# CS61B Lectures #29

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 sorting must take $\Omega(N \lg N)$.

- Basic idea: there are $N!$ possible ways the input data could be scrambled.

- Therefore, your program must be prepared to do $N!$ different combinations of data-moving operations.

- Therefore, there must be $N!$ possible combinations of outcomes of all the **if**-tests in your program, since those determine what move gets moved where (we're assuming that comparisons are 2-way).

Decision Tree
Height $\propto$ Sorting time
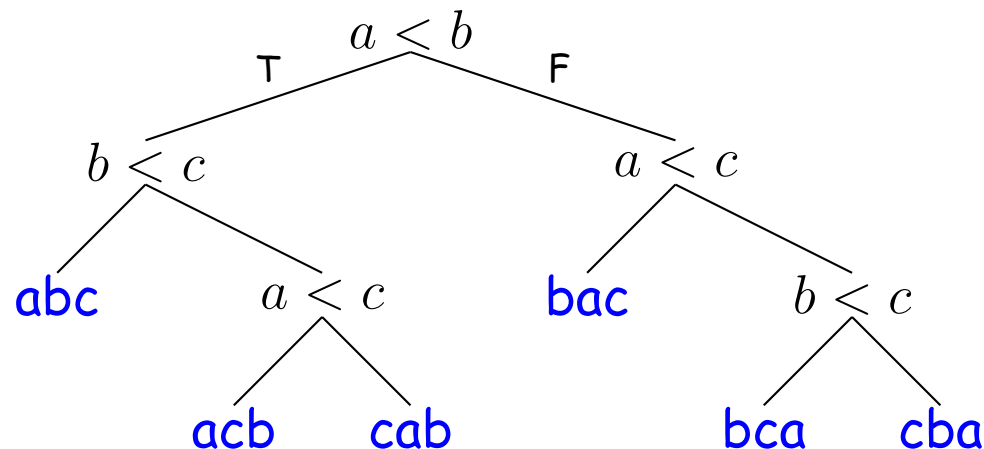
Necessary Choices

- Since each **if**-test goes two ways, number of possible different outcomes for k **if**-tests is 2^k.

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

- Using Stirling's approximation,

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

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

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

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

Beyond Comparison: Distribution

- But suppose can do more than compare keys?

- For example, how can we sort a set of N integer keys whose values range from 0 to kN, for some small constant k?

- One technique: put the integers into N buckets, with an integer p going to bucket $\lfloor p/k \rfloor$.

