CECS 528, LO Assessment 2, Fall 2026, Dr. Ebert

IMPORTANT: For each problem (LO1,LO2,GK,Advanced, All LO Makeups) write your solution using a **SINGLE** and **SEPARATE SHEET OF PAPER (FRONT AND BACK)**. Write your **FIRST NAME**, **ALPHABETIZING NAME** and **PROBLEM CODE** on each sheet.

25 Points Each

LO3 a. Recall that an instance of the `Task Selection` problem is a finite set T of tasks, where each task t has a start time $s(t)$ and finish time $f(t)$ that indicate the interval for which the task should be completed by a single processor. The goal is to find a subset T_{opt} of T of maximum size whose tasks are pairwise non-overlapping, meaning that no two tasks in T_{opt} share a common time in which both are being executed. Note: a task with respective start and finish times 2 and 4 does *not* overlap with a task with respective start and finish times 4 and 7, but *does* overlap with a task with respective start and finish times 3 and 6.

(a) For the algorithm described in the lecture notes, state the greedy choice that is being made in each round of the algorithm. (7 pts)
Solution. Choose the task that starts at or after the last scheduled task and, of all such tasks, finishes the earliest.

(b) Apply the algorithm to the following set of tasks, where each triple in set T represents the id, start time, and finish time.

$$T = \{(1, 90, 120), (2, 110, 170), (3, 100, 120), (4, 20, 140), (5, 20, 70), (6, 40, 90), (7, 180, 190),$$

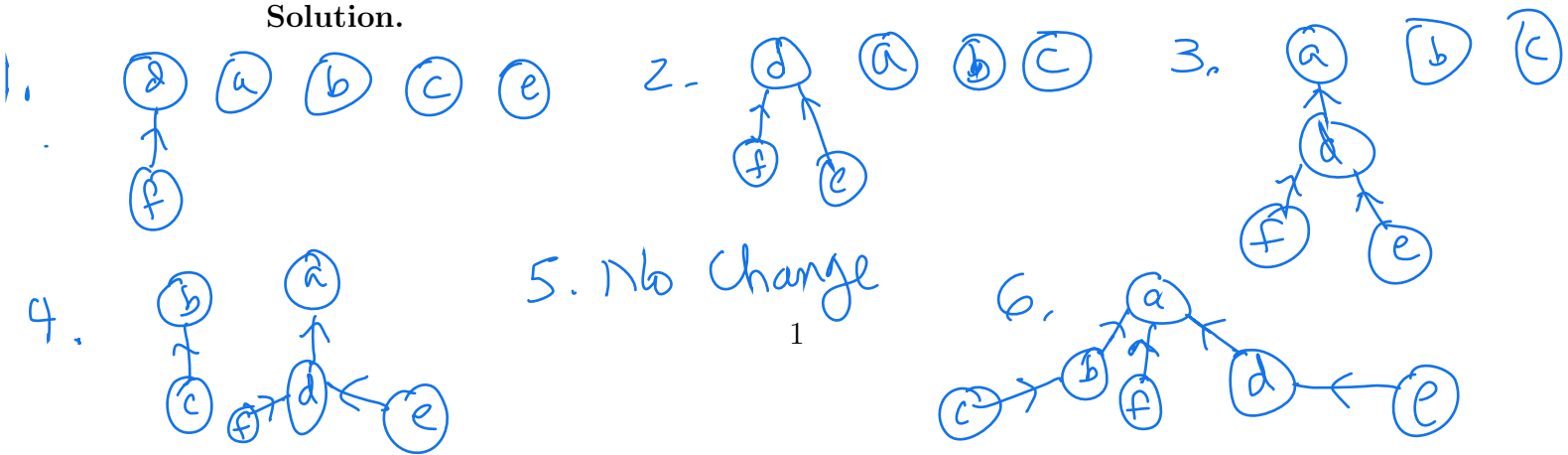
$$(8, 50, 170), (9, 60, 170), (10, 90, 200), (11, 20, 130), (13, 60, 150), (14, 30, 50), (15, 160, 170)\}.$$

Indicate the task that is selected in each round. (5 pts)
Solution. The optimal set of tasks is $\{14, 1, 15, 7\}$.

b. For the weighted graph with edges

$$(a, d, 5), (a, f, 3), (b, c, 4), (c, f, 6), (d, f, 1), (e, f, 2),$$

Show how the forest of disjoint-set data structure trees changes when processing each edge in the Kruskal's sorted list of edges. When merging two trees, use the convention that the root of the merged tree should be the one having *lower* alphabetical order. For example, if two trees, one with root a, the other with root b, are to be merged, then the merged tree should have root a. Redraw the forest each time an edge is processed. (13 pts)
Solution.



