

## Solutions to Practice Problems for Homework 3

1. Prove the best lower bound you can (using the decision tree technique) on the number of distance computations needed to find the closest pair of  $n$  points in the plane under the model of computation in which you cannot directly access the coordinates of a point but instead can just compute the distance between two points.

There is an different answer corresponding to each pair of points. So, one way to count the number of answers is to note that the first point could be matched with any of the remaining  $n - 1$  points, then we could match the second point with the remaining  $n - 2$  points and so on. Thus the number of leaves must be at least

$$(n - 1) + (n - 2) + \cdots + 1 = \sum_{i=1}^{n-1} i = \frac{n(n - 1)}{2} \geq \frac{(n - 1)^2}{2}.$$

Suppose there were less leaves. Then for two different pairs of points  $p, q$  and  $p', q'$ , they would end in the same leaf and the answer output could not be the right answer for both inputs contradicting that the algorithm is correct.

Thus the number of comparisons made over distance computations (and thus the time complexity) is at least  $\log_2 \frac{(n-1)^2}{2} = 2 \log_2(n - 1) - 1 = \Omega(\log n)$ .

2. Prove the best lower bound you can (using the decision tree technique) on the time complexity of a comparison based algorithm for the following problem: You are given a sorted array  $A$  (of  $n$  elements) and two integers  $x$  and  $y$  where  $x \leq y$ . The algorithm is required to compute how many elements in  $A$  are less than both  $x$  and  $y$ , how many elements of  $A$  are between  $x$  and  $y$  (inclusive), and how many elements of  $A$  are bigger than both  $x$  and  $y$ . (Integers  $x$  and  $y$  are not necessarily in  $A$ .)

Each leaf can be viewed as a string that is a sequence (possibly empty) of “1”s corresponding to elements less than  $x$  and  $y$ , 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  $y$ . So the number of leaves is at least the number of bit strings of this form. 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 there are at least  $C(n + 2, 2) = (n + 2)(n + 1)/2$  leaves. Thus  $h \geq \log_2(n + 2)(n + 1)/2 = \log_2(n + 2) + \log_2(n + 1) - 1$ . 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.