

Analysis of Algorithms
CS 376, Spring 2023

Problem Set 4

Due **AT THE BEGINNING OF CLASS** Monday, April 24

- From CLRS, please read Chapter 15 (Sections 15.1 and 15.5 are skippable).
- Please follow the usual CS376 file naming conventions: submit this assignment to your private `filer` folder in a file named `CS376_PS4_<YourName>.pdf`.
- 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. Consider a rectangle whose side lengths are two consecutive Fibonacci numbers. (Of course, neither of them is 0.) Such a rectangle could be, for example, 3 by 5, or 8 by 13, or 21 by 34, etc.
 - (a) Give a recursive algorithm to dissect such a rectangle into squares such that no more than two of the resulting squares are the same size. (For example, if you had two 3 by 3 squares, you could have at most one 4 by 4 square.) Here's a specification for your algorithm:

```
// Input: Two consecutive Fibonacci numbers f0, f1,  
         representing an f0 by f1 rectangle, such that f0 <= f1.  
         (Neither f0 nor f1 will be 0.)  
// Output: A list of integers representing side lengths of squares,  
           such that the input rectangle can be dissected into squares  
           of those sizes. No more than two of the squares can be the  
           same size.
```

Please be sure to give an English description of the algorithm along with pseudocode, explaining the main points of its design, and a concise inductive argument for its correctness (i.e., say what makes the base case correct, what makes the recursive cases correct, and how you know the algorithm terminates).

- (b) What is the time complexity of your algorithm in part 1a? Give a recurrence for your algorithm, and solve it to get a Θ complexity bound. As always, fully explain your answer, show your work in solving the recurrence, and be sure to explicitly say what each variable in your complexity class stands for (e.g., if you're presenting a $\Theta(n^3)$ algorithm, be sure to say what n refers to).
2. **A Rook-y Move!** 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. Along with giving the recurrence and a correctness argument / explanation of the recurrence, show your work by giving 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; therefore, the answer to this exercise will be the number in the top-right element of your table.

Note that you do not need to provide a shortest path from one corner to the other, just the number of shortest such paths. Please be sure to include an explanation of the correctness of your recurrences (a well-worded paragraph could suffice); you do not need to explain how you used the generated table from the recurrence.

3. 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: One good approach to this exercise could be to approach it as a dynamic programming exercise—look to design a recursive solution, and then look to make it more efficient. 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.!