CS 210 Advanced Data Structures & Algorithms
Fall 2020

Homework Assignment #1

(Due: 9/29/20)

[100 points]

Part 1: Written Exercises

1. **[15 points]** ***(Algorithms: Efficiency & Analysis)*** **Chapter 1: Exercise #4, #25, #30 and #34 & Appendix B: #24** (Remove *n* in (c) *n T(n)*= …)
2. **[10 points]** ***(Divide-and-Conquer)*** **Chapter 2: Exercise #9 and #20**. (Draw the tree like Figure 2.2 and Figure 2.3 in Chapter 2.)
   - **a.** What is the **worst-case** time complexity of **MergeSort** using $O(n^2)$ time & O(1) space merge? Explain.
   - **b.** What is the **average-case** time complexity of **QuickSort**? Explain.
3. **[10 points]** ***(Dynamic Programming)*** **Chapter 3: Exercise #5, #6 and #10** (No need to show all the actions step by step. You may show some intermediate steps and the final answer.)
4. **[15 points]** ***(Advanced Linked Lists & Graphs)***
   - **a.** Draw the **perfect skip list** that results when you insert items with the keys 19, 6, 26, 9, 2, 12, 25, 7, 21 and 17 in that order into an initially empty perfect skip list.
   - **b.** Draw the **randomized skip list** that results when you insert items with the keys 19, 6, 26, 9, 2, 12, 25, 7, 21 and 17 in that order into an initially empty randomized skip list.
   - **c.** Compare the **binary search tree** with the **perfect skip list** and with the **randomized skip list** (regarding the worst-case and average case time complexity of the *search*, *insert* & *delete* operations).
   - **d.** Compare **depth-first-search** with **breadth-first-search** of graphs. List four types of **shortest path problems** on an edge-weighted directed graph.

Part 2: Programming Exercises*

1. **[20 points]** ***(Dynamic Programming)*** **Dynamic Programming-based Floyd-Warshall Algorithm for the All-Pairs Shortest Path Problem** (**Chapter 3: Exercise #8**)

   You should output the distance array **D**, the path array **P** and the **shortest paths**. Test your implementation using (1) the test case given in class and (2) your own test cases including digraphs with *non-negative* edges only and digraphs with some *negative* edges.

2. **[30 points]** ***(Advanced Linked Lists)*** **A Self-Organizing Linked List with Move-To-Front & Transpose**

   Design and implement the Self-Organizing (Unsorted) Singly-Linked List with Move-To-Front & Transpose ADT.

   You should include ***search*** *(linear search using no self-organizing heuristic)****, search_mtf*** *(linear search using the* ***move-to-front*** *self-organizing heuristic),* ***search_t (****linear search using the* ***transpose*** *self-organizing heuristic),* ***insert*** and ***showSOLL***.

   The ***insert*** operation scans the entire list to verify that the item is not already present and then inserts the item *at the end of the list*.

   The operation ***showSOLL*** prints the values in the self-organizing linked list *in a linked list form*.

   Test your implementation and compare ***search***, ***search_mtf*** and ***search_t*** via *showSOLL* using (1) the test case given in class and (2) your own test cases via *showSOLL*.

*** NOTE:**

- Programming in C++ using Visual Studio
- **Pair Programming**
- Deliverables:
  - ✓ **The screenshots of test results for all test cases.**
  - ✓ **The Visual Studio project folder.**

*** End of Homework Assignment #1 ***