CS61B Lectures #28

s on sorting by comparison

counting, radix sorts

ay: *DS(IJ)*, Chapter 8; Next topic: Chapter 9.

Better than N lg N?

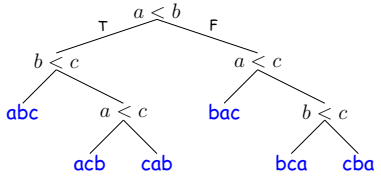
at *if all you can do to keys is compare them,* then sorting $N \lg N)$.

there are $N!$ possible ways the input data could be

our program must be prepared to do $N!$ different com- data-moving operations.

here must be $N!$ possible combinations of outcomes of sts in your program, since those determine what move where (we're assuming that comparisons are 2-way).

ree
ting time



Necessary Choices

-test goes two ways, number of possible different out- if-tests is 2^k.

nough tests so that $2^k \geq N!$, which means $k \in \Omega(\lg N!)$.

g's approximation,

$$\sqrt{2\pi N}\left(\frac{N}{e}\right)^N\left(1+\Theta\left(\frac{1}{N}\right)\right),$$

$$1/2(\lg 2\pi + \lg N) + N\lg N - N\lg e + \lg\left(1+\Theta\left(\frac{1}{N}\right)\right)$$

$$\Theta(N \lg N)$$

that k, the worst-case number of tests needed to sort omparison sorting, is in $\Omega(N \lg N)$: there must be cases ed (some multiple of) $N \lg N$ comparisons to sort N

Beyond Comparison: Distribution

can do more than compare keys?

, how can we sort a set of N integer keys whose values) to kN, for some small constant k?

ue: put the integers into N buckets, with an integer p ket $\lfloor p/k \rfloor$.

ys per bucket, so catenate and use insertion sort, which ast.

$= 10$:

10 13 4 2 19 17 0 9

s:

2 | 4 | | 9 | 10 | 13 | 14 | 17 | 19 |

n sort is fast. Putting in buckets takes time $\Theta(N)$, and t takes $\Theta(kN)$. When k is fixed (constant), we have ne $\Theta(N)$.

Distribution Counting

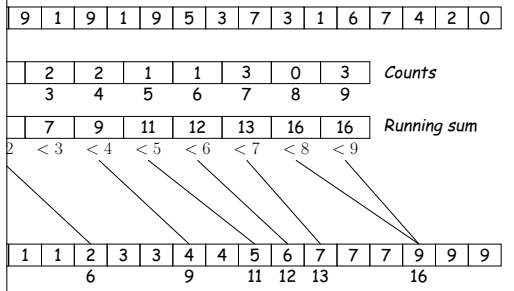
nique: *count* the number of items < 1, < 2, etc.

ems with value $< p$, then in sorted order, the j^{th} item must be item #$M_p + j$.

r *linear-time* algorithm.

Distribution Counting Example

tems are between 0 and 9 as in this example:



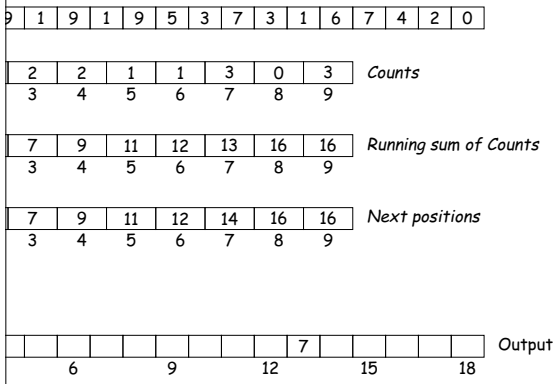
gives # occurrences of each key.

" gives cumulative count of keys $<$ each value...

