CS602A: Design and Analysis of Algorithms

Pawan Kumar

February 24, 2018

This document consists of some good practice problems in topics taught so far (till Mid-sems), advice on writing good proofs, and some good references for thinking about algorithms.

1 Some Nice Problems for Practice

Try them out for few minutes before looking at the solutions.

1.1 Divide and conquer

I will often write only the keywords by which the problem is famous. (For example: Merge-sort stands for given an array of objects, you have to sort them and so on.)

1. Counting inversions in an array.
2. Convex hull using Divide and conquer, Quickhull for convex hull.
3. Find closest number in array: Given an array of sorted integers. We need to find the closest value to the given number.
4. Search for a number in sorted matrix.
5. Given a sorted array of n elements containing elements in range from 1 to $n-1$ i.e. one element occurs twice, the task is to find the repeating element in an array.
6. Consider an array of distinct numbers sorted in increasing order. The array has been rotated (anti-clockwise) k number of times. Given such an array, find the value of k. (This is a variant of binary search algorithm)

Most of the solutions you can find easily on web.

1.2 Recurrence Equations

1. $T(n) = 2T(n/2) + n/lgn$
2. $T(n) = 4T(n/2) + nlgn$
3. $T(n) = \sqrt{n}T(\sqrt{n}) + n$
4. $T(n) = T(3n/4) + T(n/4) + n$
5. $T(n) = T(n/5) + T(7n/10) + n$
6. $T(n) = 1/4T(n/4) + 3/4T(3n/4) + 1$

The respective solutions are: $n \log\log n, n^2, n \log\log n, n \log n, n, \log n$.

1.3 Trees and Augmented Trees

1. What order should we insert the elements $\{1, 2, \cdots, 7\}$ into an empty AVL tree so that we dont have to perform any rotations on it? (Answer: $\{4, 2, 6, 1, 3, 5, 7\}$ Multiple solutions possible)
2. Given root of binary search tree and K as input, find K-th smallest element in BST. (Rank is done in the class)
3. Given an array of n elements, find the sum of elements in a range. (May need to build segment tree for this)
4. Given Binary Search Tree. The task is to find sum of all elements smaller than and equal to Kth smallest element.

1.4 Greedy Paradigm

1. Kruskal, Prim (Difference between both; Proof of correctness; Time complexity analysis)
2. Activity Selection Problem: You are given n activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.
3. Shortest Job First Scheduling.
4. Fractional Knapsack problem. (Given in Cormen) (also, Notice the difference with 0-1 Knapsack problem)
5. Given a value V, if we want to make change for V Rs, and we have infinite supply of each of the denominations in Indian currency, i.e., we have infinite supply of $\{1, 2, 5, 10, 20, 50, 100, 500, 1000\}$ valued coins/notes, what is the minimum number of coins and/or notes needed to make the change?
6. Suppose you are given a connected graph G, with edge costs that are all distinct. Prove that G has a unique minimum spanning tree.

2 Some tips for writing a good algorithm

Among the answer-sheets of assignments and quiz-1, we have noticed the following pattern:

1. Even though the proof idea is correct, the presentation is not clear.
2. It is hard to figure out when the algorithm is over and proof of correctness has begun.
3. Proof of correctness is not written properly.
4. Too much low level of description (Machine level details and programming language dependent things are being given, which are unnecessary.)

2.1 Designing the algorithm

Few tips on designing a good algorithm:

1. Read the problem carefully.
2. Start jotting down things on some other rough area. Write your assumptions on one side and the target on another side.
3. Make sure you understand what is being asked and why is it worth asking for.

4. Start drawing pictures, working through small examples often help.
5. Can you notice a pattern? Throw everything on the problem. Hopefully, you will reach somewhere. If not, keep trying.
6. In case, you are unable to figure it out, come back to it later.
7. If you succeed in step 5, Write out the rough draft of the algorithm in the rough sketch.
8. Now, while writing it in the main page (for evaluation), clean out the details. (Check out the next section for how to actually write it.) Often, I have seen that if the problem is easy, students write down the solution without working out the details and the written solution lacks flow and ideas get tangled. It becomes hard to comprehend for a reader.

2.2 Writing the algorithm

1. In the beginning, write out which paradigm you are using for solving the problem. (For example: greedy, dynamic programming etc. It makes the reader's job easy. It will give an idea that what is to be expected.)
2. Write the algorithm on the level which is expected. (For example: Don't give details on machine level and how to run it, unless needed. It won't be needed in this course)
3. Start and end points of the algorithm should be clear to see.
4. The sentences of the proof should flow logically. If you are declaring a new variable or set, don't use the existing variables.
5. Feel free to use any data structure you want. As long as you are correct, it is okay. If you are designing a data structure by yourself (not covered in class), clearly state it and prove every specific claim you make about that data structure.
6. Feel free to use any result done in the class without proving them. (Note that this doesn't include assignment problems.)
7. In the end, do a proof-reading. Read it out loud in head. Do a clean-up if needed.

2.3 Writing the proof of correctness

1. Again, in the beginning write out the paradigm you are using for proof of correctness. (For example: Induction, contradiction, contrapositive etc.)
2. Sometimes, you may have to prove a few intermediate results to prove the correctness of the algorithm. Each such intermediate result is a lemma.
3. Prove each lemmas separately. Clearly state the lemma and prove it. (Sometimes, a lemma may require another lemma.)
4. Write in complete sentences. Don't use symbols like $\because$ or $\therefore$ while writing proofs. They are okay to be used in rough drafts.
5. The claims should be specific and unambiguous. Ambiguity often means you are not 100% clear about what to write.
6. Don't write clearly ...., obviously ...., "it is easy to see" etc.
7. Mark with some square box symbol ■ or write QED if the proof is over. Sometimes, it is hard for a reader to figure out if you are done with the proof or not. (This applies to lemmas too)

Some of the well-known proof methods are: direct proof, by contrapositive, by contradiction, by exhaustive case analysis, mathematical induction, exchange argument (in case of greedy algorithm, it is very useful; try it out for Kruskal algorithm correctness), reducing to a well-known problem whose proof of correctness you can show etc.

2.4 Writing the complexity analysis

I have hardly noticed any problem in this area(At least with students in the $\cdots 23$ to $\cdots 48$ roll number zone). So, leaving this one out.

3 Some good references

- Reference books on CS602A webpage. (Cormen book, Dasgupta book, Tardos book) (If you don't have them, please let me know.)
- For Interval, segment, Range, Priority Search trees Here is a good source.
- Another good set of slides on Interval trees: Slides
- Proving the optimality of Greedy algorithm Exchange Argument
- A good book on writing proofs is *"How to read and do proofs"* by *Daniel Solow.*
- First chapter of *"Discrete Mathematics*" book by *Kenneth and Rozen.*
- Revision of CS601: (Formal Logic part only)