6.851	Lecture 14	Apr. 10, 2012

TODAY: Integer sorting & priority queues
- reduction between them
- survey of sorts
- signature sort: $O(n)$ for $w = \Omega(\lg^{2+\varepsilon} n)$
- packed sort: $O(n)$ for $w = \Omega(b \lg n \lg\lg n)$

  bits in input integers
- bitonic sort for merging sorted words

Priority queues:
- $O(P(n,w))$ priority queue $\Rightarrow O(n\,P(n,w))$ sort [trivial]
- $O(n\,S(n,w))$ sorting algorithm $\Rightarrow$ $O(S(n,w))$ worst-case priority queue [Thorup – J.ACM 2007]

  insert, delete, find-min
- $O(P(n,w))$ priority queue $\Rightarrow$ $O(P(n,w)+\alpha(n))$ meldable priority queue [Mendelson, Tarjan, Thorup, Zwick – TALG 2006]

  merge two queues in $O(1)$ am.

$\boxed{\text{OPEN}}$: $O(n\,S(n,w))$ sorting alg. $\Rightarrow$ $O(S(n,w))$ delete-min & $O(1)$ decrease-key & insert? [Demaine & Pătraşcu 2005]

Integer sorting: sort n w-bit integers

- comparison sort: $O(n \lg n)$
- counting sort: $O(n+u)$
  $= O(n)$ for $w = \lg n$
- radix sort: $O(n \frac{w}{\lg n})$
  $= O(n)$ for $w = O(\lg n)$
- van Emde Boas sort: $O(n \lg w)$
  $= O(n \lg\lg n)$ for $w = \lg^{O(1)} n$
  - with more care: $O(n \lg \frac{w}{\lg n})$ [Spring'05, PS7]

TODAY * signature sort: $O(n)$ for $w = \Omega(\lg^{2+\varepsilon} n)$ $\forall \varepsilon > 0$
  $\Rightarrow O(n \lg\lg n)$ for all w
  [Andersson, Hagerup, Nilsson, Rahman — JCSS 1998]
  - note: much better than "fusion sort" $O(n \frac{\lg n}{\lg w})$
- Han [J. Alg. 2001]: $O(n \lg\lg n)$ deterministic $AC^{\emptyset}$
- Han & Thorup [FOCS 2002]: $O(n\sqrt{\lg \frac{w}{\lg n}})$ randomized
  $= O(n\sqrt{\lg\lg n})$ for $w = \lg^{O(1)} n$
  $\Rightarrow O(n\sqrt{\lg\lg n})$ for all w

$\boxed{\text{OPEN}}$: optimal sorting for $w = \omega(\lg n)$ & $o(\lg^{2+\varepsilon} n)$

Signature sort: [Andersson et al. 1998]

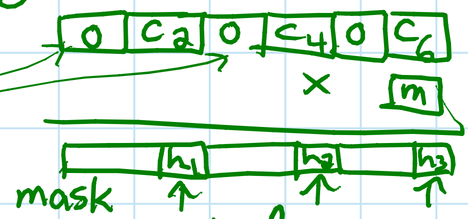
- assume $w \geq \lg^{2+\varepsilon} n \cdot \lg\lg n$ (change ε)

(1) break each integer into $\lg^{\varepsilon} n$ equal-size chunks

(2) replace each chunk by $O(\lg n)$-bit hash "signature"

$\Rightarrow n$ $O(\lg^{1+\varepsilon} n)$-bit signatures

- need to be able to hash $\lg^{\varepsilon} n$ chunks in $O(1)$
- e.g. multiplication method:
- just need adjacent blanks to prevent overflow collision