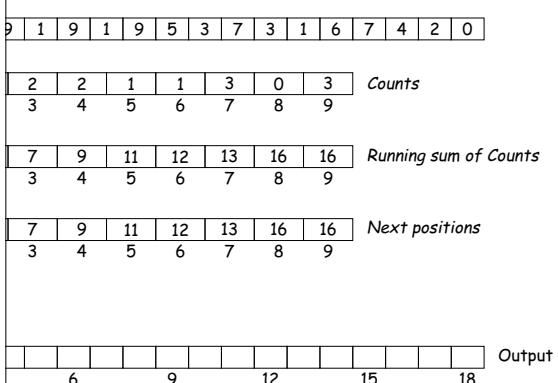
s us where to put each key:

tance of key k goes into slot m, where m is the number ces that are $< k$.

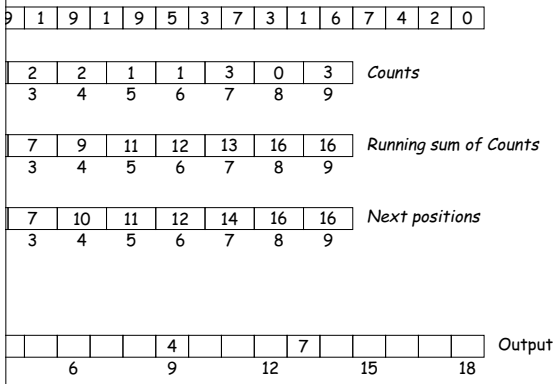
istribution Counting Example (II)



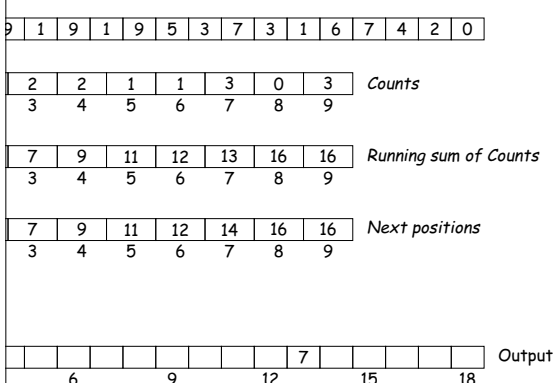
istribution Counting Example (II)



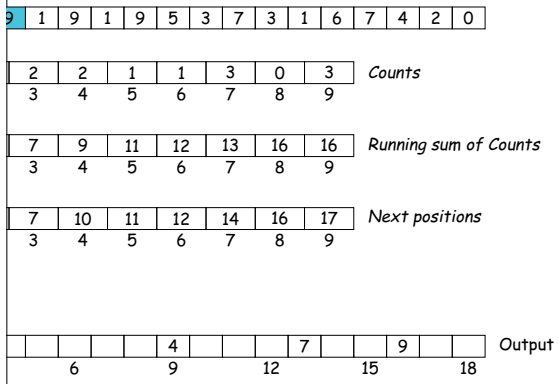
istribution Counting Example (II)



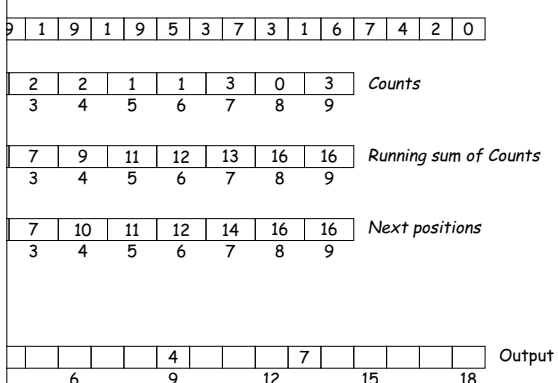
istribution Counting Example (II)



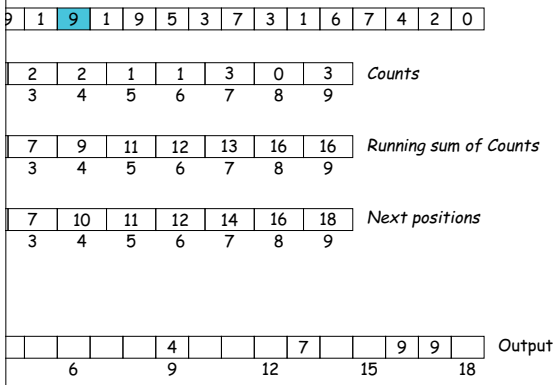
istribution Counting Example (II)



