Questions and Exercises to work out and turn in:

Grading Guidelines:

* A right answer will get full credit when:

1. It is right (worth 25%)
2. It is right **AND** neatly presented making it easy and pleasant to read. (worth an **extra** 15%)
3. There is an **obvious and clear link** between 1) the information provided in the exercise and in class and 2) the final answer. A clear link is built by properly writing, justifying, and documenting an answer (worth an **extra** 60%).
4. Calculation mistakes will be minimally penalized (2 to 5% of full credit) while errors on units will be more heavily penalized.

You are welcome/encouraged to discuss exercises with other students or the instructor. But, ultimately, **personal** writing is expected.

* USE THIS FILE AS THE STARTING DOCUMENT YOU WILL TURN IN. **DO NOT DELETE ANYTHING FROM THIS FILE:** JUST **INSERT** YOUR ANSWERS.
* IF USING HAND WRITING (STRONGLY DISCOURAGED), **USE THIS FILE** BY CREATING SUFFICIENT SPACE AND WRITE IN YOUR ANSWERS.
* FAILING TO FOLLOW TURN IN DIRECTIONS /GUIDELINES WILL COST **A 30% PENALTY.**

Objectives of this assignment:

* to use and manipulate the concepts presented in this module
* to propose and write algorithms in pseudocode
* to analyze the time complexity of algorithms
* to analyze the space complexity of algorithms
* to learn autonomously new concepts

What you need to do:

Answer the questions and/or solve the exercises described below.

Exercise 1 (25 points)

Suppose that instead of always selecting the first activity to finish, we instead select the last activity to start that is compatible with all previously selected activities.

1. (25 points) Describe how this approach is a greedy algorithm. Just using common sense, explain how this heuristic “makes sense”

This approach is a greedy algorithm because it looks at the locally optimal solution to a subproblem, and then only 1 subproblem remains. In this case, the problem is which activity to choose, and the heuristic is to choose the activity with the latest starting time. This heuristic means that the algorithm does not have to look at all sub-problems, but rather only look at the locally optimal solution, and then create another subproblem including only activities that end before the chosen activity starts.

This heuristic makes sense due to its similarity to the already-proven heuristic of choosing the first-finishing activity. The overall idea is to maximize the available time left for other activities, so the latest start time is chosen because that means all times from 0 up to the chosen start time is free. Since I already said the chosen start time is the latest among all available start times, the total time available for other activities is maximized. Then, of course, the algorithm continues to make the same greedy choice over and over until the set of compatible activities is empty. In this way, at each step the time for other activities is at a maximum value, so the result is the maximum number of activities that can take place within a certain time frame.

1. (10 **bonus** points) **Prove** that it yields an optimal solution. Insure to follow the same steps used in the lecture to show that the greedy approach to select the earliest finish time activity that is compatible does deliver an optimal solution. Define well your notations.

Sk is a nonempty subproblem, am is an activity in Sk with the latest start time. Ak is the maximum size subset of mutually compatible activities in Sk, and aj is the activity in Ak with the latest start time. If aj = am, then it is already proven that am­ is a part of some maximum size subset. If aj does not equal am, then let a new set A’k = Ak – {aj} U {am}. This new set A’k has disjoint activities, since all of the activities in Ak are disjoint, and the new activity am has a starting time >= the starting time of aj. The cardinality of Ak and A’k are the same, therefore A’k is a maximum size subset of mutually compatible activities in Sk, and it includes am.

Exercise 2 (20 points)

This exercise is about the greedy approach to the activity-selection problem. Kevin claims that selecting the shortest activity can lead to a larger set of mutually compatible activities than selecting the earliest finish activity. Is Kevin right or wrong? **Prove** your answer.

Kevin is wrong, this is self-evident as it has already been proven that selecting the earliest finish activity provides a set that contains the highest number of mutually compatible activities. It is not possible to have a set with a higher number of mutually compatible activities.

As a short more formal proof, I will use proof by contradiction. Suppose Kevin is right, and selecting the shortest activity each time results in a set with more activities than picking the activity with the earliest finishing time. Then, if Ak is the set that Kevin creates, and Am is a set of the maximum number of possible mutually compatible activities, |Ak| <= |Am|. Now, suppose that Ae is the set created by picking the earliest finishing time activity. Kevin is claiming that |Ak| > |Ae|. On page 418 of the textbook, theorem 16.1 indirectly states that picking the earliest finishing time activity at each step will result in |Ae| = |Am|. Since |Ak| <= |Am| = |Ae|, |Ak| <= |Ae|, which directly contradicts the earlier assumption that |Ak| > |Ae|.