- At most k keys per bucket, so catenate and use insertion sort, which 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 |   19   |
```

- Now insertion sort is fast. Putting in buckets takes time $\Theta(N)$, and insertion sort takes $\Theta(kN)$. When k is fixed (constant), we have 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 jth item 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	2	0

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 sum*
< 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	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 number 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	0

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 sum of Counts
0	1	2	3	4	5	6	7	8	9	

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

Output

0   3   6   9   12   15   18

Distribution Counting Example (II)

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

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 sum of Counts
0	1	2	3	4	5	6	7	8	9	

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

													7						Output
0			3			6			9			12			15			18	

Distribution Counting Example (II)

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

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 sum of Counts
0	1	2	3	4	5	6	7	8	9	

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

0												7							Output
0			3			6			9			12			15			18	

Distribution Counting Example (II)

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

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 sum of Counts
0	1	2	3	4	5	6	7	8	9	

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

0									4			7							Output

0   3   6   9   12   15   18

Distribution Counting Example (II)

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

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 sum of Counts
0	1	2	3	4	5	6	7	8	9	

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

0	0								4			7						Output

0   3   6   9   12   15   18

Distribution Counting Example (II)

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

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 sum of Counts
0	1	2	3	4	5	6	7	8	9	

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

0	0								4			7			9			Output

0　　　　3　　　　6　　　　9　　　　12　　　　15　　　　18

Distribution Counting Example (II)

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

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 sum of Counts
0	1	2	3	4	5	6	7	8	9

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

0	0		1					4			7			9				Output
0			3			6			9			12			15			18

Distribution Counting Example (II)

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

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 sum of Counts
0	1	2	3	4	5	6	7	8	9	

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

0	0		1					4			7			9	9		Output
0			3			6			9			12			15		18

Distribution Counting Example (II)

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

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 sum of Counts
0	1	2	3	4	5	6	7	8	9	

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

0	0		1	1				4				7			9	9		Output
0			3			6			9			12			15			18

Distribution Counting Example (II)

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

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 sum of Counts
0	1	2	3	4	5	6	7	8	9	

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

0	0		1	1				4			7			9	9	9	Output
0			3			6			9			12			15		18	

Distribution Counting Example (II)

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

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 sum of Counts
0	1	2	3	4	5	6	7	8	9	

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

0	0		1	1				4		5		7			9	9	9	Output
0			3			6			9			12			15			18

Distribution Counting Example (II)

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

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 sum of Counts
0	1	2	3	4	5	6	7	8	9	

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

0	0		1	1			3		4		5		7			9	9	9	Output
0			3			6			9			12			15			18	

Distribution Counting Example (II)

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

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 sum of Counts
0	1	2	3	4	5	6	7	8	9	

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

0	0		1	1			3		4		5			7	7			9	9	9	Output
0			3			6			9			12			15			18				

Distribution Counting Example (II)

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

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 sum of Counts
0	1	2	3	4	5	6	7	8	9	

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

0	0		1	1			3	3	4		5		7	7		9	9	9	Output
0			3			6			9			12			15			18	

Distribution Counting Example (II)

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

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 sum of Counts
0	1	2	3	4	5	6	7	8	9	

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

0	0		1	1	1		3	3	4		5		7	7		9	9	9	Output

0   3   6   9   12   15   18

Distribution Counting Example (II)

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

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 sum of Counts
0	1	2	3	4	5	6	7	8	9	

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

0	0		1	1	1		3	3	4		5	6	7	7		9	9	9	Output
0			3			6			9			12			15			18	

Distribution Counting Example (II)

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

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 sum of Counts
0	1	2	3	4	5	6	7	8	9	

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

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

0			3			6			9			12			15			18

Distribution Counting Example (II)

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

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 sum of Counts
0	1	2	3	4	5	6	7	8	9	

2	6	6	9	11	12	13	16	16	19	Next positions
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	9	Output

0			3			6			9			12			15			18

Distribution Counting Example (II)

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

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 sum of Counts
0	1	2	3	4	5	6	7	8	9	

2	6	7	9	11	12	13	16	16	19	Next positions
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	9	Output
0			3			6			9			12			15			18	

Distribution Counting Example (II)

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

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 sum of Counts
0	1	2	3	4	5	6	7	8	9	

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

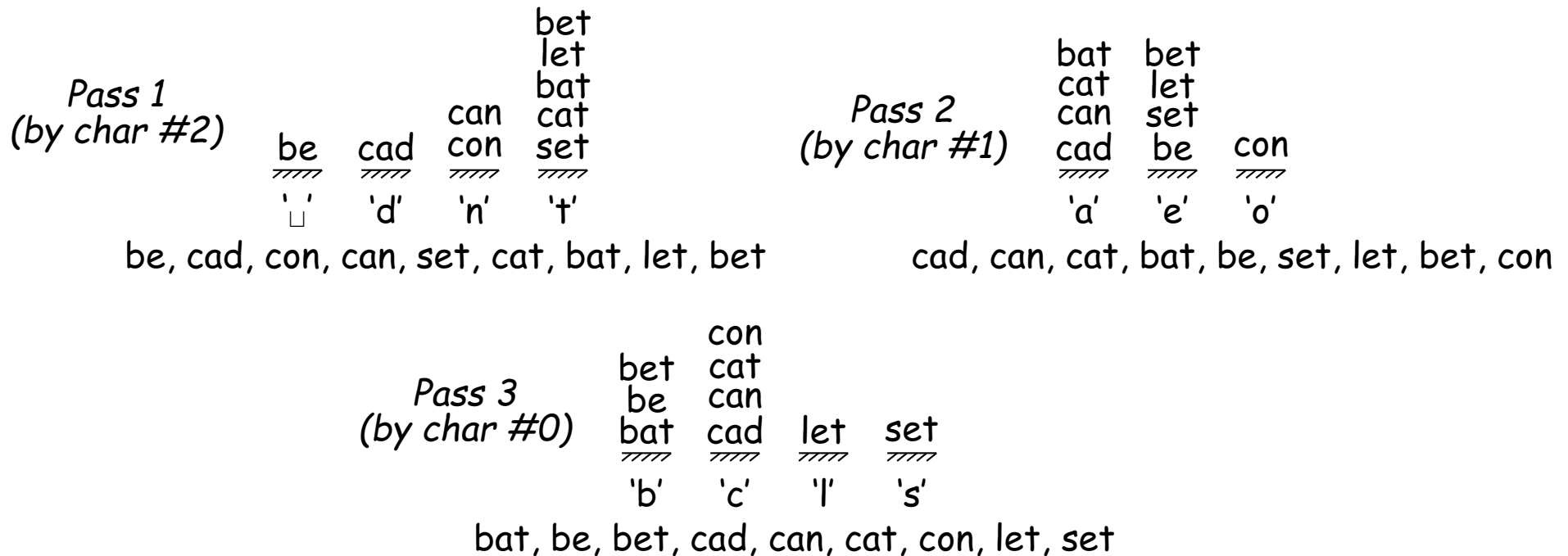| 0	0	0	1	1	1	2	3	3	4	4	5	6	7	7	7	9	9	9	Output
0			3			6			9			12			15			18	

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 right (MSD 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)

```
                            bet
                            let
                            bat
                 can        cat
          be  cad  con  set
          ‾‾‾  ‾‾‾  ‾‾‾  ‾‾‾
          '␣'  'd'  'n'  't'
```
be, cad, con, 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

MSD Radix Sort

- A bit more complicated: must keep lists from each step separate

- But, can stop processing 1-element lists

A	posn
⋆ set, cat, cad, con, bat, can, be, let, bet	0
⋆ bat, be, bet / cat, cad, con, can / let / set	1
bat / ⋆ be, bet / cat, cad, con, can / let / set	2
bat / be / bet / ⋆ cat, cad, con, can / let / set	1
bat / be / bet / ⋆ 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 data*.

- Have measured other sorts as function of #records.

- How to compare?

- To have N different records, must have keys at least $\Theta(\lg N)$ long [why?]

- Furthermore, comparison actually takes time $\Theta(K)$ where K is size 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 minimal-length keys.

- On the other hand, must work to get good constant factors with 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 in time $\lg N$ 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 maximum 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 not pathological. Worst case $O(N^2)$.

- Merge sort: $\Theta(N \lg N)$ guaranteed. Good for external sorting.

- Heapsort, treesort with guaranteed balance: $\Theta(N \lg N)$ guaranteed.

- Radix sort, distribution sort: $\Theta(B)$ (number of bytes). Also good for external sorting.