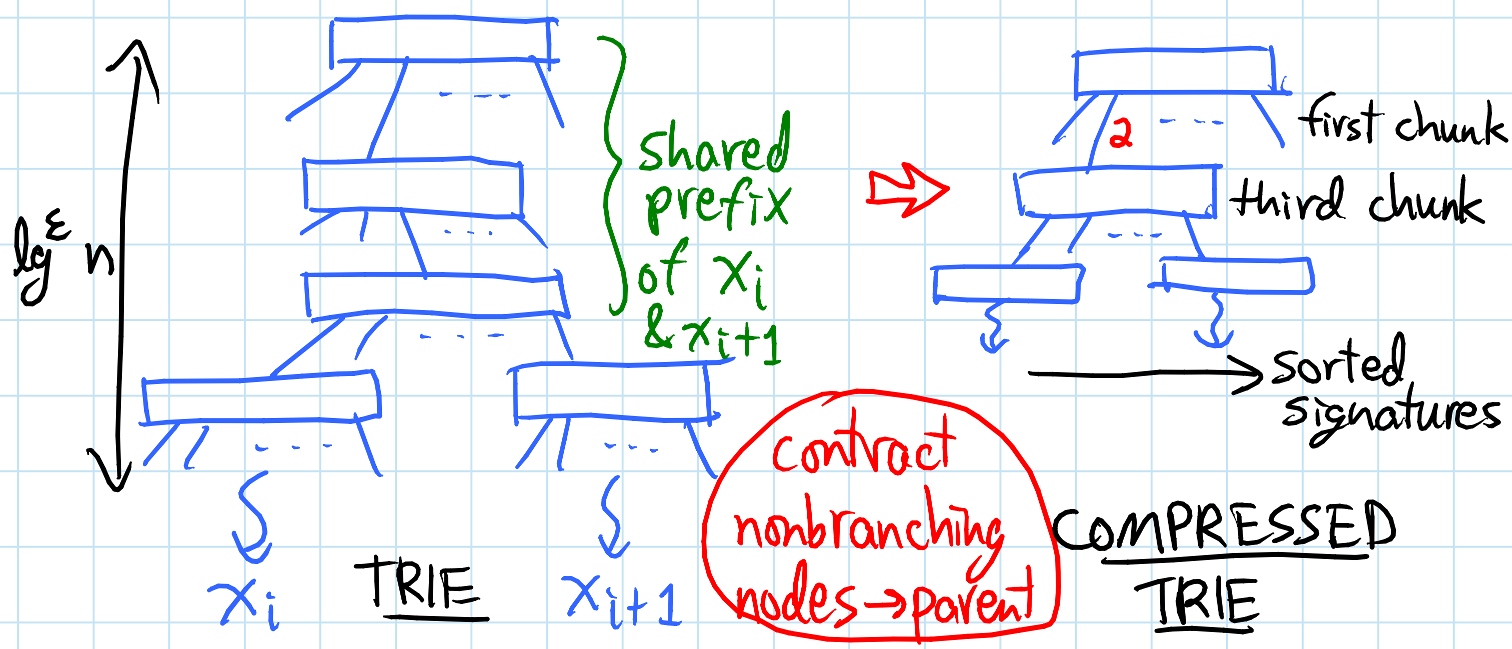
- so mask & do odds & evens separately, then OR together
- can compactify via sketch techniques [L12]

(3) packed sorting sorts them in $O(n)$ time: } TO DO

n b-bit integers with $w = \Omega(b \lg n \lg\lg n)$

- trouble: hash does not preserve order

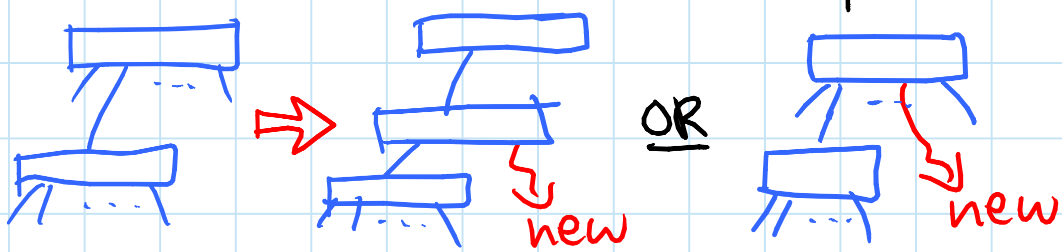
(4) build compressed trie of sorted signatures:



Building compressed trie in $O(n)$ time:

(like suffix array → tree conversion [L16])

- for $i = 1, 2, \ldots, n$: add ith signature
- compute lcp with $(i-1)$st signature: first 1 bit in XOR (like fusion trees) rounded to chunk #
- walk up to appropriate node/compressed edge
- charge distance walked to decrease in rightmost path length (potential)
- add new branch from lca/lcp — $O(1)$



$\Rightarrow O(n)$ total time

~ or notice you're just doing an in-order traversal of the tree to be computed

5) recursively sort $(\underbrace{\text{node ID}}_{O(\lg n)\text{ bits}}, \underbrace{\text{actual chunk}}_{w/\lg^\varepsilon n\text{ bits}}, \underbrace{\text{edge index}}_{O(\lg n)\text{ bits}})$ $\forall$ edge

$\Rightarrow n$ remains same, $\underbrace{b}_{\#\text{bits in an integer}}$ reduces to $b/\lg^\varepsilon n + O(\lg n)$

