6.064
lecture 5

How fast can we sort?

depends on model of what you can do with the elements.

e.g. quicksort, heapsort, mergesort, insertion sort

$\Theta(n \lg n)$ avg or $\Theta(n^2)$ $\quad \Theta(n \lg n)$ $\quad \Theta(n \lg n)$ $\quad \Theta(n^2)$

Can we do better than $\Theta(n \lg n)$?

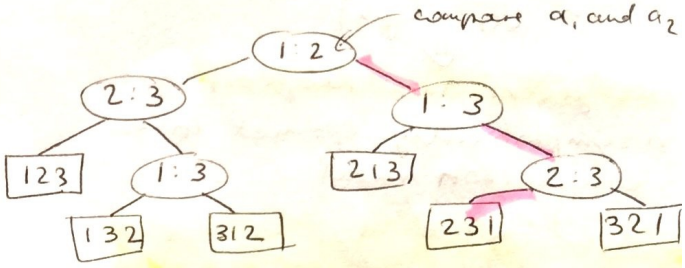
Comparison sorting (model):

only use comparisons to determine rel. order of elements

Decision-tree model:

Example: sort $\langle a_1, a_2, a_3 \rangle$

e.g. 9 4 6



in general $\langle a_1, \ldots, a_n \rangle$

- each internal node has label $i:j$ $\quad i,j \in \{1, 2, \ldots, n\}$
  - means: compare a_i vs. a_j
  - left subtree gives subsequent comparisons if $a_i \leq a_j$
  - right subtree gives subsequent comparisons if $a_i > a_j$
- each leaf gives a permutation $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ such that $a_{\pi(1)} \leq a_{\pi(2)} \leq \cdots \leq a_{\pi(n)}$ (↖ permutation function)

any comparison sort can be reduced to a decision tree on input n

Decision trees model comparison sorts:

- one tree for each n ← one tree assumption breaks in randomised
- view algorithm as splitting whenever it makes a comparison
- trace both comparisons along all possible instruction traces

size of tree: leaves have to represent [redacted] each permutation of [redacted] list of size n ⇒ n! permutations (exponential above a)

⇒ graphical representation as decision tree is not [redacted] practical as a representation of a comp. sort algorithm

pseudocode is constant length representation

- running time (# comparisons) = length of path
- worst-case run time = height of tree

Lower bound on decision-tree sorting:
Any decision tree sorting n elements has height $\Omega(n \lg n)$

Proof:
- # leaves must be $\geq n!$
- height h $\Rightarrow$ # leaves $\leq 2^h$ $\Rightarrow n! \leq 2^h$

$\Rightarrow h \geq \lg n!$ (lg is monotonically increasing) $\Rightarrow$ ineq. stays the same

Stirling $\rightarrow \geq \lg \left(\frac{n}{e}\right)^n$

$= n \lg \left(\frac{n}{e}\right)$

$= n(\lg(n) - \lg(e))$

$= \Omega(n \lg n)$ (actually Θ but we care only about Ω)

$\Omega(n \lg n)$ for height of any decision tree

$\Downarrow$

all comparison sorts run in $\Omega(n \lg n)$

$\Downarrow$

mergesort and heapsort are asymptotically optimal in the comparison model

in randomised algs we get a probability distr. over trees, due to coin flips
↳ proof applies to any tree $\Rightarrow$ lower bound also for randomised comparison sorts

randomised quicksort is asymptotically optimal in expectation

Sorting in linear time

cannot sort better than $\Theta(n)$ → need to look at data

Counting sort:

Input: $A[1 \ldots n]$
each $A[i] \in \{1, 2, \ldots, k\}$

Output: $B[1 \ldots n]$ sorting of A

Aux storage: $C[1 \ldots k]$

Counting sort:

```
for i ← 1 to k
  do C[i] ← 0
for j ← 1 to n
  do C[A[j]] ← C[A[j]] + 1
     // C[i] = |{key = i}|
for i ← 2 to k
  do C[i] ← C[i] + C[i-1]
     // C[i] = |{key ≤ i}|
for j ← n downto 1
  do B[C[A[j]]] ← A[j]
     C[A[j]] ← C[A[j]] - 1
```

distribution →

Ex: $n = 5$

1st for loop: $A =$ | 4 | 1 | 3 | 4 | 3 | $\quad C =$ | 0 | 0 | 0 | 0 |

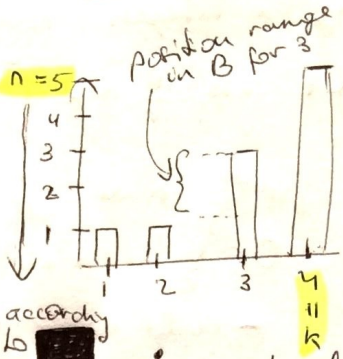
2nd for loop: $C =$ | 0 | 0 | 0 | 0 |

3rd for loop: $C =$ | 1 | 0 | 2 | 2 |

$C =$ | 1 | 0 | 2 | 2 |

$C =$ | 1 | 1 | 3 | 5 |

4th for loop



according to [redacted] order in A!

- get value from A
- get position of value in B according to cum. distr. in C
- update distr. in C

$B =$ | 1 | 3 | 3 | 4 | 4 |

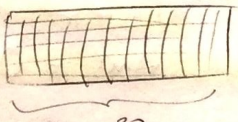
Time $O(k+n)$

if $k = O(n)$ then $O(n)$

but as soon as $k = O(n \log n)$ → can switch to comparison sorting

Stable sort: preserves the relative order of equal elements

Radix sort (Hollerith - 1890)



digit-by-digit

first: sort by most sig. digit first

right: sort by least sig. digit first

Ex (3-digit #s)

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

→ sorted:

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

same order (stable) →

sorted:

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

→

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

Correctness: induct on digit position t

- assume sorted on low order $t-1$ digits (IH)
- sort on digit t
  1. if two elts have same t^{th} digit → same order by stability (by IH are sorted)
  2. if different t^{th} digit, => sorted order

Analysis:

- use counting sort / digit $O(k+n)$ ← each round counting sort
- say n integers, each b bits (range $= 0 \ldots 2^b - 1$)
- split into $\frac{b}{r}$ "digits" each r bits, (base 2^r)

$\frac{b}{r}$ ← # of rounds; k ← counting sort run

My concept resembles the idea of Bloom filters

Time: $O\left(\frac{b}{r} \cdot (n+k)\right) = O\left(\frac{b}{r}(n+2^r)\right)$

- could differentiate with respect to r, set to 0
- $\frac{b}{r} \cdot n$ wants r big, $\frac{b}{r} 2^r$ wants r small

↳ choose r max, subject to $n \geq 2^r$

$r = \lg n$ (get some via differentiation)

$\Rightarrow O\left(\frac{bn}{\lg n}\right)$ ← [redacted] size of problem b

if numbers in range $0 \ldots 2^b - 1$,

as a polynomial in $0 \ldots n^d - 1$

then time $= O(dn)$

$O(\lg n)$ bits long #'s

advantage over $\Theta(n \lg n)$ comparison sorts

advantage over $\Theta(n \lg n)$

counting sort	radix sort
$k = O(n \lg n)$ [redacted]	$d = O(\lg n$ [redacted]$)$
	$k = O(n \lg n)$

if we know that numbers a $O(\lg n)$ bits long consider radix