CS61B Lectures #28

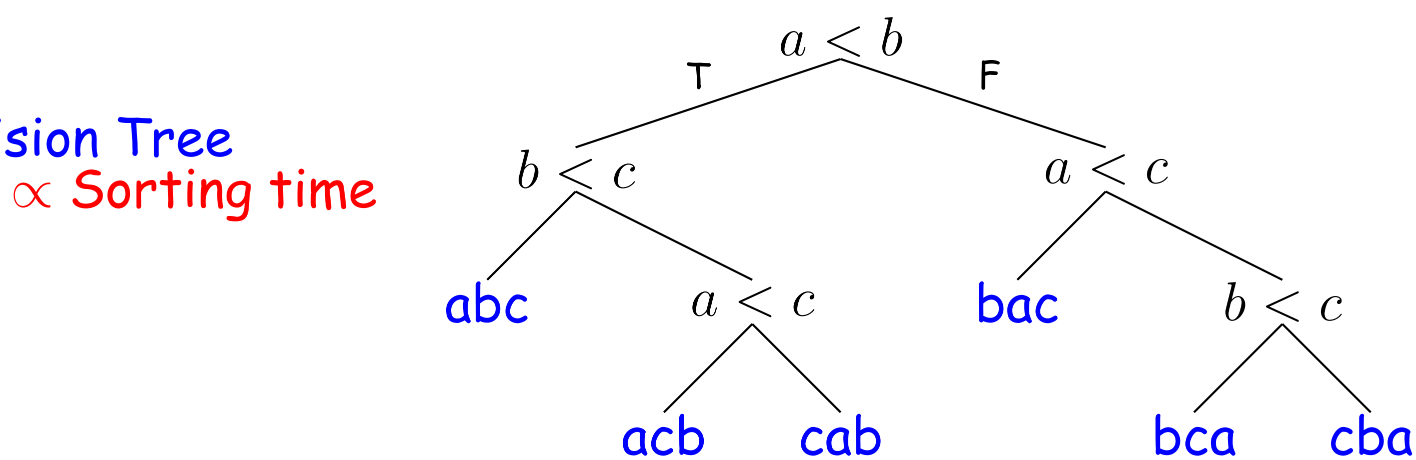
Today:

- Lower bounds on sorting by comparison
- Distribution counting, radix sorts

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

Better than N lg N?

- Can prove that *if all you can do to keys is compare them,* then must take $\Omega(N \lg N)$.
- Basic idea: there are $N!$ possible ways the input data c scrambled.
- Therefore, your program must be prepared to do $N!$ differe binations of data-moving operations.
- Therefore, there must be $N!$ possible combinations of outc all the **if**-tests in your program, since those determine wh gets moved where (we're assuming that comparisons are 2-w



Necessary Choices

- Since each **if**-test goes two ways, number of possible differe comes for k **if**-tests is 2^k.
- Thus, need enough tests so that $2^k \geq N!$, which means $k \in \Omega$
- Using Stirling's approximation,

$$
\begin{aligned}
N! &\in \sqrt{2\pi N}\left(\frac{N}{e}\right)^N\left(1+\Theta\left(\frac{1}{N}\right)\right), \\
\lg(N!) &\in 1/2(\lg 2\pi + \lg N) + N\lg N - N\lg e + \lg\left(1+\Theta\left(\right.\right. \\
&= \Theta(N\lg N)
\end{aligned}
$$

- This tells us that k, the worst-case number of tests needed N items by comparison sorting, is in $\Omega(N\lg N)$: there must b where we need (some multiple of) $N\lg N$ comparisons to things.

Beyond Comparison: Distribution

- But suppose can do more than compare keys?
- For example, how can we sort a set of N integer keys whos range from 0 to kN, for some small constant k?
- One technique: put the integers into N buckets, with an in going to bucket $\lfloor p/k \rfloor$.
- At most k keys per bucket, so catenate and use insertion sor will now be fast.
- E.g., $k = 2, N = 10:$

```
Start:
  14  3   10  13  4   2   19  17  0   9
In buckets:
  | 0 | 3  2 |  4 |   |  9  | 10 | 13 | 14 |  17
```

- Now insertion sort is fast. Putting in buckets takes time $\Theta($ insertion sort takes $\Theta(kN)$. When k is fixed (constant), sorting in time $\Theta(N)$.

Distribution Counting

- Another technique: *count* the number of items < 1, < 2, etc
- If $M_p =$ #items with value $< p$, then in sorted order, the j
  with value p must be item #$M_p + j$.
- Gives another *linear-time* algorithm.