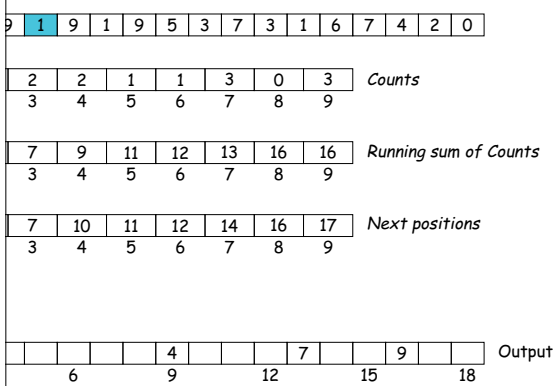
istribution Counting Example (II)



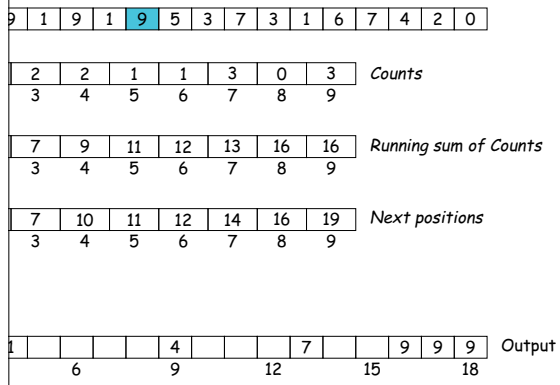
istribution Counting Example (II)



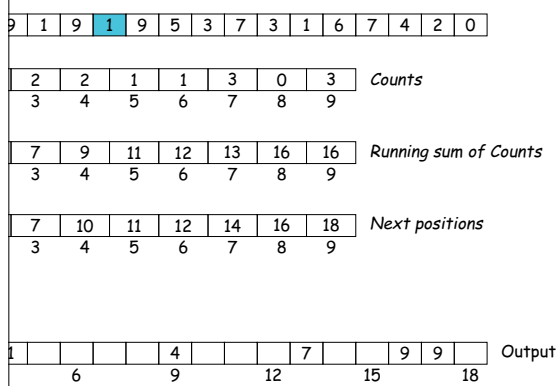
istribution Counting Example (II)



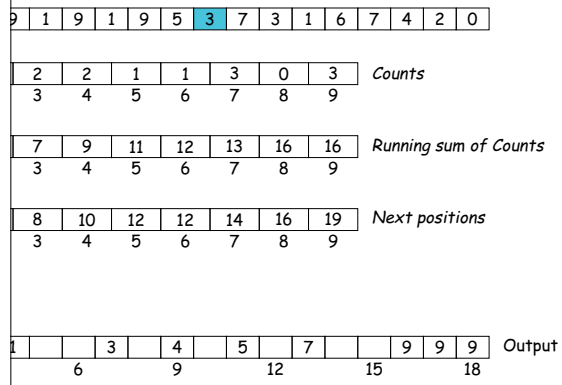
istribution Counting Example (II)



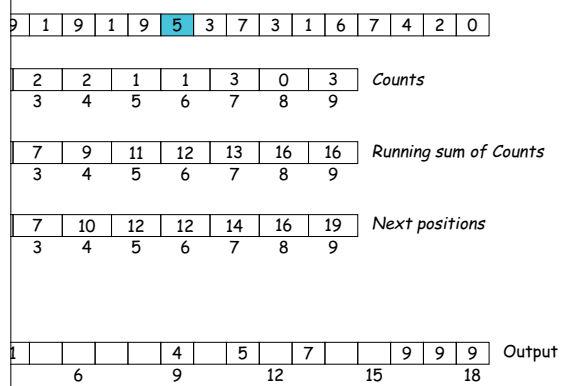
istribution Counting Example (II)