$\Rightarrow$ after $1/\varepsilon + 1 = O(1)$ recursions,

$$b = O\left(\lg n + \frac{w}{\lg^{1+\varepsilon} n}\right) = O\left(\frac{w}{\lg^{1+\varepsilon} n}\right) = O\left(\frac{w}{\lg n \lg\lg n}\right)$$

$\Rightarrow$ packed sort in base case

6) scan through & permute each node accordingly

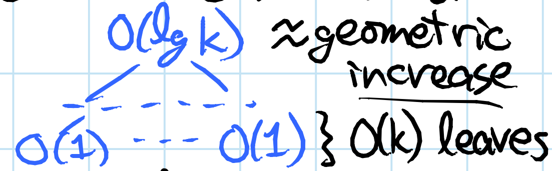
7) in-order traversal of leaves

Packed sorting: $w \geq 2(b+1) \lg n \lg\lg n$ (for convenience)

⓪ pack $\lg n \lg\lg n$ elements into each word:

① merge pair of sorted words with $k \leq \lg n \lg\lg n$ elts. into one sorted word with $2k$ elts. in $O(\lg k)$ time
- hardest step (TO DO) - bitonic sorting + bit tricks

② mergesort $k = \lg n \lg\lg n$ elts. into one word in $T(k) = 2T(k/2) + O(\lg k)$
$= O(k)$ time



③ merge two sorted lists of r sorted words into one sorted list of $2r$ sorted words in $O(r \lg k)$ time
- like standard merge but with ① as comparator
- merge first word of each list → 2 words
- output first word
- put second word at front of list containing max elt. in that word

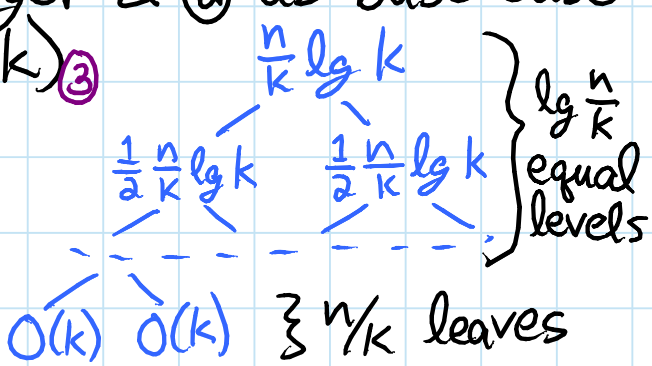
④ mergesort with ③ as merger & ② as base case

$$\Rightarrow T(n) = 2T\left(\frac{n}{2}\right) + O\left(\underbrace{\frac{n}{k}}_{r} \lg k\right) \text{ ③}$$

$$T(k) = O(k) \text{ ②}$$

$$\Rightarrow T(n) = O\left(\frac{n}{k} \lg k \lg \frac{n}{k} + \frac{n}{k} \cdot k\right)$$

$$\leq O\left(\frac{n}{k} \lg k \lg n + n\right)$$



- $k = \lg n \lg\lg n \Rightarrow \lg k = \Theta(\lg\lg n)$

$$\Rightarrow T(n) = O(n)$$

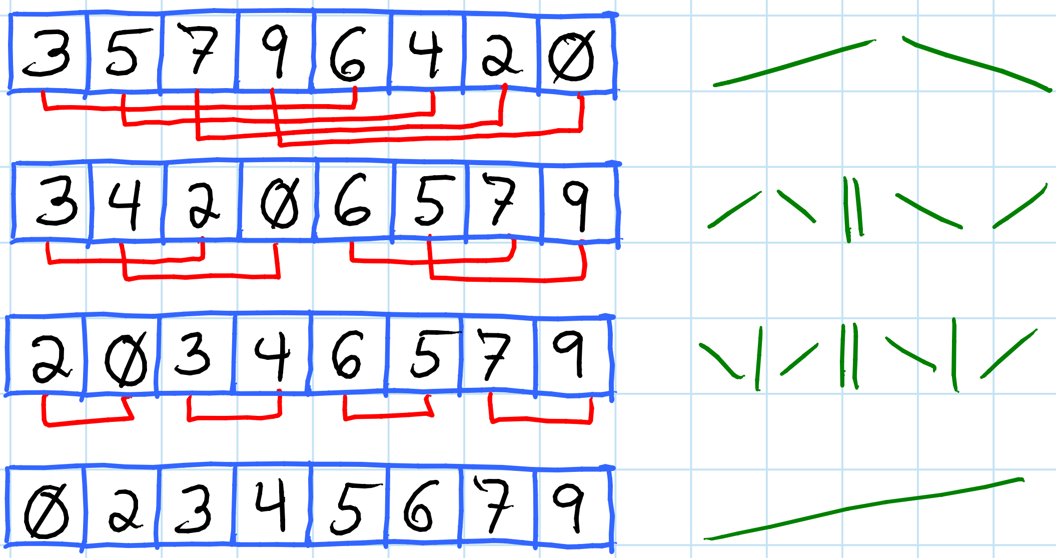
Bitonic sorting: (from parallel algorithms / sorting networks)

