Hao's Guide - Algorithm Crash Course - Questions Walkthrough

Introduction

Below contain a number of written out walktthrough solutions of some selected tutorial questions that I best believe explains and trains the various lessons and mindsets depending on the topic. A lot of these were assignment questions from when I studied these topics.

Also note that the answers I have written are to demonstrate the thought process, don't let their "word count" intimidate you!

I wholeheartedly recommend attempting these yourselves before you looking at the answer (or not, I'm not your mother).

Divide and Conquer

Question - Skipped Numbers

> You are given an array containing a sequence of 2^n − 1 consecutive positive integers starting with 1
> except that one number was skipped; thus the sequence is of the form 1, 2, 3, . . . , k − 1, k + 1, . . . , 2n.
> You have to determine the missing term accessing at most O(n) many elements of A.

With every divide and conquer question, the first step is to figure out the naive method. Since we need to find the skip number we would need to find the place index where the number is skipped.

Since we have array A, it would stand to reason that for all element prior to the skip value (let's call it **s**), the difference between the index and the value is of 1. However, every number after the skip value would have a different of 2.

Example array A: [1,2,3,4,5,6,7,9,10,11,12,13,14,15,16]
(Skipped value is 8)

Before the skip: `A[0] = 1, A[1] = 2, A[2] = 3, hence A[b] = b+1`
After the skip: `A[7] = 9, A[8] = 10, A[9] = 11, hence A[a] = a+2`

Hence the naive solution:

> Scan every index from start to finish until you find 2 index position, where the difference between the index and the element is changes from 1 to 2. Time complexity is O(2^n)

However, the problem is that you are scanning 2^n elements (re-read the question).

Notice that you are searching a value in an ordered data set. The first thing you should think of is "binary search".

So lets try binary search, but what are the partition conditions?
You know that if the current index/element difference is 2, every subsequent values would also have to be 2. You also know that if the current index/element difference is 1, every prior value will have a difference to 1. So checking those values are redundant.

Hence you are trying to find indices where the index/difference is transits from 1 to 2.

Optimised solution:

> Apply a binary search algorithm on the array, where we drop the upper partition when the index/difference of the current index is 2 or we drop the lower partition when the index/difference of the current index is 1.

Since the binary search algorithm is O(log(n)) and the array size is 2^n. The final time complexity is O(log(2^n)), which simplifies to O(n)

Note that this question is a tricky one for it primes you into thinking linear search due to the expected complexity of O(n). But remember that the complexity is not simply dependent on the algorithm but on the data set size being operated on as well.

Question - Distinct Pairs

> You are given an array A of n distinct positive integers.
>
> 1. Design an algorithm which decides in time O(n^2 log n) (in the worst case) if there exist four
>   distinct pairs of integers {m, s} and {k, p} in A such that m^2 + s = k + p^2
> 2. Solve the same problem but with an algorithm which runs in the expected time of O(n^2).

Part 1

Naive solution:

> We test ever combination of 4 numbers in array. Since we have n distinct integers, that would mean n * (n-1) * (n-2) * (n-3) combination. Which would mean your time complexity would be, asymptotically, O(n^4)

Well that was terrible but let's have a re-read of the question. There are thing that you may have noticed.

The equation (m^2 + s = k + p^2), the LHS and the RHS are actually identical, albeit written in a different order.
So let's firstly re-write it: (m^2 + s = p^2 + k). Lets generalise this formula structure as (a^2 + b)

But so what? Well, since they have the same structure, we don'tt have to search every 4 number combinations but instead look at every 2 number combination and check whether numbers produce the same 2 answer.

So if we test every combination of 2 numbers instead, that brings our search space to n^2. We then calculate the result of generalised formula (a^2 + b) with every combination of numbers, and lets place it in an array R (for result). If any 2 combination are equal and their numbers are unque, it would suggest that these 4 numbers are unique. This will take time complexity O(n^2).

To search, we can simply sort the result array (keeping track of which 2 combination of number were used, maybe using a tuple or something), and linearly scan through the array and checking for duplicates. The sort is O(n^2 log (n^2)), which by way of logartihmic laws, becomes O(n^2 log(n)). The linear search is O(n^2).

At each result value that are the same, you check the 4 integers between the two values for duplicates. If you do find duplicates integers, you continue the search. If none of them are duplicates, 4 distinct integer exist to satisfy the original formula. Since it's a comparison between 4 values regardless of space, that would mean time complexity is O(4), which becomes O(1).

Since operations are carried out in consecutive order and the O(n^2 log(n)) is the greatest time complexity. The time complexity of this algorithm is O(n^2 log(n))

Part 2

As you can see with this question, they indicate they want an **expected time** of O(n). The one data structure that is commonly referred to is hashmaps.

Since hashmap improves our lookup time, notice that initial answer too an additional log(n) from the merge sort. So instead, store every 2 number combination in a hashmap, where their key is their result when passed into the equation (a^2 + b).

Then, check by brute force, every element of the n^2 combination pairs until a pair is found to share the same result and have distinct integers.

Hence the time complexity is O(n^2) isntead

Greedy