Distribution Counting Example

- Suppose all items are between 0 and 9 as in this example:

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

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

0	3	6	7	9	11	12	13	16	16	*Running*
< 0	< 1	< 2	< 3	< 4	< 5	< 6	< 7	< 8	< 9	

0	0	0	1	1	1	2	3	3	4	4	5	6	7	7	7	9
0			3			6			9		11	12	13			16

- "Counts" line gives # occurrences of each key.
- "Running sum" gives cumulative count of keys $<$ each value. .
- . . . which tells us where to put each key:
- The first instance of key k goes into slot m, where m is the
  of key instances that are $< k$.

Distribution Counting Example (II)

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

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

0	3	6	7	9	11	12	13	16	16	*Running su*
0	1	2	3	4	5	6	7	8	9	

0	3	6	7	9	11	12	13	16	16	*Next posi*
0	1	2	3	4	5	6	7	8	9	

0			3			6			9			12			15		

Distribution Counting Example (II)

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

	0	1	2	3	4	5	6	7	8	9
Counts	3	3	1	2	2	1	1	3	0	3
Running su	0	3	6	7	9	11	12	13	16	16
Next posi	0	3	6	7	9	11	12	14	16	16

0			3			6			9			12			15		
													7				

Distribution Counting Example (II)

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

Counts

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

Running su

0	3	6	7	9	11	12	13	16	16
0	1	2	3	4	5	6	7	8	9

Next posi

1	3	6	7	9	11	12	14	16	16
0	1	2	3	4	5	6	7	8	9

0													7				
0			3			6			9			12			15		

Distribution Counting Example (II)

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

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

0	3	6	7	9	11	12	13	16	16	*Running su*
0	1	2	3	4	5	6	7	8	9	

1	3	6	7	10	11	12	14	16	16	*Next posi*
0	1	2	3	4	5	6	7	8	9	

0									4				7				
0			3			6			9			12			15		

Distribution Counting Example (II)

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

Counts

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

Running su

0	3	6	7	9	11	12	13	16	16
0	1	2	3	4	5	6	7	8	9

Next posi

2	3	6	7	10	11	12	14	16	16
0	1	2	3	4	5	6	7	8	9

0	0								4				7				
0			3			6			9			12			15		

Distribution Counting Example (II)

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

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

0	3	6	7	9	11	12	13	16	16	*Running su*
0	1	2	3	4	5	6	7	8	9	

2	3	6	7	10	11	12	14	16	17	*Next posi*
0	1	2	3	4	5	6	7	8	9	

0	0								4				7			9	
0			3			6			9			12			15		

Distribution Counting Example (II)

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

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

0	1	2	3	4	5	6	7	8	9	
0	3	6	7	9	11	12	13	16	16	*Running su*

0	1	2	3	4	5	6	7	8	9	
2	4	6	7	10	11	12	14	16	17	*Next posi*

0			3			6			9			12			15		
0	0		1						4				7			9	

Distribution Counting Example (II)

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

	0	1	2	3	4	5	6	7	8	9
Counts	3	3	1	2	2	1	1	3	0	3
Running su	0	3	6	7	9	11	12	13	16	16
Next posi	2	4	6	7	10	11	12	14	16	18

0			3			6			9			12			15		
0	0		1						4				7			9	9

Distribution Counting Example (II)

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

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

0	3	6	7	9	11	12	13	16	16	*Running su*
0	1	2	3	4	5	6	7	8	9	

2	5	6	7	10	11	12	14	16	18	*Next posi*
0	1	2	3	4	5	6	7	8	9	

0	0		1	1					4				7			9	9
0			3			6			9			12			15		

Distribution Counting Example (II)

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

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

0	3	6	7	9	11	12	13	16	16	*Running su*
0	1	2	3	4	5	6	7	8	9	

2	5	6	7	10	11	12	14	16	19	*Next posi*
0	1	2	3	4	5	6	7	8	9	

0	0		1	1					4				7			9	9
0			3			6			9			12			15		

Distribution Counting Example (II)

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

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

0	3	6	7	9	11	12	13	16	16	*Running su*
0	1	2	3	4	5	6	7	8	9	

2	5	6	7	10	12	12	14	16	19	*Next posi*
0	1	2	3	4	5	6	7	8	9	

0	0		1	1					4		5		7			9	9
0			3			6			9			12			15		

Distribution Counting Example (II)

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

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

0	3	6	7	9	11	12	13	16	16	*Running su*
0	1	2	3	4	5	6	7	8	9	

2	5	6	8	10	12	12	14	16	19	*Next posi*
0	1	2	3	4	5	6	7	8	9	

0	0		1	1			3		4		5		7			9	9
0			3			6			9			12			15		

Distribution Counting Example (II)

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

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

0	3	6	7	9	11	12	13	16	16	*Running su*
0	1	2	3	4	5	6	7	8	9	

2	5	6	8	10	12	12	15	16	19	*Next posi*
0	1	2	3	4	5	6	7	8	9	

0	0		1	1			3		4		5		7	7		9	9
0			3			6			9			12			15		

Distribution Counting Example (II)

