Lecture 17

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 2005, 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]

- Han [J. Alg. 2001]: $O(n \lg \lg n)$ deterministic, AC^0
- Han & Thorup [FOCS 2002] $O(n \sqrt{\lg \frac{w}{\lg n}})$
  $= O(n \sqrt{\lg \lg n})$ for $w = O(\lg n)$
  $\Rightarrow O(n \sqrt{\lg \lg n})$ for all w

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

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

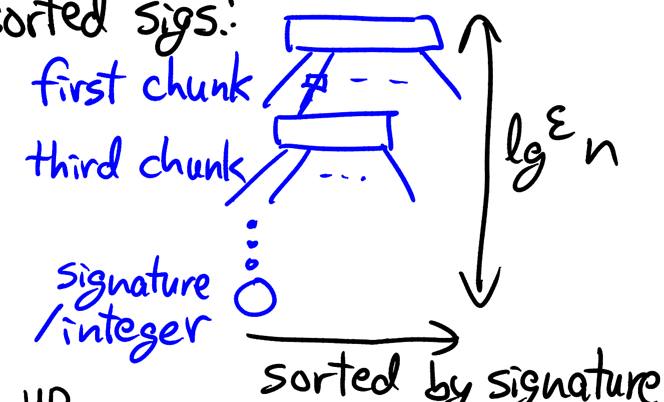
② replace each chunk by $O(\lg n)$-bit hash → signature

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

③ packed sorting sorts them in $O(n)$ time: TO BE DONE
n b-bit integers with $w = \Omega(b \lg n \lg\lg n)$

- trouble: hashes do not preserve order

④ build compressed trie of sorted sigs.:



- for $i = 1, 2, \ldots, n$:
  add ith signature
- compute lcp with $(i-1)$st sig.: first 1 bit in XOR
- charge length of walk up to decrease in rightmost path length (potential)
- add new branch from lca/lcp — $O(1)$

⇒ $O(n)$ total (like suffix array → tree construction)

- now just need to permute edges from each node

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

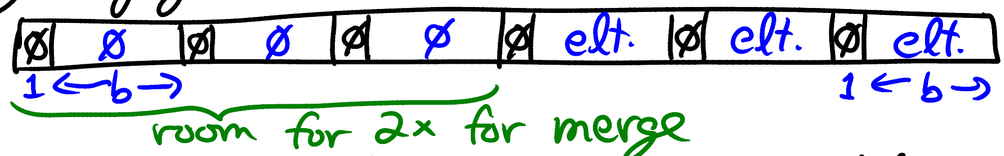
⇒ after $O(1/\varepsilon) = O(1)$ recursions, have $O(\lg n)$-bit integers
⇒ radix sort or packed sort in base case

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)

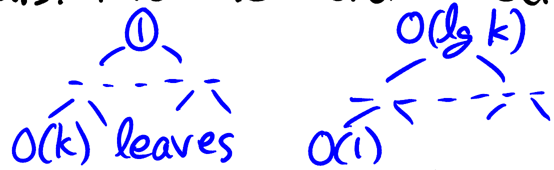
(0) pack $\lg n \lg\lg n$ elements into each word:



(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
- bitonic sort + bit tricks

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

$\Rightarrow T(k) = 2T(k/2) + O(\lg k)$
$= O(k)$



(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(word)

