Sorting when there are only O(log n) many different numbers

We have n integers with lot's of repeated numbers. In this list, the number of distinct elements is $O(\log n)$. What's the best asymptotic number of comparisons for sorting this list?

Any idea or hint or pseudo code? In fact I want to learn pseudo code.

algorithms sorting

edited Aug 17 '14 at 14:52
Raphael ♦
42.3k 13 104 225

asked Aug 17 '14 at 14:07
user3661613
42 5

2 What have you tried and where did you get stuck? For instance, which of the well-known sorting algorithms are affected by duplicates, and have you ideas on how to fix those that are? Do you have reason to believe that you can do any better? – Raphael ♦ Aug 17 '14 at 14:53

Regarding the question: do you allow algorithms *tailored* to this situation, or do they have to perform within certain bounds in general, too? – Raphael ♦ Aug 17 '14 at 19:47

1 Answer

Because you asked for minimum number of comparisons, so I assume the algorithm can only compare the numbers.

The idea is to extend the sorting lower bound argument. Assume you want to sort n elements knowing there exist at most k distinct values. There are $n!$ ways to permute the elements, but many of them are equivalent. If there are n_i element of the ith values. Each permutation is equivalent to $\prod_{i=1}^k n_i!$ other permutations. So the total number of distinct permutations is

$$\frac{n!}{\prod_{i=1}^k n_i!}$$

The number of required comparisons is bounded below by

$$\log_2\left(n!/\min\{\prod_{i=1}^k n_i! | \sum_{i=1}^k n_i = n, n_i \geq 0 \text{ for all } i\}\right)$$

Good thing that minimization part can be easily shown by extend factorial to the continuous domain. $\min\{\prod_{i=1}^k n_i! | \sum_{i=1}^k n_i = n, n_i \geq 0 \text{ for all } i\}$ is attained when $n_i = n/k$. (note the $\log$ in the next computation is base e for convenience)

$$\log(n!/(n/k)!^k) = \log(n!) - k\log((n/k)!) = n\log(n) - n\log(n/k) \\ + O(\log n) = n(\log(n) - \log(n) + \log(k)) + O(\log n) = \Omega(n \log k)$$

$\log(n!) = n\log n - n + O(\log n)$ is Ramanujan's approximation.

To get an upper bound. Just consider storing the unique values in a binary search tree, and each insert we either increase the number of occurrence of an element in the BST, or insert a new element into the BST. Finally, print the output from the BST. This would take $O(n \log k)$ time.

Since both the lower bound and upper bound works for all k, the algorithm would take $O(n \log\log n)$ time for your problem.

Addendum:

I just figured out from @Pseudonym's comment that this proof also proves that we need at least nH comparisons where H is the entropy of the alphabet, so I might as well add this to the answer.

Let $c = \log 2$ and $p_i = n_i/n$. The entropy of the alphabet where the ith letter appears n_i time is $H = -\sum p_i \log_2 p_i$.
$nH = -\sum n_i(\log_2(n_i) - \log_2(n)) = \sum n_i(\log_2(n) - \log_2(n_i)) = c.$
$\sum n_i(\log(n) - \log(n_i))$

$$
\begin{aligned}
\log_2\left(n!/\prod_{i=1}^{k} n_i!\right) &= \log_2(n!) - \sum_{i=1}^{k} \log_2(n_i!) \\
&= \log_2(n!) - \sum_{i=1}^{k} \log_2(n_i!) \\
&= c(\log(n!) - \sum_{i=1}^{k} \log(n_i!)) \\
&= c(n\log n - n + O(\log n) - \sum_{i=1}^{k} n_i \log(n_i) - n_i + O(\log n_i)) \\
&\geq c(n\log n - n - \sum_{i=1}^{k} n_i \log(n_i) - n_i) \\
&= c(-n + \sum_{i=1}^{k} n_i(\log(n) - \log(n_i)) + n_i) \\
&= c(\sum_{i=1}^{k} n_i(\log(n) - \log(n_i))) \\
&= nH
\end{aligned}
$$

edited Aug 19 '14 at 0:53

answered Aug 17 '14 at 20:55

Chao Xu
1,501 6 23

2 Great answer! One thing I'd add: If you want a practical algorithm, Bentley & McIlroy's variant of quicksort with ternary partitioning would achieve this lower bound for this type of problem (for non-pathological input, because this is quicksort we're talking about). citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.8162 – Pseudonym Aug 18 '14 at 2:51

2 One more thing while I think about it. There's a useful theorem that comparison-based sorting takes at least $nH - n$ comparisons, where H is the entropy of the key distribution. That's another way to derive this result. – Pseudonym Aug 18 '14 at 2:56

Dear @Pseudonym, with your answer H=log n and result is n log n ? but the last result is n lg lg n? – user3661613 Aug 18 '14 at 8:29

1 $H = \sum_i -p_i \log p_i$ where i ranges over the unique keys and p_i is the probability that an element has key i. If there are $\log n$ unique keys distributed evenly, then $p_i = \frac{1}{\log n}$, and so $H = \sum_{i=1}^{\log n} -\frac{1}{\log n} \log \frac{1}{\log n} = \log\log n$ – Pseudonym Aug 18 '14 at 23:23

Do you have a reference for $nH - n$ lowerbound? I got that we must use at least nH comparisons. I might have missed something. – Chao Xu Aug 19 '14 at 0:49

Nice proof! Very elegant. IIRC, the $-n$ term probably comes from using more terms in Stirling's approximation. – Pseudonym Aug 19 '14 at 1:24