Moving from left to right, compare each character of pattern to the corresponding character in text until either all characters are found to match (successful search) or a mismatch is detected

  Step 3 While a mismatch is detected and the text is not yet exhausted, realign pattern one position to the right and repeat Step 2

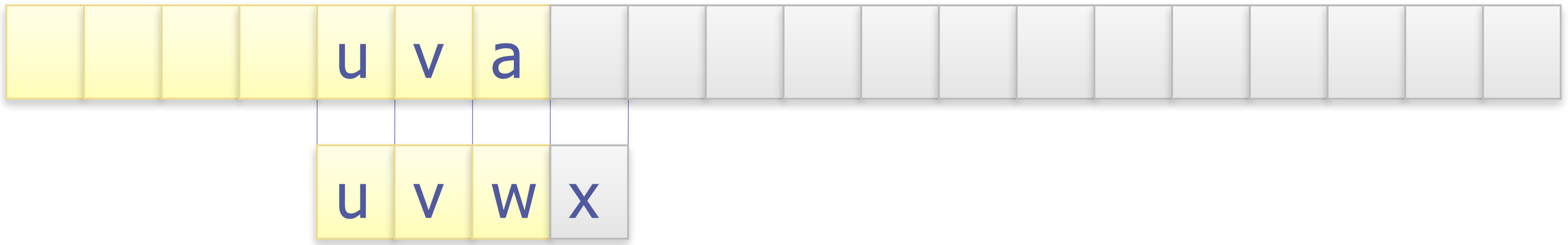
String searching by preprocessing

- Several string searching algorithms are based on the input enhancement idea of preprocessing the pattern
  - Knuth-Morris-Pratt (KMP) algorithm preprocesses pattern left to right to get useful information for later searching
  - Boyer-Moore algorithm preprocesses pattern right to left and store information into two tables
  - Horspool's algorithm simplifies the Boyer-Moore algorithm by using just one table

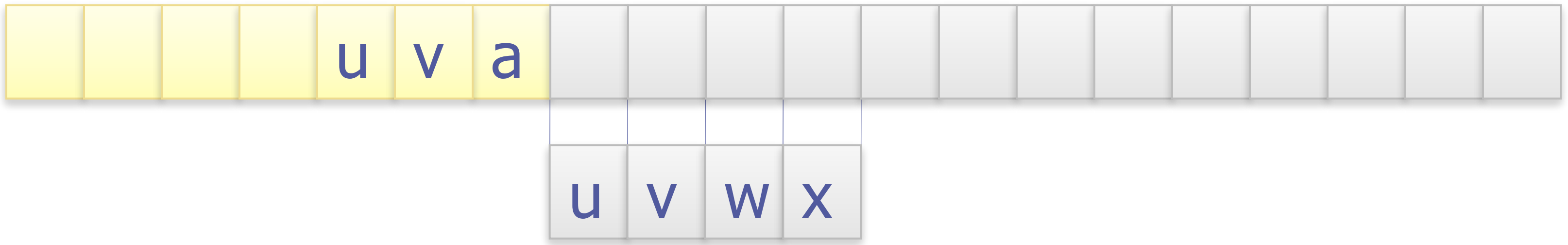
Key idea:

- When a match fails, you learn something!

- For example when the following test fails after matching two characters …



- … and given this specific pattern string, the next possible match can’t occur until the following position:



Horspool's Algorithm

- A simplified version of Boyer-Moore algorithm:
  - preprocesses pattern to generate a *shift table* that determines how much to shift the pattern when a mismatch occurs
  - makes a shift based on the text's character c aligned with the last character in the pattern. Uses the shift table's entry for c

How far to shift?

Look at first (rightmost) character in text that was compared:

1. The character is not in the pattern

```
.....c...................... (c not in pattern):
BA0BAB
```

2. The character is in the pattern (but not the rightmost):

```
.....0...................... (0 occurs once in pattern)
BA0BAB

.....A...................... (A occurs twice in pattern)
BA0BAB
```

3. The rightmost characters match:

```
.....B......................
BA0BAB
```

Shift table

- Shift sizes can be precomputed by the formula:
  $t(c)$ = **if** c is not in first m-1 characters of pattern
  then m
  else distance from c's rightmost occurrence in pattern to right end of pattern
- scan pattern before search begins
  - store $t(c)$ in a table called the shift table
  - Shift table is indexed by (text and pattern) alphabet. *e.g.*, for BAOBAB:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
1	2	6	6	6	6	6	6	6	6	6	6	6	6	3	6	6	6	6	6	6	6	6	6	6	6	6

Example of running Horspool's alg.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
1	2	6	6	6	6	6	6	6	6	6	6	6	6	3	6	6	6	6	6	6	6	6	6	6	6	6

```
BARD LOVED BANANAS
BAOBAB
      BAOBAB
        BAOBAB
              BAOBAB  (unsuccessful search)
```

Question

- How many character comparisons will be made by Horspool's algorithm, in searching in a text of 1000 zeros for 00001

   A. 1000
   B. 200
   C. 250
   D. 996

Question

- How many character comparisons will be made by Horspool's algorithm, in searching in a text of 1000 zeros for 10000

