Algorithms (CS2443) : Problem Set 1

Department of Computer Science and Engineering
IIT Hyderabad

Please read the following comment before you work on the problems.

This is a set of questions that you should solve to master the contents taught in the class. *You don't have to submit the solutions*! But you should still solve them to understand the material, and also because these problems or their variants have a devilish way of finding their way in exams/quizzes!

We suggest to try to solve them from scratch under exam conditions—by yourself, *without* your notes, *without* the internet, and if possible, even without a cheat sheet.

If you find yourself getting stuck on a problem, try to figure out *why* you're stuck. Do you understand the problem statement? Are you stuck on choosing the right high-level approach, or are you stuck on the details? Are you solving the right (recursive) subproblems?

Discussing problems with other people (in your study groups, with me, or on Google Classroom) and/or looking up old solutions can be *extremely* helpful, but **only after** you have (1) made a good-faith effort to solve the problem on your own, and (2) you have either a candidate solution or some idea about where you are getting stuck.

When you do discuss problems with other people, remember that your goal is not merely to "understand" the solution to any particular problem, but to become more comfortable with solving a certain type of problem on your own. Identify specific steps that you find problematic, read more about those steps, and focus your practice on those steps.

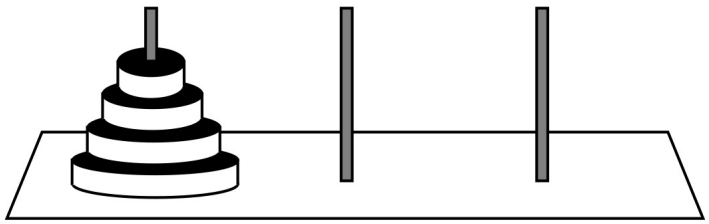

Figure 1: Tower of Hanoi

1. We are given a tower of n disks stacked on a peg from largest on the bottom to smallest on top as shown in Figure 1. We are also given two empty pegs. Our goal is to move the stack of n disks from peg 1 to peg 3, while obeying the following rules:

   - Only one disk may be moved at a time.
   - Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty peg.

- No disk may be placed on top of a disk that is smaller than it.

In other words, our objective is to transfer the entire tower to one of the other pegs, moving only one disk at a time and never moving a larger one onto a smaller.

Describe a recursive algorithm for achieving our objective. More importantly, compute how many moves does your algorithm make?

2. Consider the following more complex variant of the Tower of Hanoi puzzle. The puzzle has a row of k pegs, numbered from 1 to k. In a single turn, you are allowed to move the smallest disk on peg i to either peg $i-1$ or peg $i+1$, for any index i; as usual, you are not allowed to place a bigger disk on a smaller disk. Your mission is to move a stack of n disks from peg 1 to peg k.

   (a) Describe a recursive algorithm for the case $k = 3$. Exactly how many moves does your algorithm make?

   (b) Describe a recursive algorithm for the case $k = n+1$ that requires at most $O(n^3)$ moves.

   (c) Describe a recursive algorithm for the case $k = n+1$ that requires at most $O(n^2)$ moves.

   (d) Describe a recursive algorithm for the case $k = \sqrt{n}$ that requires at most a polynomial number of moves. (Which polynomial??)

   (e) Describe and analyze a recursive algorithm for arbitrary n and k. How small must k be (as a function of n) so that the number of moves is bounded by a polynomial in n?

3. Give asymptotic upper bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 10$. Make your bounds as tight as possible, and justify your answers.

   (a) $T(n) = 2T(n/4) + \sqrt{n}$

   (b) $T(n) = 2T(n/4) + n$

   (c) $T(n) = 2T(n/4) + n^2$

   (d) $T(n) = 3T(n/3) + \sqrt{n}$

   (e) $T(n) = 3T(n/3) + n$

   (f) $T(n) = 3T(n/3) + n^2$

   (g) $T(n) = 4T(n/2) + \sqrt{n}$

   (h) $T(n) = 4T(n/2) + n$

   (i) $T(n) = 4T(n/2) + n^2$

   (j) $T(n) = T(n/2) + T(n/3) + T(n/6) + n$

   (k) $T(n) = T(n/2) + 2T(n/3) + 3T(n/4) + n^2$

(l) $T(n) = T(n/15) + T(n/10) + 2T(n/6) + \sqrt{n}$

(m) $T(n) = 2T(n/2) + O(n \log n)$

(n) $T(n) = 2T(n/2) + O(n/\log n)$

(o) $T(n) = \sqrt{n}T(\sqrt{n}) + n$

(p) $T(n) = \sqrt{2n}T(\sqrt{2n}) + \sqrt{n}$

4. Suppose you are given a stack of n pancakes of different sizes. You want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation you can perform is a *flip* – insert a spatula under the top k pancakes, for some integer k between 1 and n, and flip them all over.

