

CS447 Computer Algorithms. Notes 0. Fall 2020

Consider the following basic search problem.

MINMAX

Input: An array A of N integers.

Output: The smallest and largest elements of A .

In any algorithm for the above problem, we charge *one unit* of cost for every *comparison* made between elements of the array, while any other operation is free.

1. (T/F) There is a simple algorithm for finding the smallest element (MIN) of an array of size N that makes $N - 1$ comparisons.
2. (T/F) There is a simple algorithm for finding the largest element (MAX) of an array of size N that makes $N - 1$ comparisons.
3. (T/F) Any algorithm to find the smallest element of an array of size N must make at least $N - 1$ comparisons.
4. (T/F) There is a simple algorithm for finding the smallest and largest elements (MINMAX) of an array of size N that makes $2N - 3$ comparisons.
5. (T/F) The algorithm that sorts the input and returns the elements at the first and last position has complexity $\omega(N)$.

Here, we describe a simple recursive algorithm for MINMAX that makes fewer than $2N - 3$ comparisons. The algorithm is as follows. If $N = 1$, the algorithm does not need to make any comparisons. If $N = 2$, the algorithm makes one comparison and will find which element is the smallest and which is the largest. In general, if $N \geq 2$, the algorithm partitions the array of size N into two halves where each half is of size $N/2$. It recursively finds the MINMAX on the left half and the MIXMAX on the right half. It then compares the two MINs and the two MAXs to obtain the final answers.

Next, we analyze the cost of the above recursive algorithm for MINMAX. Let us define $T(N)$ to be the number of comparisons made on an array of size N . Then, we have the following recurrence:

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

To solve this recurrence, we “unroll” this. For simplicity, assume that N is a power of two. Therefore

$$\begin{aligned}T(N) &= 2T(N/2) - 2 \\&= 2(2T(N/4) - 2) - 2 \\&= \dots \\&= 2^k T(N/2^k) - 2 \sum_{i=0}^{k-1} 2^i.\end{aligned}$$

Note the summation is a geometric sum. Recall that if $\alpha \neq 1$, we have $\sum_{i=0}^{k-1} \alpha^i = \frac{\alpha^k - 1}{\alpha - 1}$. Thus, by letting $\alpha = 2$ and $N/2^k = 2$ (or $N = 2^{k+1}$), we obtain

$$T(N) = \frac{N}{2}T(2) + 2(2^k - 1) = \frac{N}{2} + 2\left(\frac{N}{2} - 1\right) = \frac{3N}{2} - 2.$$

So, the recursive algorithm makes $\frac{3}{2}N - 2$ comparisons to find the smallest and largest elements in an array of size N .

1. (T/F) The divide-conquer algorithm works even if N is not a power of two.
2. (T/F) There is an iterative algorithm (not recursive) with the same complexity as the divide-conquer algorithm.