  A. 1000
  B. 996
  C. $5 \times 996 = 4980$
  D. $5 \times 1000 = 5000$

Question

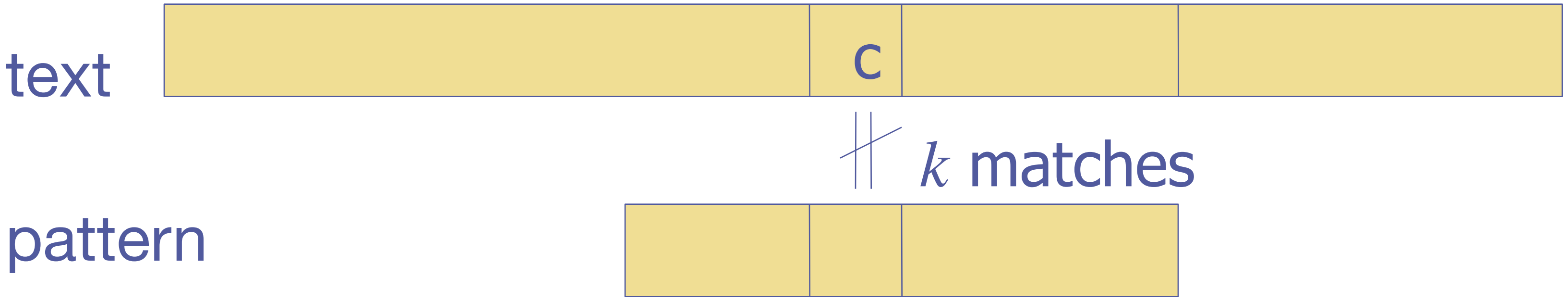
- Is it possible for Horspool’s Algorithm to make more comparisons than the Brute-force algorithm?

Boyer-Moore algorithm

- Based on same two ideas:
  - comparing pattern characters to text from right to left
  - precomputing shift sizes
    - but using *two* tables
      - *bad-symbol table* indicates how much to shift based on the text character that caused a mismatch
      - *good-suffix table* indicates how much to shift based on matched part (suffix) of the pattern

Bad-symbol shift in Boyer-Moore algorithm

- If the rightmost character of the pattern *doesn't* match, B-M algorithm acts just like Horspool's
- If the rightmost character of the pattern *does* match, BM compares preceding characters right to left until either all pattern's characters match, or a mismatch on text's character c is encountered after $k > 0$ matches



bad-symbol shift $d_1 = \max\{t(c)-k, 1\}$

Good-suffix shift in Boyer-Moore algorithm

- Good-suffix shift d_2 is applied after $0 < k < m$ characters were matched
- $d_2(k)$ = the distance between matched suffix of size k and its rightmost occurrence in the pattern that is not preceded by the same character as the suffix

  Example: CABABA $d_2(1) = 4$

- If there is no such occurrence, match the longest part of the k-character suffix with corresponding prefix;
  if there are no such suffix-prefix matches, $d_2(k) = m$

  Example: WOWWOW: $d_2(2) = 5$, $d_2(3) = 3$, $d_2(4) = 3$, $d_2(5) = 3$

Boyer-Moore Algorithm

After matching successfully $0 < k < m$ characters, the algorithm shifts the pattern right by

$$d = \max(d_1, d_2(k))$$

where $d_1 = \max(t(c)-k, 1)$ is bad-symbol shift

$d_2(k)$ is good-suffix shift

Example: Find pattern AT_THAT in

WHICH_FINALLY_HALTS. _ _ AT_THAT

Boyer-Moore Algorithm (cont.)

Step 1 Fill in the bad-symbol shift table

Step 2 Fill in the good-suffix shift table

Step 3 Align the pattern against the beginning of the text

Step 4 Repeat until a matching substring is found or text ends:

Compare the corresponding characters *right to left*.

(a) If no characters match, retrieve entry $t(c)$ from the bad-symbol table for the text's character c causing the mismatch and shift the pattern to the right by $t(c)$.

(b) If $0 < k < m$ characters are matched, retrieve entry $t(c)$ from the bad-symbol table for the text's character c causing the mismatch and entry $d_2(k)$ from the good-suffix table and shift the pattern to the right by

$$d = \max\{d_1, d_2(k)\}$$

where $d_1 = \max\{t(c)-k, 1\}$.

Example of Boyer-Moore algorithm

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	␣
1	2	6	6	6	6	6	6	6	6	6	6	6	6	3	6	6	6	6	6	6	6	6	6	6	6	6

B E S S ␣ K N E W ␣ A B O U T ␣ B A O B A B S

B A O B A B

$d_1 = t(\text{K}) = 6$

B A O B A B

$d_1 = t(␣)-2 = 4$

$d_2(2) = 5$

B A O B A B

$d_1 = t(␣)-1 = 5$

$d_2(1) = 2$

B A O B A B (success)

k	pattern	d_2
1	BAOBAB	2
2	BAOBAB	5
3	BAOBAB	5
4	BAOBAB	5
5	BAOBAB	5

Question:

- Would the Boyer-Moore algorithm work correctly with just the bad-symbol table to guide pattern shifts?

A. Yes

B. No

Question:

- Would the Boyer-Moore algorithm work correctly with just the good-suffix table to guide pattern shifts?

A. Yes

B. No

Question:

If the last characters of a pattern and its counterpart in the text *do* match, does Horspool's algorithm have to check other characters right to left, or can it check them left to right too?

A. must check right-to-left

B. could also check left-to-right

Question:

If the last characters of a pattern and its counterpart in the text do match, does The Boyer-Moore algorithm have to check other characters right to left, or can it check them left to right too?

A. Yes
B. No

Question

- How many character comparisons will be made by the Boyer-Moore algorithm, in searching in a text of 1000 zeros for 00001

A. 1000
B. 200
C. 250
D. 996

Question

- How many character comparisons will be made by the Boyer-Moore algorithm, in searching in a text of 1000 zeros for 10000

   A. 1000
   B. 996
   C. $5 \times 996 = 4980$
   D. $5 \times 1000 = 5000$

Question

- Is it possible for the Boyer-Moore Algorithm to make more comparisons than the Brute-force algorithm?