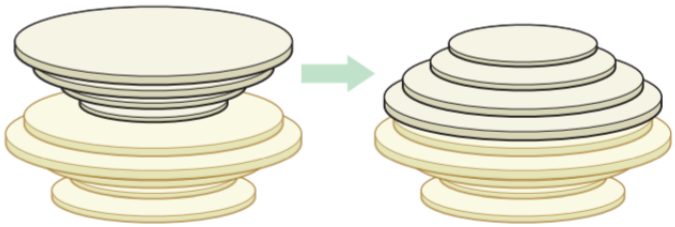

Figure 2: Flipping the top four pancakes.

   (a) Describe an algorithm to sort an arbitrary stack of n pancakes using $O(n)$ flips.

   (b) For every positive integer n, describe a stack of n pancakes that requires $\Omega(n)$ flips to sort.

5. An ***inversion*** in an array $A[1, \ldots, n]$ is a pair of indices (i, j) such that $i < j$ but $A[i] > A[j]$. The number of inversions in an n-element array is between 0 (if the array is sorted) and $\binom{n}{2}$ (if the array is sorted backward). Describe and analyze an algorithm to count the number of inversions in an n-element array in $O(n \log n)$ time.

6. Suppose you are given two sets of n points, one set $\{p_1, p_2, \ldots, p_n\}$ on the line $y = 0$ and the other set $\{q_1, q_2, \ldots, q_n\}$ on the line $y = 1$. Create a set of n line segments by connecting each point p_i to the corresponding point q_i. Describe and analyze a divide-and-conquer algorithm to determine how many pairs of these line segments intersect, in $O(n \log n)$ time.

7. Describe a $O(n \log n)$-time algorithm that, given a set S of n integers and another integer x, determines whether or not there exist two elements in S whose sum is exactly x.

8. Suppose we are given a set S of n items, each with a value and a weight. For any element $x \in S$, we define two subsets

- $S_{<x}$ is the set of elements of S whose value is less than the value of x.
- $S_{>x}$ is the set of elements of S whose value is more than the value of x.

For any subset $R \subseteq S$, let $w(R)$ denote the sum of the weights of elements in R. The ***weighted median*** of R is any element x such that $w(S_{<x}) \leq w(S)/2$ and $w(S_{>x}) \leq w(S)/2$.

Describe and analyze an algorithm to compute the weighted median of a given weighted set in $O(n)$ time. Your input consists of two unsorted arrays $S[1, \ldots, n]$ and $W[1, \ldots, n]$, where for each index i, the ith element has value $S[i]$ and weight $W[i]$. You may assume that all values are distinct and all weights are positive.

9. (a) Describe an algorithm to determine in $O(n)$ time whether an arbitrary array $A[1, \ldots, n]$ contains more than $n/4$ copies of any value.

   (b) Describe and analyze an algorithm to determine, given an arbitrary array $A[1, \ldots, n]$ and an integer k, whether A contains more than k copies of any value. Express the running time of your algorithm as a function of both n and k.

   Do not use hashing, or radix sort, or any other method that depends on the precise input values, as opposed to their order.

10. Describe an algorithm to compute the median of an array $A[1, \ldots, 5]$ of distinct numbers using at most 6 comparisons. Instead of writing pseudocode, describe your algorithm using a ***decision tree***: A binary tree where each internal node contains a comparison of the form "$A[i] \gtrless A[j]$?" and each leaf contains an index into the array. See Figure 3.

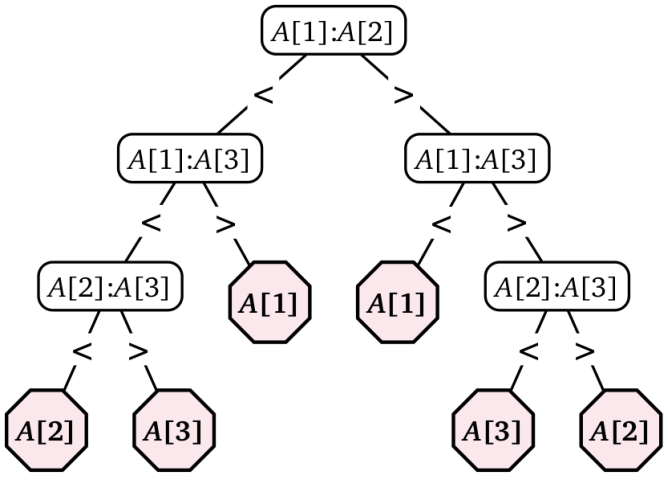

Figure 3: Finding the median of a 3-element array using at most 3 comparisons.