(4) mergesort with (3) as merger & (2) as base case

$\Rightarrow T(n) = 2T(n/2) + O(\underbrace{\tfrac{n}{k}}_{r} \lg k)_{(3)}$

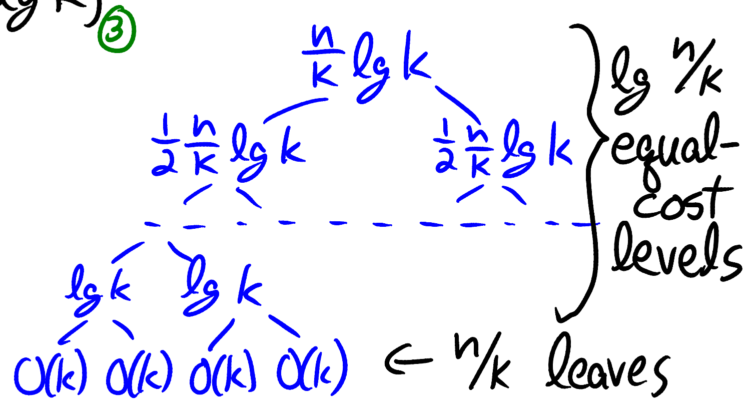
$T(k) = O(k)_{(2)}$

$\Rightarrow T(n) = O(\tfrac{n}{k} \lg k \lg \tfrac{n}{k} + \tfrac{n}{k} \cdot k)$

$\leq O(\tfrac{n}{k} \lg k \lg n + n)$

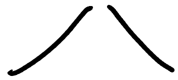
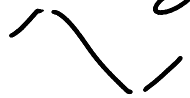
- $k = \lg n \lg\lg n$

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



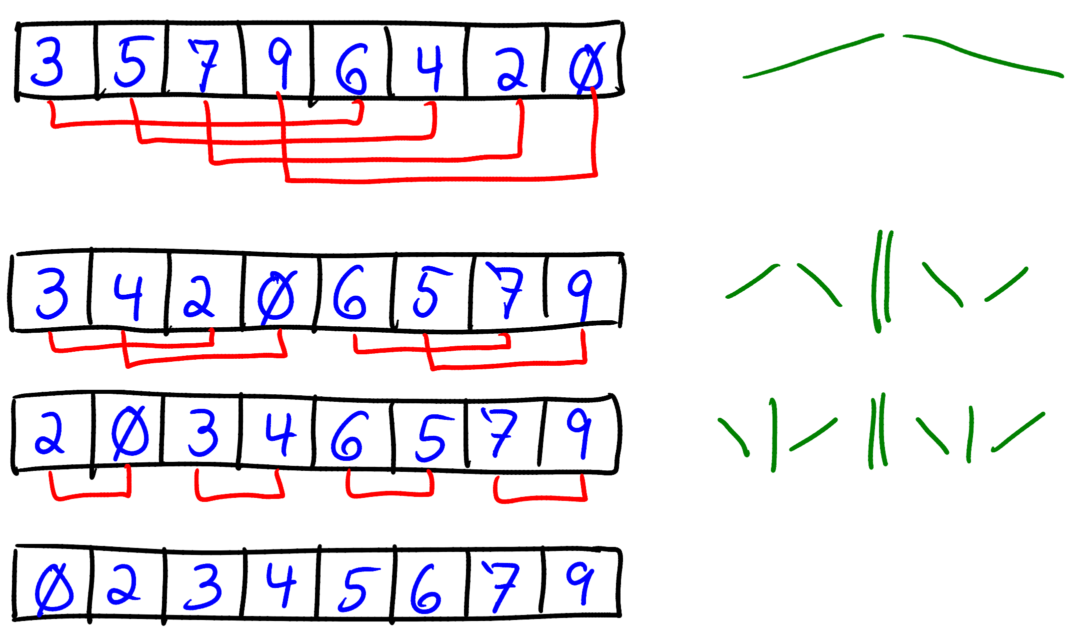
Bitonic sorting: (from parallel computing world)

- sequence of numbers is bitonic if it is a cyclic shift of nondecreasing + nonincreasing seq.

i.e.:  or  etc.

Bitonic sorting algorithm: (sorting network)

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



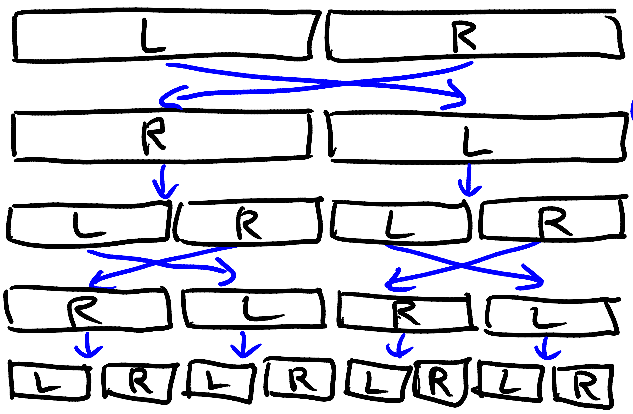
Invariant after round: [CLRS]