Exercise 3 (20 points)

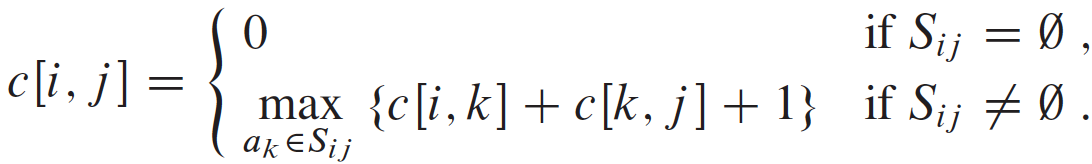
This exercise is about the greedy approach to the activity-selection problem. James claims that in some cases the elimination of the first activity to finish from the set of activities can lead to a larger set of mutually compatible activities. Is James right or wrong? **Prove** your answer.

James is wrong, eliminating the first activity will not lead to a larger set of mutually compatible activities.

The proof will be by contradiction. Suppose James is right, and eliminating the first activity from a set of n activities S will yield a set Aj that is larger than the set of picking the earliest finishing time Ae. Let a1, a2 … an be each individual activity in S. James is proposing to remove a1 and pick from the rest of S. Since a2, … an all have finishing times >= a1, there are essentially two possible cases. One case is that the finishing time for a1 and a2 are equal. In this case, the algorithm will continue on as if nothing had changed, making |Aj| = |Ae|. However, since James is claiming that |Aj| > |Ae|, his point has been disproven for the case when the finishing time of a2 is equal to a1. The other case is that a2 finishes after a1. In this case, the set S2 that contains all activities mutually compatible with a2 will contain the same or fewer number of activities as S1. If it contains the same activities as S1, then again the algorithm will continue on as normal until |Aj| = |Ae|. If it contains fewer activities than S1, then the algorithm will continue to pick the earliest finish time, but with the possibility each time of containing less, never more, activities per iteration than the original algorithm. The possibility of containing fewer activities means fewer iterations, and subsequently fewer activities in Sj. Thus, overall, |Sj| <= |Se|, which contradicts the earlier assumption that |Sj| > |Se|.

Exercise 4 (35 points)

1. (30 points) Write the pseudocode for an algorithm using dynamic programming to solve the activity-selection problem based on this recurrence (refer to lecture and textbook):



ACTIVITY-SELECTION(s, f)

1. s[0] = 0
2. f[0] = 0
3. s[s.Length] = max(f)
4. f[f.length] = max(f)
5. c = new 2D array [0… s.Length - 1, 0… f.Length - 1]
6. for i = 0 to s.Length - 1
7. for j = 0 to f.Length - 1
8. c[i, j] = -1
9. i = 0
10. j = f.Length - 1
11. ACTIVITY-SELECTION-WITH-MEMOIZATION(s, f, i, j)

ACTIVITY-SELECTION-WITH-MEMOIZATION(s, f, i, j)

1. a = 0
2. for x = i + 1 to j – 1
3. if s[x] >= f[i] && f[x] <= s[j]
4. a++
5. if a == 0
6. return 0
7. for k = i to j
8. if c[i, k] < 0
9. c[i, k] = ACTIVITY-SELECTION-WITH-MEMOIZATION(s, f, i, k)
10. if c[k, j] < 0
11. c[k, j] = ACTIVITY-SELECTION-WITH-MEMOIZATION(s, f, k, j)
12. if c[i, k] >= 0 && c[k, j] >= 0
13. c[i, j] = c[i, k] + c[k, j] + 1
14. return c[i, j]
15. (5 points) Analyze the running time (time complexity) of your algorithm and compare it to the iterative greedy algorithm.

Overall, this algorithm creates a 2D array that has to be completely filled. Letting n be the size of s.length, the array is (n + 2) x (n + 2). This means that the size of the 2D array is dependent on n, and the asymptotic running time of filling up the array is Ω(n2). Since the greedy algorithm only has to scan the array once, its asymptotic running time is θ(n), meaning that the greedy algorithm is significantly more efficient that the dynamic programming approach to this algorithm.

What you need to turn in:

* Electronic copy of this file (including your answers) (standalone). Submit the file as a Microsoft Word or PDF file.
* Recall that answers must be well written, documented, justified, and presented to get full credit.
* How this assignment will be graded:
* A right answer will get full credit when:
* It is right (worth 25%)
* It is right AND neatly presented making it easy and pleasant to read. (worth 15%)
* There is an obvious and clear link between 1) the information provided in the exercise and in class and 2) the final answer. A clear link is built by properly writing, justifying, and documenting an answer (worth 60%).
* Calculation mistakes will be minimally penalized (2 to 5% of full credit) while errors on units will be more heavily penalized.
* You are welcome/encouraged to discuss exercises with other students or the instructor. But, ultimately, personal writing is expected.