Bitonic sequence = cyclic shift of nondecreasing + nonincreasing sequences

- i.e.:

Algorithm: (sorting network)

- put $A[i]$ & $A[n/2+i]$ in right order for $i = 0, 1, \dots, n/2 - 1$
- split A in half (at $n/2$)
- recurse on halves in parallel



- $O(\lg n)$ rounds

Invariant after round: [CLR & CLRS 2e (not 3e)]

- both halves are bitonic
- all elts. in left half < all elts. in right

(look at which red comps. straddle peak)

Merging two sorted words of k elts. in $O(\lg k)$ time

① reverse order of second word in $O(\lg k)$ time

– idea: $\mathrm{rev}(LR) = \underbrace{\mathrm{rev}(R)\ \mathrm{rev}(L)}_{\text{recurse on halves in parallel}}$

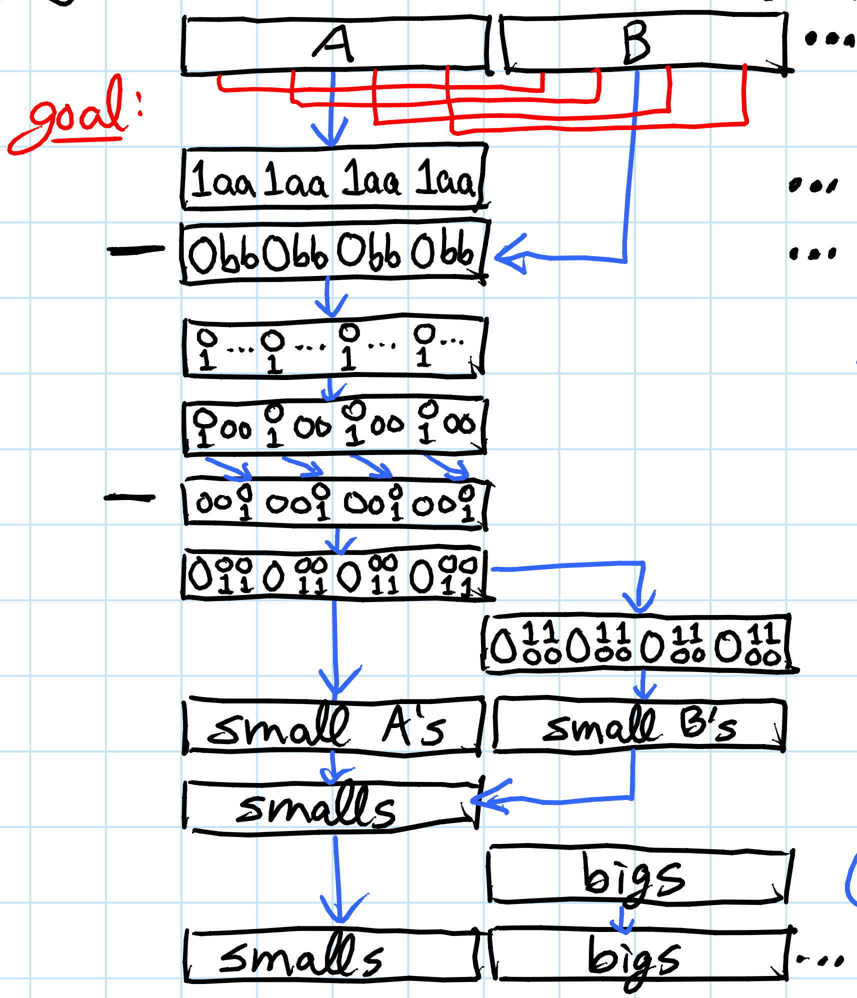
[(mask L) >> k/2] OR [(mask R) << k/2]

ditto, but shifts of k/4

etc.

② concatenate two words (shift & OR) ⇒ bitonic

③ bitonic sort, each round in $O(1)$ time:



... mask A, OR lead bits

... mask B, shift left

subtract: 0 ⇒ B smaller, 1 ⇒ A smaller

mask

shift right

subtract

shift, negate, mask

mask with A, B

shift, OR

(similar)

... OR