COMPARISON MODEL AND SORT IN LINEAR TIME

HAOCHEN

1. Comparison model

In this model, we restrict the operations. Three distinguished features are following:

- inputs are abstract.
- comparison is the only operation.
- Time cost = # of comparison.

We would like to proof the lower bound for search and sort. And then we switch to RAM model to achieve $O(n)$ for sorting.

2. Lower bound proof

Let's take a close look at the comparison model. In binary search problem, it's convenient to use **decision tree** to sketch the whole procedures.

In decision tree:

- Each node denotes as decision time.
- Each path denotes as execution step.
- Each leaf denotes as an outcome.

picture needed.

In this abstract tree, we find that the longest path from root to a leaf is actually the worst-case comparison, and that is the height of the tree.

Now the question is how to know the height. Since we can easily know the number of leaves, so if we find some relation between height and number of leaves, it might be helpful.

For convenience, h denote height, l denotes numbers of (reachable[1]) leaves.Consider a complete tree, $l = 2^h$, which is an obvious fact in tree structure. And for incomplete tree, $l < 2^h$, that brings lower bound for h. Now let's state it formally:

Theorem 1. *In comparison model, any search algorithm requires $\Omega(log(n))$ comparisons in the worst case. And sort requires $\Omega(nlog(n))$.*

Proof. From the preceding discussion,

$$l \leq 2^h$$

and in search , there are total $n + 1$ outcomes, so we have

$$n + 1 = l$$

combine these two,

$$n < 2^h$$

[1]Notice this tree might not be complete, and this implies a bound, not an exact solution.

which yields

$$h > lg(n)$$

as desired.

Similar argument for sort, just replace $n! = l$,using Stirling's formula or scaling trick, it finally yields $h > nlog(n)$. □

Corollary 2. *Heap sort and merge sort are asymptotically optimal comparison sorts.*

Proof. $O(nlg(n))$ upper bounds heap sort and merge sort, and also lower bounds these two by 1. □

3. Counting sort

Can we improve? Well, since we have achieve optimal, we have to change our assumed model to RAM. Then we have counting sort which takes n element array with keys in the range of $\{1, 2, ..., k\}$and sort in $O(n + k)$ time.

What is worth mentioning is that this is a **stable** sort. When elements ties, a stable sort can arrange them as the order in the input array. This property is important for radix sort which applied in many spreadsheet sort cases.

The hard problem for counting sort is to make it stable. We scan through the input array from left to right, but when meet the same elements, we somehow want to mark which is previous one and which is recent one.We could record the position and increase the recorded number when meet the same element, then we use these records as index to prevent overwrite in the output array.

code needed.

4. Radix sort

We make each item d digits. It sorts from the least significant place to the most significant place. And to preserve all the digits in one item, we need stable sort such as counting sort.

The complexity for each place is $O(n + k)$ if using counting sort in each place. So the total complexity is $O(d(n + k))$. Asymptotically, we have

$$d = O(1)$$
$$dn = O(n)$$

and

$$k = O(n)$$

this yields total complexity $O(n)$, This is an improvement as previous $O(nlg(n))$ in comparison model.