Topics:
• proofs;
• induction;
• summations;
• \( O, \Omega, \Theta; \)
• complexity of an algorithm;
• stable sorting, the Gale-Shapley algorithm;
• Divide and conquer techniques and algorithms;
• Recurrences, recursion trees, the Master Theorem
• Dynamic programming (recursive and iterative);
• Optimal substructure and the Bellman equation;
• The recursion tree and the subproblem graph;
• Complexity of a dynamic algorithm;

What you should do:
• read the book;
• read and understand all the homework solutions;
• read all the homework problems and make sure you can do them;
• think about the problems below.

1. (induction) Consider the theorem below.

**Theorem 0.1.** If \( a \in \mathbb{R}_{>0} \) and \( n \in \mathbb{Z}_{>0} \) then \( a^{n-1} = 1 \).

**Proof.** We shall proceed by induction on \( n \). If \( n = 1 \), then we have

\[
a^{n-1} = a^1 = a^0 = 1,
\]

establishing the base case. For the inductive step, let us assume that the theorem is true for \( 1, 2, \ldots, n \), and we shall prove that it holds for \( n + 1 \). We have

\[
a^{(n+1)-1} = a^n = \frac{a^{n-1} a^{n-1}}{a^{(n-1)-1}} = \frac{(1)(1)}{1} = 1,
\]

completing the induction and thus proving the theorem. \( \square \)

The theorem is clearly false, so there must be an error in the proof. Find it.

2. (induction) Consider the theorem below.
Theorem 0.2. If \( a \in \mathbb{R}_{>0} \) and \( n \in \mathbb{Z}_{>0} \) then \( a^{n-1} = 1 \).

Proof. We shall proceed by induction on \( n \). If \( n = 1 \), then we have
\[
a^{n-1} = a^{1-1} = a^0 = 1,
\]
establishing the base case. For the inductive step, let us assume that the theorem is true for \( 1, 2, \ldots, n \), and we shall prove that it holds for \( n + 1 \). We have
\[
a^{(n+1)-1} = a^n = \frac{a^{n-1}a^{n-1}}{a^{(n-1)-1}} = \frac{(1)(1)}{1} = 1,
\]
completing the induction and thus proving the theorem. \( \square \)

The theorem is clearly false, so there must be an error in the proof. Find it.

3. (induction) Prove or disprove that
\[
\sum_{k=1}^{n} k^3 = \left( \sum_{k=1}^{n} k \right)^2.
\]

4. (induction) The Fibonacci sequence is given by the recurrence
\[
F(0) = 0, \quad F(1) = 1, \quad F(n) = F(n-1) + F(n-2) \quad \text{for} \quad n \geq 2.
\]
Let \( \varphi = (1/2)(1 + \sqrt{5}) \). Prove that
\[
F(n) = \frac{1}{\sqrt{5}} \left( \varphi^n - (1 - \varphi)^n \right).
\]
[NB: \( \varphi \) is commonly called the Golden Ratio.]

5. (complexity, divide and conquer) Prove that binary tree search of full balanced tree with \( n \) nodes is in \( \Theta(\log n) \).

6. (complexity, divide and conquer) What is the \( \Theta \)-complexity of searching a tree \( T \) with \( n \) nodes where each internal node has exactly 3 children.

7. (complexity, divide and conquer) What is the \( \Theta \)-complexity of searching a tree \( T \) with \( n \) nodes where each internal node has at least 2 children.

8. (complexity) Prove that the jar testing algorithm from homework 2 has \( O(\sqrt{n}) \) for \( k \) jars, where \( n \) is the number of rungs on the ladder.

9. (complexity) Prove or disprove: if \( f_1(n) \prec g_1(n) \) and \( f_2(n) \prec g_2(n) \) then \( f_1(n) + f_2(n) \prec g_1(n) + g_2(n) \).

10. (complexity) Prove or disprove: \( (2^n)^2 \in O(2^{n^2}) \).

11. (complexity) Prove or disprove: \( 2^{n^2} \in O(n^n) \).

12. (stable sorting) Produce a stable sorting for some given problem.

13. (stable sorting) Modify stable sorting so that there is a single set \( A \) of even size, and each element of \( A \) has a preference list. Must there always exist a stable sorting? If so, then give an algorithm to find it. If not, then give a counterexample.
14. (recurrences) Prove or disprove: if \(c(n-1)\log(n-1) \leq T(n) \leq cn\log(n)\) then \(T(n) \in \Theta(n\log n)\).

15. (divide and conquer) State the merge sort algorithm, find the recurrence for its runtime, and find the \(\Theta\)-complexity of it.

16. (divide and conquer) Repeat the above problem, but break the list into 4 equal parts.

17. (divide and conquer) In the merge sort algorithm, what if the subarrays are passed by copying (\(\Theta(n)\)), and not by pointer (\(\Theta(1)\))? How does the runtime change?

18. (recurrences, dynamic programming) Solve the tower of Hanoi problem in the fewest number of moves, find the recurrence for its runtime, and solve the recurrence.

19. (recurrences, dynamic programming) Repeat the above problem, but where instead of having \(n\) disks of all different sizes, we have \(2n\) disks with 2 of each size.

20. (recurrences, dynamic programming) Repeat the tower of Hanoi problem, but where instead of having 3 pegs we have 4 pegs. Your solution should be optimal.

21. (Master Theorem, recurrences) Give asymptotic upper and lower bounds for \(T(n)\) in each of the following recurrences. Assume that \(T(n)\) is constant for sufficiently small \(n\). Make your bounds as tight as possible, and justify your answers.

   (i) \(T(n) = 4T(n/3) + n\log(n)\).

   (ii) \(T(n) = 3T(n/3) + n/\log(n)\).

   (iii) \(T(n) = 4T(n/2) + n^2\sqrt{n}\).

   (iv) \(T(n) = 3T(n/3 - 2) + n/2\).

   (v) \(T(n) = 2T(n/2) + n/\log(n)\).

   (vi) \(T(n) = T(n/2) + T(n/4) + T(n/8) + n\).

   (vii) \(T(n) = T(n - 1) + 1/n\).

   (viii) \(T(n) = T(n - 1) + \log(n)\).

   (ix) \(T(n) = T(n - 2) + 1/\log(n)\).

   (x) \(T(n) = \sqrt{n}T(\sqrt{n}) + n\).

22. (dynamic programming) Given some dynamic programming problem, produce the Bellman equation for it.

23. (dynamic programming) From above, also produce a recursive algorithm to solve it.

24. From above, implement your recursive algorithm from the previous question iteratively.

25. (dynamic programming) From above, Draw the recursion tree for the above problem.
26. (dynamic programming) From above, Draw the subproblem graph for the above problem.

27. (dynamic programming) Given a recursive dynamic programming algorithm, convert it to an iterative algorithm.

28. (dynamic programming) Given a non-memoized dynamic programming algorithm, calculate the non-memoized $\Theta$-complexity (count the number of nodes in the recursion tree).

29. (dynamic programming) Given a non-memoized dynamic programming algorithm, calculate the $\Theta$-complexity of the memoized version (count the number of arrows in the subproblem graph).

30. (dynamic programming) Given a price table and rod length, solve the rod-cutting problem using the dynamic programming approach (don’t brute force it).

31. (dynamic programming) Given intervals and their weights, solve the weighted interval scheduling problem using the dynamic programming approach.