Integer sorting

Word RAM model

Memory is divided into an array of size S, each entry of the array is a word of w bits. Normally, we also have $2^w \geq S$, so that a pointer can fit in a word.

The processor may perform standard operations on words in $O(1)$ time:

- arithmetic ops: $+, -, *, /, \%$
- comparisons: $<, >, ==$
- bitwise ops: and, or, xor, $<<, >>$.

Problem: Sort n w-bit integers.

Recall: van Emde Boas tree: $O(\log w)$ ins/pred

sorting in $O(n \log w)$ time.

$O(n \log\log n)$ time when $w \leq \log^{O(1)} n$.

Today: Signature sort: $O(n)$ time when $w \geq \log^{2+\varepsilon} n$ for const $\varepsilon > 0$.

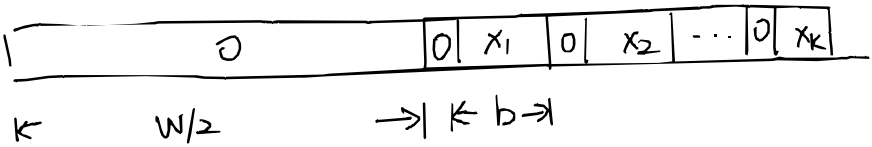
Combining the two: $O(n \log\log n)$ time for any w.

Packed sorting

Sort n b-bit integers in $O(n)$ time, for

$$b \leq \frac{w}{2 \log n \log\log n} - 1.$$

This bound allows us to pack $k = \log n \log\log n$ numbers in one word, spaced out by a 0-bit, and leaving first $\frac{w}{2}$ bits 0.



one word.

Subroutine 1: Given two words consisting of $m \leq k$ sorted b-bit integers, they can be merged in $O(\log m)$ time.

Omit this proof. Assume we have it now.

Subroutine 2: Sort k integers packed in one word in $O(k)$ time.

This can be done with a merge sort:

- split the integers into two, with $k/2$ each, using $O(1)$ bitwise ops.
- recurse on each
- merge using Subroutine 1.

Let $T(k)$ be the time to sort k ints.

$$T(k) = 2T(k/2) + O(\log k).$$

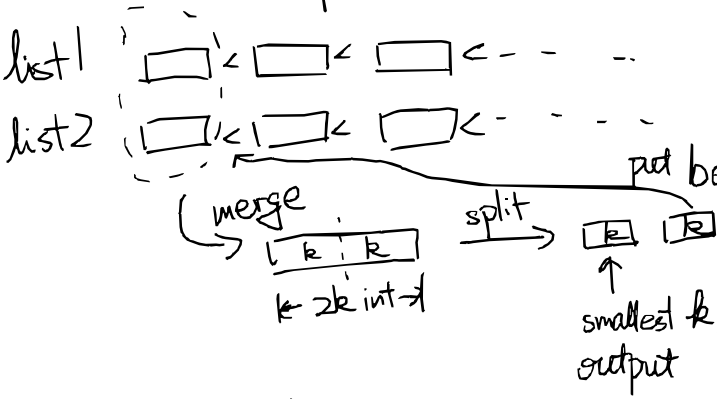
$$\Rightarrow T(k) = O(k).$$

Subroutine 3: Given two lists of r sorted words of $r \cdot k$ ints, merge them into one list of $2r$ sorted words in $O(r \log k)$ time.

Similar to merge in merge sort:

- take first word from both lists
- use Subroutine 1 to merge them into a word of $2k$ ints
- then split into 2 words of k ints
- the smallest must contain the smallest k ints overall add it to the output
- put the large back to the list that initially contained the max int in this word.

- repeat until all words are processed.



$O(\log k)$ time to output one word.

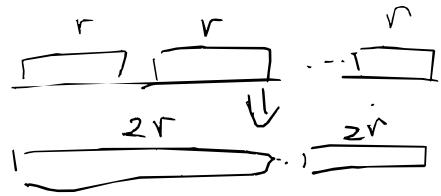
$O(r \log k)$ total time.

Thm: Let $w \geq 2(b+1)\log n \log\log n$. n integers can be sorted in $O(n)$ time.

Proof. ① Pack n integers into $\frac{n}{k}$ words, $k = \log n / \log\log n$.

② Use subroutine 2 to sort each word.

③ Then use subroutine 3 to iteratively merge them into longer sorted lists.



① $O(n)$ time

② $O(\frac{n}{k}) \cdot O(k) = O(n)$

③ Each level takes $O(\frac{n}{rk}) \cdot O(r \log k) = O(\frac{n \log k}{k}) = O(\frac{n}{\log n})$

levels is $O(\log n)$

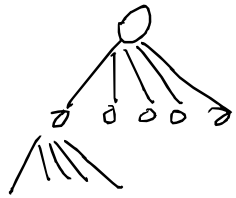
Total time is $O(n)$.

Signature sort

Now sort n w-bit integers for $w \geq \log^{2+\varepsilon} n$.

1. break bits of each integer into $\log^{\varepsilon} n$ equal-sized chunks
2. replace each chunk by an $O(\log n)$-bit hash. each integer becomes $O(\log^{1+\varepsilon} n)$ bits.
   Note: the hashes do not preserve order, only preserves identity.
   each integer can be processed into such an $O(\log^{1+\varepsilon} n)$-bit "signature" in $O(1)$ time using bitwise operations. (omit)
3. sort the signatures using packed sorting, in $O(n)$ time.
4. need to fix the order of hashed chucks.

build a compressed trie.



o o o signatures

Similar to suffix tree

- only keep the ancestors of the signatures
- "compress" nonbranching paths, and label with "i-th chunk to j-th chunk of signature #k".

Similar to constructing suffix tree from suffix array in $O(n)$ time.

- This compressed trie can also be constructed in $O(n)$ time.

5. recursively sort the edges of the trie based on their actual value. Write each edge as a triple (node ID, actual chunk value, edge index)

This takes $O(\frac{w}{\log^{\varepsilon} n} + \log n)$ bits to write down.

recursively sort these n edges.

reduced # bits in each integer $w \mapsto O(\frac{w}{\log^{\varepsilon} n} + \log n)$

after $O(1/\varepsilon)$ levels of recursion,

bits in each int becomes $b \leq O(\frac{w}{\log n \log\log n})$

Then can apply packed sorting as base case.

6. Permute the edge of each node to get back the actual order

7. Inorder traversal to get the sorted list from the leaves.