Computer Science 385
Design and Analysis of Algorithms
Siena University
Spring 2026

Problem Set 5

Due: 4:00 PM, Wednesday, April 15, 2026

You may work alone or in a group of size 2 or 3 on this assignment. However, in order to make sure you learn the material and are well-prepared for the exams, you should work through the problems on your own before discussing them with your partner, should you choose to work with someone. In particular, the "you do these and I'll do these" approach is sure to leave you unprepared for the exams.

Submitting

Please submit a hard copy (typeset preferred, handwritten OK but must be legible) for all written questions. Only one submission per group is needed.

Dynamic Programming Practice

Read Section 8.1 of Levitin to learn all about the robot coin collection problem. There, you will find a bottom-up dynamic programming solution to this problem.

Question 1: Complete the pseudocode below that uses a **recursive exhaustive search algorithm** (**not** dynamic programming) to solve the robot coin collection. (6 points)

ALGORITHM ROBOTCOINCOLLECTION(r, c, C)
//Input: r, c, starting row and column position
//Input: $C[1..n, 1..m]$ matrix of 0 and 1 values indicating if a coin exists at each position
//Output: the max number of coins collected when the robot reaches (n, m)

Question 2: Rewrite your algorithm from above so that it uses **top-down dynamic programming** to get a recursive algorithm that is more efficient. (4 points)

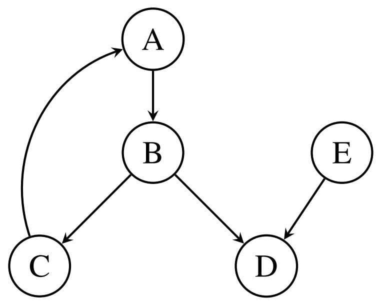
ALGORITHM ROBOTCOINCOLLECTION($r, c, C, sols$)
//Input: r, c, starting row and column position
//Input: $C[1..n, 1..m]$ matrix of 0 and 1 values indicating if a coin exists at each position
//Input: $sols[1..n, 1..m]$ initialized to all -1
//Output: the max number of coins collected when the robot reaches (n, m)

Question 3: Give the Θ efficiency class of your top-down dynamic programming algorithm and briefly explain how you arrived at this bound. (2 points)

Transitive Closure Algorithms

Refer to Levitin Section 8.4, Warshall's algorithm and Floyd's algorithm, to complete this section.

Consider this directed graph:



Question 4: Draw the adjacency matrix representation of this graph. (2 points)

One way to compute the transitive closure is to, for each vertex, compute the spanning tree for that vertex using an algorithm like breadth-first or depth-first traversal, and add edges from that start vertex to all vertices in the spanning tree.

Question 5: Assuming an adjacency matrix representation, give the worst case Θ efficiency class of computing the spanning tree of a graph with V vertices. (1 point)

Question 6: What edges will be added to the graph after the spanning tree is computed for each vertex. Process the vertices in alphabetical order. (3 points)

Question 7: Give the Θ efficiency class for the algorithm to find the transitive closure using this method of computing the the spanning tree starting at each vertex. (1 point)

Question 8: Draw the final graph, both as a diagram like above showing the vertices and edges, and in adjacency matrix form. (2 points)

Question 9: Now use Warshall's Algorithm to compute the transitive closure for the same graph. Show the graph both in adjacency matrix form and visually (vertices and edges) after each iteration of the outer loop of the alforithm. These correspond to the R^i steps in Figure 8.13. (3 points)

Question 10: Give the Θ efficiency class of Warshall's algorithm for a graph with V vertices. (1 point)

Now consider the weighted digraph represented by the following adjacency matrix W, where W_{ij} represents the weight of the edge from vertex i to vertex j:

$$W = \begin{pmatrix} & A & B & C & D & E \\ A & \infty & 5 & 12 & \infty & \infty \\ B & \infty & \infty & 2 & 7 & \infty \\ C & \infty & \infty & \infty & \infty & 15 \\ D & \infty & \infty & \infty & \infty & \infty \\ E & \infty & 4 & \infty & 9 & \infty \end{pmatrix}$$

Question 11: Draw the graph showing vertices and edges with the edge weights. (1 point)

Question 12: Apply Floyd's Algorithm to this graph. At each iteration, show the graph in adjacency matrix form, but only for the first and last steps also show it visually (vertices and edges) (5 points)

Refresher on Heaps and Heapsort

For this part of the problem set, you will review and expand your understanding of the heap data structure and the heapsort algorithm. If you have not studied or do not recall the details of heaps and heapsort, please review Levitin Section 6.4 and/or the topic notes on Heaps linked from the schedule page.

