

Solutions to Practice Problems for Homework 3

1. The adversary will begin by creating the following $n + 1$ possibilities for the array A . First for each $i \in \{0, 1, \dots, n - 1\}$ a sorted array is created in which $A[i] = x$ (and all other elements are different than x). Finally, a sorted array is created that does not contain x . Let L be a list holding these $n + 1$ possible values for A . For example, when $n = 6$ and $x = 10$ the list L could contain the 7 possible inputs:

$$\langle 10, 11, 12, 13, 14, 15 \rangle, \langle 9, 10, 11, 12, 13, 14 \rangle, \langle 8, 9, 10, 11, 12, 13 \rangle, \langle 7, 8, 9, 10, 11, 12 \rangle, \\ \langle 6, 7, 8, 9, 10, 11 \rangle, \langle 5, 6, 7, 8, 9, 10 \rangle, \langle 4, 5, 6, 7, 8, 9 \rangle.$$

Whenever the algorithm asks to compare x to $A[i]$, the adversary examines all arrays in L and counts how many have $x < A[i]$, how many have $x = A[i]$, and how many have $x > A[i]$. It will answer in accordance to the answer that occurs most frequently (with ties broken arbitrarily). Observe that $x = A[i]$ could only occur once. Thus the most frequent answer must occur in at least $\lceil (n - 1)/2 \rceil$ of the arrays in L . From this it follows that the adversary can guarantee that $|L| \geq 2$ until $\lceil \log_2(n + 1) \rceil$ comparisons have been made. Since a correct algorithm cannot be done until $|L| = 1$, it follows that any correct algorithm for this problem must execute at least $\lceil \log_2(n + 1) \rceil$ statements leading to a $\Omega(\log n)$ lower bound.

Note: A slightly simpler argument argues that the most frequent answer occurs in at least $\lceil |L|/3 \rceil$ of the arrays in L . This leads to a weaker (though asymptotically equivalent) lower bound of $\lceil \log_3 n \rceil$.

2. An array can be placed into the list L for each distinct answer. Observe that each answer can be viewed as a string that is a sequence (possibly empty) of “1”s corresponding to elements less than x and y (inclusive), followed by a sequence (possibly empty) of “2”s corresponding to elements between x and y , and finally a sequence (possibly empty) of “3”s corresponding to elements greater than x and y . Each such bit string is defined by the positions where you switch from 1 to 2 and where you switch from 2 to 3. Hence we must count the number of ways to arrange 2 dividers with n array elements. Equivalently, we must count the number of ways to position 2 dividers among $(n + 2)$ slots. Hence the adversary can create a list L of $C(n + 2, 2) = (n + 2)(n + 1)/2$ inputs with distinct answers among with the algorithm must force the adversary to eliminate all but one.

The adversary answers each question based on the majority of the elements in L . So with each comparison the adversary can guarantee that at least $|L|/2$ of these possible inputs remain. Thus the number of comparisons that the adversary can force any algorithm to make before $|L| = 1$ is $\lceil \log_2(n + 2)(n + 1)/2 \rceil = \lceil \log_2(n + 2) + \log_2(n + 1) - 1 \rceil$. So any comparison-based algorithm for this problem takes $\Omega(\log n)$ time.

3. Suppose you are given the task to sort one thousand 32-bit keys. You have decided to use radix sort for this problem and want to decide how many bits each radix sort digit. Which is best among having 1 bit per radix sort digit, 4 bits per radix sort digit, 8 bits per radix sort digit or 16 bits per radix sort digit? You are provided with a counting sort procedure with exact time complexity of $5n + 4k$. Show your work.

bits/radix digit	k	# digits	$d(5n + 4k) = d(5000 + 4k)$
1	2	$32/1 = 32$	$32(5000 + 4 \cdot 2) = 160,256$
4	$2^4 = 16$	$32/4 = 8$	$8(5000 + 4 \cdot 16) = 40,512$
8	$2^8 = 256$	$32/8 = 4$	$4(5000 + 4 \cdot 256) = 24,096$
16	$2^{16} = 65536$	$32/16 = 2$	$2(5000 + 4 \cdot 65536) = 534,288$

So of the given choices, the best is to pick 8 bits per radix digit.

4. Give the asymptotically fastest algorithm you can to sort n integers in the range of 0 to $(n^4) - 1$. You should give a very clear and complete high-level description of your algorithm. Be sure to analyze the time complexity of your algorithm as a function of n . You are NOT restricted to use a comparison sorting algorithm (although are welcome to if you want).

Here is an $O(n)$ algorithm for this problem. Since any sorting algorithm must at least access each integer, this algorithm is asymptotically optimal.

We use radix sort where we represent each number as a 4 digit base- n number. Thus the time complexity is $O(d(n + k)) = O(4(n + n)) = O(n)$ since $d = 4$ and $k = n$.

If you saw the above solution right away that's great. Here's a way to have solved this otherwise. Each element requires roughly $\log_2(n^4) = 4 \log_2 n$ bits. (The exact number is $\lceil 4 \log_2 n \rceil$). Suppose you grouped b bits per digit. Then there would be $d = \lceil (4 \log_2 n)/b \rceil$ digits each which takes on one of 2^b values. Hence the time complexity for radix sort is

$$O(d(n + k)) = O\left(\frac{4 \log_2 n}{b}(n + 2^b)\right).$$

At this point you could minimize this function with respect to b . However, if we think about it a little, we can find a value of b that gives us an asymptotically optimal solutions. Since, the time complexity has the term $(n + 2^b)$, asymptotically we can't do better than picking b such that $2^b = n$ (and hence $b = \log_2 n$). By doing this we obtain a time complexity of $O(4(2n)) = O(n)$ which is clearly asymptotically optimal. Once you've done this, you can then look back and see that you picked a base- n representation and thus give the two line solution given above.