|  |  |
| --- | --- |
|  | **Algorithm Design and Analysis**  **Assignment 03: Divide-and-Conquer, Dynamic Programming** |

**Total Points:** 40

**Assigned:** Thursday, 25 March 2021

**Due:** Thursday, 1 April 2021, 11:59pm.

**Assignment Type:** Individual

**Submit:** PDF for written work, Source files for code

**Instructions**

* Submit individual solutions to this assignment. Remember that you may brainstorm with your peers (e.g. your study group), but everyone should write/code up their solution independently. Remember also that you may not search for solutions to assignment questions online, nor may you refer to such solutions if you happen to come across them.
* As mentioned in the course syllabus, remember to list anyone you brainstorm or discuss with on the assignment. Refer to other course policies in the syllabus.
* All algorithms must be written in a well-structured way, like the examples in the problems and in the textbook.

**Preparation**

Study Chapters 5 & 8 of the textbook by Levitin.

You are encouraged to study with your study group. Discuss key ideas as well as things you do not understand, and post questions on Piazza or bring them to office hours.

**PART A: Basic Practice with Key Algorithms**

**Problem 1 [4 points]:** *Divide & Conquer: Quicksort*

Apply quicksort to sort the list **S,O,R,T,I,N,G** in alphabetical order. Draw the tree of the recursive calls made.

**Problem 2 [4 points]:** *Divide & Conquer: Large integer multiplication*

Compute 1201∗2430 by applying the divide-and-conquer algorithm for large integer multiplication outlined in the text/slides.

**Problem 3 [10 points]**: *Dynamic Programming: Knapsack Problem*

1. [4 points] Apply the bottom-up dynamic programming algorithm to the following instance of the knapsack problem:

|  |  |  |
| --- | --- | --- |
| Item | Weight | Value |
| 1 | 3 | ¢25 |
| 2 | 2 | ¢20 |
| 3 | 1 | ¢15 |
| 4 | 4 | ¢40 |
| 5 | 5 | ¢50 |

capacity W = 6.

1. [1 point] How many different optimal subsets does the instance of part (a) have?
2. [1 point] In general, how can we use the table generated by the dynamic programming algorithm to tell whether there is more than one optimal subset for the knapsack problem’s instance?
3. [4 points] Instead of computing values in a bottom-up way, suppose we apply the memory function method to the instance of the knapsack problem given in part (a). Indicate the entries of the dynamic programming table that are (i) never computed by the memory function method, (ii) retrieved without a recomputation.

*Pick ONE of either problem 4 or problem 5. You may do the other as extra credit if desired.*

**Problem 4 [4 points]**: *Dynamic Programming: Warshall’s algorithm*

Apply Warshall’s algorithm to find the transitive closure of the digraph defined by the following adjacency matrix:



**Problem 5 [4 points]:** *Dynamic Programming: Floyd’s algorithm*

Solve the all-pairs shortest-path problem for the digraph with the weight

matrix:



**PART B:** **Applying Algorithm Design Paradigms**

*Pick ONE of either problem 6 or problem 7. You may do the other as extra credit if desired.*

**Problem 6 [8 points]:** *Divide & Conquer: Computing Levels in a Binary Search Tree*

Design a divide-and-conquer algorithm for computing the number of levels in a binary tree. (In particular, the algorithm must return 0 and 1 for the empty and single-node trees, respectively.) What is the time efficiency class of your algorithm?

**Problem 7 [8 points]**: *Dynamic Programming: Minimum-sum descent*

Some positive integers are arranged in an equilateral triangle with n numbers in its base like the one shown in the figure below for n = 4. The problem is to find the smallest sum in a descent from the triangle apex to its base through a sequence of adjacent numbers (shown in the figure by the circles). Design a dynamic programming algorithm for this problem and indicate its time efficiency.



**PART C: Programming Exercises**

*For programming tasks, you may use either Python or Java.*

**Problem 8 [10 points]:** *Implementing divide-and-conquer closest pair*

Implement the divide & conquer algorithm for computing the closest pair problem. Your method/function should take two parameters: an array of x coordinates and an array of y coordinates representing the points. It should return the pair of points that are closest to each other (In Java, this could, for example, be returned as an array containing two Point objects. In Python, this could be a list (of length 2) of tuples).

***Additional Extra Credit Problem*** (Optional)

Implement a solution to the convex hull problem.