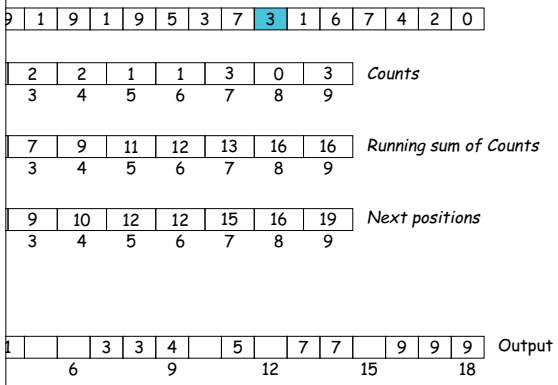
istribution Counting Example (II)



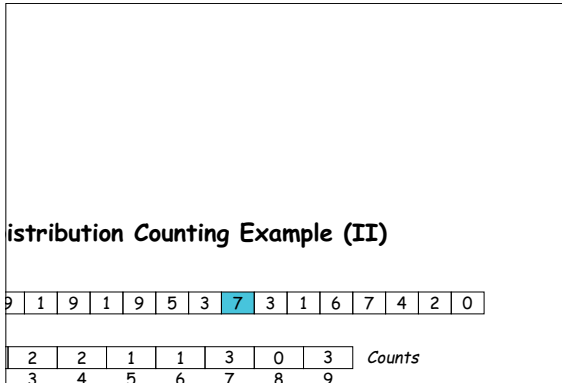
istribution Counting Example (II)



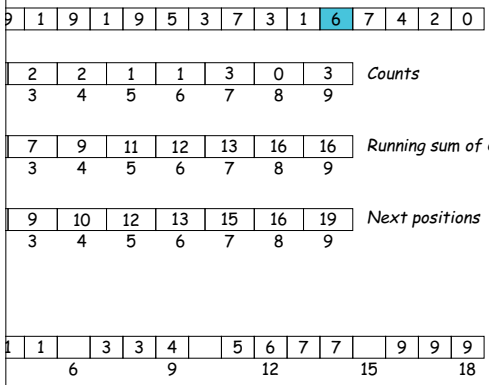
istribution Counting Example (II)



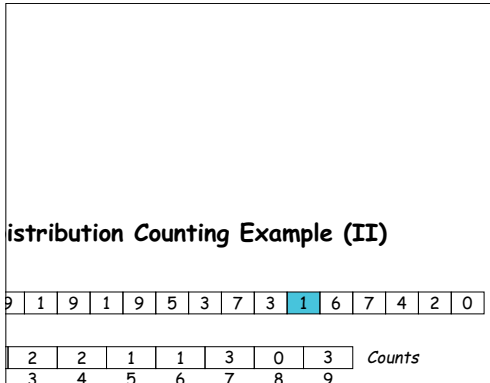
istribution Counting Example (II)



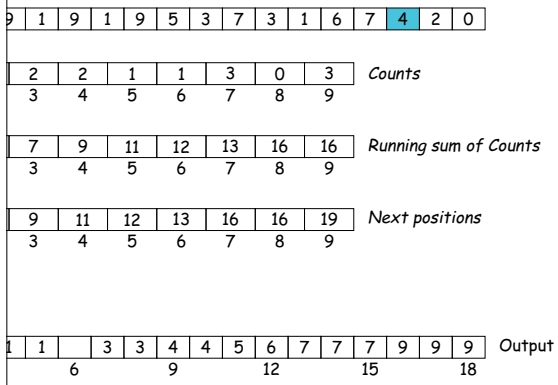
istribution Counting Example (II)