Question 13: Consider the construction of a min-heap by starting with an empty heap, then inserting values in a given order. Show the min-heap after each of the values 25, 8, 35, 39, 12, 3, and 31 are added to an initially empty heap. (4 points)

Question 14: Consider the construction of a min-heap as done in the first phase of a heapsort. Start with an array containing the values 25, 8, 35, 39, 12, 3, and 31. Show the contents of the array (in array or tree form, your choice) after each non-trivial iteration of the first phase of the heapsort (which goes from an unsorted array to a min-heap). (4 points)

Question 15: Finally, show the transformation from the min-heap you just constructed to a sorted array, completing the heapsort procedure. (4 points)

Question 16: Consider a binary min-heap like the ones in Levitin Section 6.4. Which locations in a binary min-heap of n elements could possibly contain a) the second-smallest element, b) the third-smallest element and c) the largest element? (3 points)

Generalized Heaps and Heapsort

The heaps you know from data structures and class, where the heap is represented by a binary tree stored in an array, are one specific case of a more general structure called a *d-heap*. In a d-heap, each node has up to d children. So the binary heaps you worked with in the previous section would be considered "2-heaps". For the questions below, assume that the minimum value is stored at the root node (*i.e.*, that it is a min-heap), and that the following numbers are added to the heap in this order: 7, 3, 9, 6, 1, 10, 11, 5.

Question 17: Show the construction of a 2-heap that results from the values being inserted. How many comparisons are needed? (3 points)

Question 18: Show the construction of a 3-heap that results from the values being inserted. How many comparisons are needed? (3 points)

Question 19: For the heap element at position i in the underlying array of a 3-heap, what are the positions of its immediate chidren and its parent? (Give formulas in terms of i.) (1 point)

Question 20: For the heap element at position i in the underlying array of a d-heap, what are the positions of its immediate chidren and its parent? (Give formulas in terms of i and d.) (1 point)

Question 21: Show the construction of a 1-heap that results from the values being inserted. How many comparisons are needed? (3 points)

Question 22: Show the construction of a 7-heap that results from the values being inserted. How many comparisons are needed? (3 points)

You also reviewed heapsort in the previous section. In a sense, heapsort uses a 2-heap as an intermediate representation to sort the contents of an array. Let's consider a generalization of the heapsort idea. It's not exactly the same, since it uses a secondary data structure, but the general behavior is similar.

- First, insert the elements to be sorted into a priority queue (PQ).
- Then, remove the elements one by one from the PQ and place them, in that order, into the sorted array.

For heapsort, the PQ is a 2-heap, but any PQ implementation would work (naive array- or list-based with contents either sorted or unsorted, a d-heap, or even a binary search tree). Depending on which underlying PQ is used, the sorting procedure will proceed in a manner similar, in terms of the order in which comparisons occur, to one of the other sorting algorithms we have studied (*e.g.*, selection sort, quicksort, *etc.*).

Question 23: For each of the following underlying PQ structures, state which sorting algorithm proceeds in the manner most similar to the PQ-based sort using that PQ structure, and explain your answer. Each response should be at least a few sentences long, and should discuss how the pattern of comparisons and swaps, and the resulting efficiency relates to the sorting algorithm. (10 points)

1. 1-heap
2. 3-heap
3. (n-1)-heap
4. binary search tree
5. balanced binary search tree

Grading

This assignment will be graded out of 70 points.

Feature	Value	Score
Question 1	6	
Question 2	4	
Question 3	2	
Question 4	2	
Question 5	1	
Question 6	3	
Question 7	1	
Question 8	2	
Question 9	3	
Question 10	1	
Question 11	1	
Question 12	5	
Question 13	4	
Question 14	4	
Question 15	4	
Question 16	3	
Question 17	3	
Question 18	3	
Question 19	1	
Question 20	1	
Question 21	3	
Question 22	3	
Question 23	10	
Total	70	