11. You are given an array A with n entries, with each entry holding a distinct number. You are told that the sequence of values $A[1], A[2], \ldots, A[n]$ is ***unimodal***, that is, for some index p between 1 and n, the values in the array entries increase up to position p in A and then decrease the remainder of the way until position n. (So if you were to draw a plot with the array position j on the x-axis and the value of the entry $A[j]$ on the y-axis, the plotted points would rise until x-value p, where they'd achieve their maximum, and then fall from there on.)

    You'd like to find the "peak entry" p without having to read the entire array − in fact, by reading as few entries of A as possible. Describe and analyze an algorithm to find the entry p in $O(\log n)$ time.

12. An array $A[0, \ldots, n-1]$ of n distinct numbers is ***bitonic*** if there are unique indices i and j such that $A[(i-1) \bmod n] < A[i] > A[(i+1) \bmod n]$ and $A[(j-1) \bmod n] > A[j] < A[(j+1) \bmod n]$. In other words, a bitonic sequence either consists of an increasing sequence followed by a decreasing sequence, or can be circularly shifted to become so. For example, see Figure 4. Describe and analyze an algorithm to find the

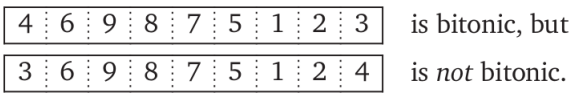

Figure 4: An example of bitonic array.

    smallest element in an n-element bitonic array in $O(\log n)$ time. You may assume that the numbers in the input array are distinct.

13. You're consulting for a small computation-intensive investment company, and they have the following type of problem that they want to solve over and over. A typical instance of the problem is the following. They're doing a simulation in which they look at n consecutive days of a given stock, at some point in the past. Let's number the days $i = 1, 2, \ldots, n$; for each day i, they have a price $p(i)$ per share for the stock on that day. (We'll assume for simplicity that the price was fixed during each day.) Suppose during this time period, they wanted to buy 1000 shares on some day and sell all these shares on some (later) day. They want to know: When should they have bought and when should they have sold in order to have made as much money as possible? (If there was no way to make money during the n days, you should report this instead.)

    For example, suppose $n = 3$, $p(1) = 9$, $p(2) = 1$, and $p(3) = 5$. Then you should return "*buy on* 2*, sell on* 3" (buying on day 2 and selling on day 3 means they would have made Rs. 4 per share, the maximum possible for that period).

    Clearly, there's a simple algorithm that takes time $O(n^2)$: try all possible pairs of buy/sell days and see which makes them the most money. Your investment friends were hoping for something a little better.

    Show how to find the correct numbers i and j in time $O(n \log n)$.

14. You are interested in analyzing some hard-to-obtain data from two separate databases. Each database contains n numerical values-so there are $2n$ values total-and you may assume that no two values are the same. You'd like to determine the median of this set of $2n$ values, which we will define here to be the n-th smallest value.

    However, the only way you can access these values is through queries to the databases. In a single query, you can specify a value k to one of the two databases, and the chosen database will return the k-th smallest value that it contains. Since queries are expensive, you would like to compute the median using as few queries as possible.

    Give an algorithm that finds the median value using at most $O(\log n)$ queries.

15. Suppose you're consulting for a bank that's concerned about fraud detection, and they come to you with the following problem. They have a collection of n bank cards that they've confiscated, suspecting them of being used in fraud. Each bank card is a small plastic object, containing a magnetic stripe with some encrypted data, and it corresponds to a unique account in the bank. Each account can have many bank cards corresponding to it, and we'll say that two bank cards are equivalent if they correspond to the same account.

    It's very difficult to read the account number off a bank card directly, but the bank has a high-tech "equivalence tester" that takes two bank cards and, after performing some computations, determines whether they are equivalent.

    Their question is the following: among the collection of n cards, is there a set of more than $n/2$ of them that are all equivalent to one another? Assume that the only feasible operations you can do with the cards are to pick two of them and plug them in to the equivalence tester. Show how to decide the answer to their question with only $O(n \log n)$ invocations of the equivalence tester.