LO4

Solve the following problems.

a. The dynamic-programming algorithm that solves the `Longest Common Subsequence (LCS)` optimization problem defines a recurrence for the function $\text{lcs}(i, j)$. In words, what does $\text{lcs}(i, j)$ equal? Hint: do *not* write the recurrence (see Part b). (5 pts)

Solution. $\text{lcs}(i, j)$ gives the length of the longest subsequence (i.e. subword) that is common to the words $u[1 : i]$ and $v[1 : j]$.

b. Provide the dynamic-programming recurrence for $\text{lcs}(i, j)$. (10 pts)

Solution.

$$\text{lcs}(i,j) = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ \text{lcs}(i-1,j-1)+1 & \text{if } u_i = v_j \\ \max(\text{lcs}(i-1,j), \text{lcs}(i,j-1)) & \text{otherwise} \end{cases}$$

c. Apply the recurrence from Part b to the words $u =$ abaaba and $v =$ babaaa. Show the matrix of subproblem solutions and use it to provide an optimal solution. (10 pts)

Solution.

lcs(i,j)

i \ j		λ	b	a	b	a	a	a
	λ	0	0	0	0	0	0	0
a		0	↑0	↖1	←1	↖1	↖1	↖1
b		0	↖1	←1	↖2	←2	←2	←2
a		0	↑1	↖2	←2	↖3	↖3	↖3
a		0	↑1	↖2	←2	↖3	↖4	↖4
b		0	↖1	↑2	↖3	←3	↑4	↑4
a		0	↑1	↖2	↑3	↖4	↖4	↖5

abaaa

GK a. What does it mean for an algorithm to be a greedy heuristic? Describe the greedy heuristic that was presented in the Greedy Algorithms exercises for solving the `Optimal Binary Search Tree` problem. (13 pts)

Solution. A greedy heuristic is a greedy algorithm that does not always produce an optimal solution, such as the one for solving the `Optimal Binary Search Tree` problem, in which, when choosing a root node among a set of choices, the node having highest weight is selected.

b. Provide one important way for which divide-and-conquer and dynamic-programming algorithms are similar, and one important way for which they are different. Please use complete sentences.

Solution. Both algorithmic strategies reduce a problem instance to solving one or more smaller instances of the same problem. They differ in how they are implemented, with divide-and-conquer algorithms being recursively implemented and dynamic-programming algorithms being non-recursively implemented.

(12 pts)

ADV a. Recall the `Matrix Chain Multiplication (MCM)` optimization problem and consider the following greedy heuristic for selecting the k value, $i \leq k < j$ in order to determine how to split the matrix sequence into two groups. Choose k such that $p_{i-1}p_kp_j$ is minimized. Provide an instance of MCM for which this heuristic fails. (10 pts)

Solution. Consider the dimensions 1,2,3,4 for matrices A_1, A_2, A_3. Then according to the heuristic, the split should be $(A_1(A_2A_3))$. Then

$$mc(A_1(A_2A_3)) = 0 + (2 \times 3 \times 4) + (1 \times 2 \times 4) = 32.$$

On the other hand,

$$mc((A_1A_2)A_3) = 0 + (1 \times 2 \times 3) + (1 \times 3 \times 4) = 18$$

which is better than the heuristic solution.

b. Provide a *recursive* implementation of the `0-1 Knapsack` function

```
void print_items(int[][] p, int i, int c, int[] w)
```

which has the side effect of printing all items that contribute to the optimal knapack, assuming items $1, \ldots, i$ and a knapsack capacity equal to c, where input matrix p represents the already-computed 0-1 knapsack profit function, and array w provides the weight of each item. You may assume the existence of a `print` function which takes as input an integer i and has the side effect of printing i to output (with surrounding whitespace). Note that the initial call to `print_items` would be `print_items`(p, n, M, w), where n is the orginal number of items and M is the original capacity. Hint: remember to include all base cases. (15 pts)

Solution.

```
void print_items(int[][] p, int i, int c, int[] w)

    //Base Case
    if(i == 0 or c == 0), then return. //no items or fulll knapsack
```

```
    //Recursive Case 1
    If(p(i,c) == p(i-1,c), then
        print_items(p, i-1, c, w). //Item i not used

    Else
        //Recursive Case 2
        print_items(p,i-1,c - w[i]). //item i used
        print(i).
}
```

LO MakeUps: Pass/No Pass

LO1 a. Use the Master Theorem to determine the growth of $T(n)$ if it satisfies the recurrence $T(n) = 8T(n/2) + n^3 \log n$. Defend your answer.

b. Use the Substitution method to establish that, if $T(n) = 2T(n/2) + n \log n$, then $T(n) = \Omega(n \log^2 n)$.

LO2 a. Consider Karatsuba's algorithm for multiplying two n-bit binary numbers x and y, where we assume that n is even. Let x_L and x_R be the leftmost $n/2$ and rightmost $n/2$ bits of x respectively. Define y_L and y_R similarly. Let P_1 be the result of calling `multiply` on inputs x_L and y_L, P_2 be the result of calling `multiply` on inputs x_R and y_R, and P_3 the result of calling `multiply` on inputs $x_L + x_R$ and $y_L + y_R$. Then return the value $P_1 \times 2^n + (P_3 - P_1 - P_2) \times 2^{n/2} + P_2$. Demonstrate Karatsuba's algorithm on $x = 36$ and $y = 43$, where both number are assumed to require six bits. Verify through calculation that the algorithm does in fact compute xy. Limit your demonstration to the root level of recursion. In other words, do not go deeper than level 0.

b. What is the divide-and-conquer recurrence for the divide-and-conquer algorithm that solves the `Minimum Positive Subsequence Sum` problem? Defend your answer by desribing the algorithm using sufficient detail.