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

$O(\lg n)$ rounds

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

1. reverse order of second word in $O(\lg k)$ time
   - idea: $\mathrm{rev}(LR) = \mathrm{rev}(R)\,\mathrm{rev}(L)$ ~ recurse in parallel

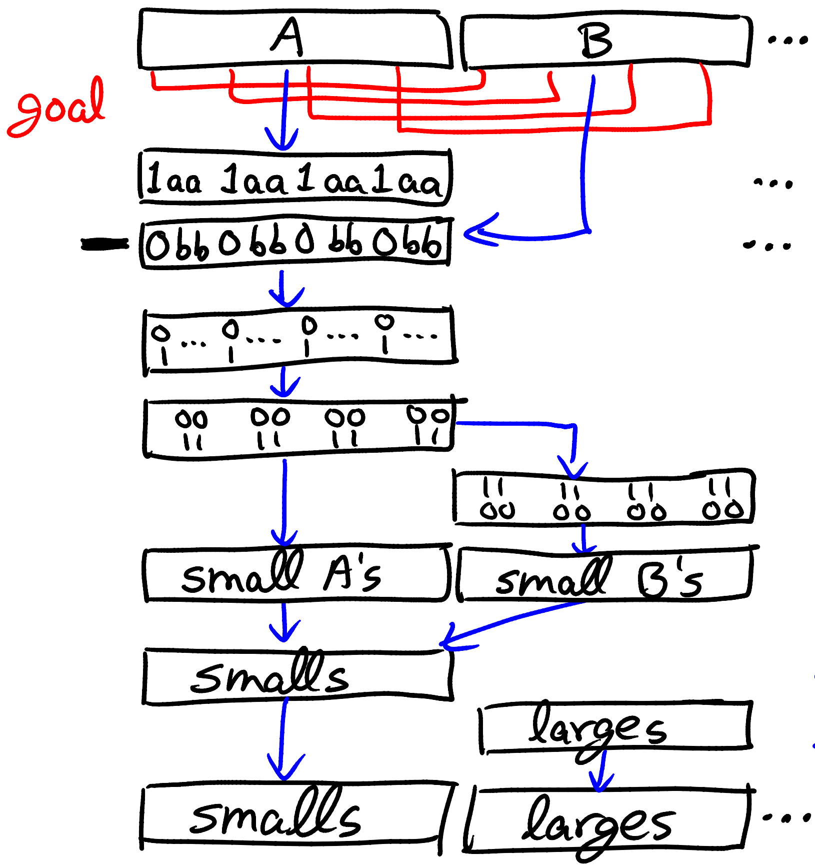


(mask L, shift right k/2) OR (mask R, shift left k/2)

ditto, with shifts of k/4

etc.

2. concatenate two words ⇒ bitonic
3. bitonic sort, each round in $O(1)$ time:



... mask A, set lead bits

... mask B, shift

subtract: 0 ⇒ B smaller
1 ⇒ A smaller

convert to mask
(shift right, subtract from self)

negate, shift

mask with A, B; OR

shift small B's, OR

similar

OR

Priority queues:

- $O(n\,S(n,w))$ sorting algorithm $\Rightarrow$ $O(S(n,w))$ worst-case priority queue — [Thorup — FOCS 2002]
  insert, delete, find-min
- $O(P(n,w))$ priority queue $\Rightarrow$ $O(P(n,w)\alpha(n))$ meldable priority queue — [Mendelson, Thorup, Zwick — SODA 2004]
- α essentially removed [Mendelson, Tarjan, Thorup, Zwick — SWAT 2004]
- $O(n\,S(n,w))$ sorting $\Rightarrow$ $O(S(n,w))$ delete-min & $O(1)$ decrease-key & insert ? — [Demaine & Pătraşcu — sketch 2005]