

CS447 Computer Algorithms. Notes 1. Fall 2020

Consider the following standard sorting problem.

SORTING

Input: An array A of N integers.

Output: The array A sorted in ascending order.

As in searching, we use the *comparison-based model* where we charge *one unit* of cost for every *comparison* made between elements of the array, while any other operation is free.

Consider a sorting algorithm (called *Insertion Sort* which works in the following manner:

1. Sort the first $N - 1$ items (recursively).
2. Insert the N th item into the sorted array of $N - 1$ items.

Let us define $T(N)$ to be the number of comparisons made on an array of size N . Then, we have the following recurrence:

$$\begin{aligned}T(N) &= T(N - 1) + N, & \text{if } N > 2 \\T(2) &= 1 \\T(1) &= 0\end{aligned}$$

Here, we assume we use a naive insert for the second step (and not, say, Binary Search). By unrolling, we see that

$$\begin{aligned}T(N) &= T(N - 1) + N \\&= T(N - 2) + (N - 1) + N \\&= \dots \\&= \sum_{k=1}^N k.\end{aligned}$$

Note the summation is an *arithmetic sum* whose closed-form solution is

$$\sum_{k=1}^N k = \frac{(N - 1)N}{2} = O(N^2). \tag{1}$$

A similar (or dual) algorithm (called *Selection Sort*) sorts by first searching for the Maximum item of the array on N items and then recursively sort the other elements. This algorithm has the same recurrence profile as Insertion Sort and thus has complexity $O(N^2)$.

Consider now an algorithm called *Mergesort* which works as follows:

1. Split the array into the left half and the right half (each of size $N/2$) and sort them recursively.
2. Merge the two sorted halves into one sorted array.

The recurrence for this algorithm is

$$\begin{aligned} T(N) &= 2T(N/2) + N, & \text{if } N > 2 \\ T(2) &= 1 \\ T(1) &= 0 \end{aligned}$$

By unrolling, the recurrence has the following property:

$$T(N) = 2^k T(N/2^k) + kN \quad (2)$$

and if $k = \log N$, then we have $T(N) = O(N \log N)$.

Finally, consider a sorting algorithm called *Quicksort* which works as follows:

1. Select a pivot m which is an element of the input array.
2. Partition the input array A into the left half (which contains all items less or equal to the pivot) and the right half (which contains all items greater than the pivot).
3. Sort the two halves recursively.

An optimal choice for the pivot is the *median* element of the input array. Since there is a linear-time algorithm for *Median finding* (due to Blum, Floyd, Pratt, Rivest and Tarjan), the recurrence for Quicksort (with median pivot) is similar to Mergesort. Hence, Quicksort (with median pivot) also has complexity $O(N \log N)$.

***K*-Selection Problem** Consider the problem of searching for the K th smallest element in an array of size N . In particular, $K = 1$ is the Minimum finding problem, $K = N$ is the Maximum finding problem, and $K = N/2$ is the Median finding problem. A naive algorithm for *K*-Selection is to repeatedly find the Minimum for K iterations. But, this costs KN or $O(N^2)$ for the Median finding problem. We describe an algorithm that costs $O(N)$ (independent of K).

Algorithm (Blum, Floyd, Pratt, Rivest and Tarjan):

1. Partition the input array into $N/5$ blocks each of size 5 items.
2. Sort each block separately.
3. Find the median m of the $N/5$ median elements.
4. Partition the input array into the left-half L (which contains all items less or equal to m) and the right-half R (which contains all items greater than m).
5. Based on K , recurse appropriately to either L or R or else return m .

Given the choice of m , it can be shown that the sizes of L and R must be at least $3N/10$. Therefore, in the worst-case, the recurrence of the above algorithm is given by

$$\begin{aligned} T(N) &= T(N/5) + T(7N/10) + N, \\ T(1) &= O(1) \end{aligned}$$

By induction, we can establish that $T(N) \leq 10N$. This shows that $T(N) = O(N)$.

Lower bounds A comparison-based sorting algorithm may be viewed as a decision tree where each internal node is labeled with a pair of indices of elements that are compared and each leaf node is labeled with a permutation on N elements (corresponding to the solution to the sorting problem). Each directed edge from an internal node to its children is labeled with Yes or No (corresponding to the result of the comparison).

The number of leaves of in a correct decision tree for sorting is at least $N!$ since there are $N!$ possible permutations of the input array. Given that the decision tree is a binary tree, its depth must be at least $\log(N!) = \Omega(N \log N)$. This shows that both Mergesort and Quicksort are (asymptotically) *optimal* abstract sorting algorithms.

Questions Determine if the following statements are true or false:

1. (T/F) There are $O(N)$ time algorithms for sorting arrays of integers.
2. (T/F) A sorting algorithm is called *stable* if it preserves the relative positioning of items which have equal values. All sorting algorithm discussed above are stable.
3. (T/F) The median finding algorithm is linear-time only for blocks of size 5.