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

Counts

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

Running su

0	3	6	7	9	11	12	13	16	16
0	1	2	3	4	5	6	7	8	9

Next posi

2	5	6	9	10	12	12	15	16	19
0	1	2	3	4	5	6	7	8	9

0	0		1	1			3	3	4		5		7	7		9	9
0			3			6			9			12			15		

Distribution Counting Example (II)

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

	0	1	2	3	4	5	6	7	8	9
Counts	3	3	1	2	2	1	1	3	0	3
Running su	0	3	6	7	9	11	12	13	16	16
Next posi	2	6	6	9	10	12	12	15	16	19

0			3			6			9			12			15		
0	0		1	1	1		3	3	4		5		7	7		9	9

Distribution Counting Example (II)

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

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

0	3	6	7	9	11	12	13	16	16	*Running su*
0	1	2	3	4	5	6	7	8	9	

2	6	6	9	10	12	13	15	16	19	*Next posi*
0	1	2	3	4	5	6	7	8	9	

0	0		1	1	1		3	3	4		5	6	7	7		9	9
0			3			6			9			12			15		

Distribution Counting Example (II)

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

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

Counts

0	3	6	7	9	11	12	13	16	16
0	1	2	3	4	5	6	7	8	9

Running su

2	6	6	9	10	12	13	16	16	19
0	1	2	3	4	5	6	7	8	9

Next posi

0	0		1	1	1		3	3	4		5	6	7	7	7	9	9
0			3			6			9			12			15		

Distribution Counting Example (II)

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

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

0	3	6	7	9	11	12	13	16	16	*Running su*
0	1	2	3	4	5	6	7	8	9	

2	6	6	9	11	12	13	16	16	19	*Next posi*
0	1	2	3	4	5	6	7	8	9	

0	0		1	1	1		3	3	4	4	5	6	7	7	7	9	9
0			3			6			9			12			15		

Distribution Counting Example (II)

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

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

0	3	6	7	9	11	12	13	16	16	*Running su*
0	1	2	3	4	5	6	7	8	9	

2	6	7	9	11	12	13	16	16	19	*Next posi*
0	1	2	3	4	5	6	7	8	9	

0	0		1	1	1	2	3	3	4	4	5	6	7	7	7	9	9
0			3			6			9			12			15		

Distribution Counting Example (II)

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

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

0	3	6	7	9	11	12	13	16	16	*Running su*
0	1	2	3	4	5	6	7	8	9	

3	6	7	9	11	12	13	16	16	19	*Next posi*
0	1	2	3	4	5	6	7	8	9	

0	0	0	1	1	1	2	3	3	4	4	5	6	7	7	7	9	9
0			3			6			9			12			15		

Radix Sort

Idea: Sort keys *one character at a time.*

- Can use distribution counting for each digit.
- Can work either right to left (LSD radix sort) or left to righ
  radix sort)
- LSD radix sort is venerable: used for punched cards.

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

Pass 1 (by char #2)

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

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

Pass 2 (by char #1)

'a'	'e'
bat	bet
cat	let
can	set
cad	be

cad, can, cat, bat, be

Pass 3 (by char #0)

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

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

MSD Radix Sort

- A bit more complicated: must keep lists from each step sep
- But, can stop processing 1-element lists

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

Performance of Radix Sort

- Radix sort takes $\Theta(B)$ time where B is *total size of the key*
- Have measured other sorts as function of #records.
- How to compare?
- To have N different records, must have keys at least $\Theta(\lg$
  [why?]
- Furthermore, comparison actually takes time $\Theta(K)$ where K
  of key in worst case [why?]
- So $N \lg N$ comparisons really means $N(\lg N)^2$ operations.
- While radix sort would take $B = N \lg N$ time with minima
  keys.
- On the other hand, must work to get good constant facto
  radix sort.

And Don't Forget Search Trees

Idea: A search tree is in sorted order, when read in inorder.

- Need *balance* to really use for sorting [next topic].
- Given balance, same performance as heapsort: N insertions $\lg N$ each, plus $\Theta(N)$ to traverse, gives

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

Summary

- Insertion sort: $\Theta(Nk)$ comparisons and moves, where k is m
amount data is displaced from final position.
  - Good for small datasets or almost ordered data sets.
- Quicksort: $\Theta(N \lg N)$ with good constant factor if data is no
logical. Worst case $O(N^2)$.
- Merge sort: $\Theta(N \lg N)$ guaranteed. Good for external sortir
- Heapsort, treesort with guaranteed balance: $\Theta(N \lg N)$ guar
- Radix sort, distribution sort: $\Theta(B)$ (number of bytes). Also g
external sorting.