

Analysis of Algorithms
CS 376, Fall 2020

Problem Set 5

Due **BY 11:59PM, Maine time** Thursday, November 5

Please submit this assignment in a file named `CS376_PS5-<YourName>.pdf`.

- From CLRS, please read Section 4.2, Section 7.1, and Chapter 15, pages 359–360, 378–396.
- For this assignment and future CS376 assignments, there may be deductions for excessively inelegant solutions (e.g., algos, proofs) to exercises. Answers do not need to be optimally elegant for full credit, but when an excessively inelegant answer misses some central concepts that would result in a cleaner solution, there is room for improvement to the answer, which may entail a deduction. Please feel free to ask me questions about this and how it might apply to your specific HW solutions!
- *A general note for CS376:* When writing up your homework, please write neatly and **explain your answers clearly**, giving all details needed to make your answers easy to understand. Clear communication *is* the point, on every assignment.

In general in CS376, unless explicitly specified otherwise, answers should be accompanied by explanations; as part of this, in general, when you present an algorithm, give an accompanying correctness argument (i.e., proof). Answers without explanations may not receive full credit. Please feel free to ask me any questions about explanations that might come up!

Exercises

1. Trace the *full operation* of Quicksort (as given in CLRS, Section 7.1) on array $A = \langle 13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11 \rangle$. Show the array and the (*first*, *last*, *pivot*) values for every recursive call of Quicksort—that is, the p and r values for every recursive call, and the resulting pivot value—and show the results of every call of **Partition** performed during the procedure. That is, show the array orderings and resulting sub-arrays at every level in the recursion tree.

The purpose of this exercise is to give you experience with how Quicksort works in detail. Because of that, this exercise is not intended to be an especially small amount of work, or have an especially slick answer—it’s intended to take you through the detailed process of Quicksort once in your Colby CS education! For this one exercise, you do not need to provide text explanations of your work; showing the recursive calls and results for every step in the recursion as described will suffice for this exercise.

NOTE: If you’re not fully comfortable with how **Partition** works, please feel free to work through exercise CLRS 7.1-1 (page 174), and I’ll be happy to go over it with you!

2. Find the number of different ways to climb an n -stair staircase if each step is either one or two stairs. (For example, a 3-stair staircase can be climbed three ways: 1-1-1, 1-2, or 2-1.)

As always, explain your work—here, prove your answer is correct.

3. In chess, a rook can move horizontally or vertically to any square in the same row or in the same column of a chessboard. Find the *number of shortest paths* by which a rook can move *from the bottom-left corner of a chessboard to the top-right corner*. (The length of a path is measured by the number of squares it passes through, including the first and the last squares.)

Solve the problem by a dynamic programming method. That is, come up with a relevant recurrence—i.e., a recursive definition of the relevant values for a solution, as we’ve seen in the examples from class—and using dynamic programming techniques, calculate the solution. As part of your answer, please give an 8×8 table (each element in the table represents the corresponding square on the chessboard), where each entry in the table should be the number of paths from the bottom-left corner to that square on the chessboard.

Note that you do not need to provide a shortest path from one corner to the other, just the number of shortest such paths. As always, explain your answers—here, in particular, please be sure to include an explanation of the correctness of your recurrence.

4. We are given a $4 \times n$ checkerboard with 4 rows and n columns and a supply of pebbles (at least $2n$). Each square on the board has an integer written on it. We want to place the pebbles on the board so as to maximize the sum of the integers that are covered by pebbles under the constraint that no two pebbles are on adjacent squares. Adjacency is defined by sharing an edge, i.e., each square has 2 adjacent squares if it is a corner, 3 adjacent squares if it is on the side, and 4 adjacent squares otherwise.

Describe an algorithm for finding the optimal placement of pebbles. As always, explain your algorithm design and prove it correct. Also, analyze the running time of your algorithm, giving the tightest asymptotic complexity bound you can.

NOTE: More efficient algorithms, as measured by asymptotic complexity classes, will receive more credit on this exercise. Also, recall that excessively inelegant algorithms may not receive full credit. If you have questions about whether your proposed solution is excessively inefficient or inelegant, please ask your Prof.!

Optional Exercises

The following exercises **are not to be submitted!** They are purely optional exercises, for those looking to build their algorithm design and analysis skills a little more. If you do work on these exercises, though, I’ll be happy to talk with you about them, just let me know!

- CLRS 9.3-8 (page 223).
- CLRS 9.3-9 (page 223).

Although these exercises are in the section of CLRS with the *amazing* result that we can find a median in worst-case linear time (wow!), and it is important that you know that result (not the algorithm, just the fact that it can be done), the details of that algorithm are not essential for these exercises.