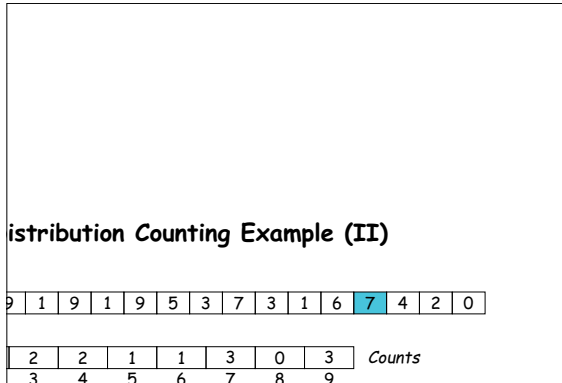
istribution Counting Example (II)



istribution Counting Example (II)



istribution Counting Example (II)



istribution Counting Example (II)

9	1	9	1	9	5	3	7	3	1	6	7	4	2	0

2	2	1	1	3	0	3	*Counts*
3	4	5	6	7	8	9	

7	9	11	12	13	16	16	*Running sum of Counts*
3	4	5	6	7	8	9	

9	11	12	13	16	16	19	*Next positions*
3	4	5	6	7	8	9	

1	1	2	3	3	4	4	5	6	7	7	7	9	9	9	Output
		6			9			12			15			18	

istribution Counting Example (II)

9	1	9	1	9	5	3	7	3	1	6	7	4	2	0

2	2	1	1	3	0	3	*Counts*
3	4	5	6	7	8	9	

7	9	11	12	13	16	16	*Running sum of Counts*
3	4	5	6	7	8	9	

9	11	12	13	16	16	19	*Next positions*
3	4	5	6	7	8	9	

1	1	2	3	3	4	4	5	6	7	7	7	9	9	9	Output
		6			9			12			15			18	

MSD Radix Sort

omplicated: must keep lists from each step separate

processing 1-element lists

A	posn
cat, cad, con, bat, can, be, let, bet	0
be, bet / cat, cad, con, can / let / set	1
$\star$ be, bet / cat, cad, con, can / let / set	2
be / bet / $\star$ cat, cad, con, can / let / set	1
be / bet / $\star$ cat, cad, can / con / let / set	2
be / bet / cad / can / cat / con / let / set	

Radix Sort

ys one character at a time.

ribution counting for each digit.

her right to left (LSD radix sort) or left to right (MSD

rt is venerable: used for punched cards.

Initial: set, cat, cad, con, bat, can, be, let, bet

bet
let
bat
can cat
e cad con set
' 'd' 'n' 't'
n, can, set, cat, bat, let, bet

Pass 2 (by char #1)

bat bet
cat let
can set
cad be con
'a' 'e' 'o'
cad, can, cat, bat, be, set, let, bet, con

Pass 3 (by char #0)

con
bet cat
be can
bat cad let set
'b' 'c' 'l' 's'
bat, be, bet, cad, can, cat, con, let, set

And Don't Forget Search Trees

h tree is in sorted order, when read in inorder.

e to really use for sorting [next topic].

e, same performance as heapsort: N insertions in time
us $\Theta(N)$ to traverse, gives

$$\Theta(N + N \lg N) = \Theta(N \lg N)$$

Performance of Radix Sort

akes $\Theta(B)$ time where B is *total size of the key data*.

ed other sorts as function of #records.

are?

ifferent records, must have keys at least $\Theta(\lg N)$ long

, comparison actually takes time $\Theta(K)$ where K is size
st case [why?]

mparisons really means $N(\lg N)^2$ operations.

sort would take $B = N \lg N$ time with minimal-length

r hand, must work to get good constant factors with

Summary

rt: $\Theta(Nk)$ comparisons and moves, where k is maximum
is displaced from final position.

small datasets or almost ordered data sets.

$(N \lg N)$ with good constant factor if data is not patho-
st case $O(N^2)$.

$\Theta(N \lg N)$ guaranteed. Good for external sorting.

eesort with guaranteed balance: $\Theta(N \lg N)$ guaranteed.

istribution sort: $\Theta(B)$ (number of bytes). Also good for
ting.