Name: Rollno:

ESO207: Data Structures and Algorithms (Final Exam)

19th November 2025

Total Number of Pages: 10 Total Points 100

Instructions

1. All questions are compulsory.
2. Answer all the questions in the space provided in the question paper booklet.
3. Use the space provided in the paper for rough work.
4. The symbols or notations mean as usual unless stated.
5. You may cite and use algorithms and their complexity as done in the class.
6. Cheating or resorting to any unfair means will be severely penalized.
7. Superfluous and irrelevant writing will result in negative marking.
8. Using pens (blue/black ink) and not pencils. Do not use red pens. for answering.
9. Please bring your ID cards.
10. For questions involving algorithm design, it is sufficient to give a clear and concise description of your algorithm. You dont need to give the formal pseudocode.
11. If you design an algorithm, that is less efficient than what has been asked, you will get partial credit, provided you clearly mention what is the time complexity of your algorithm.
12. It is advisable to solve a problem first before writing down the solution.
13. The questions are *not* arranged according to the increasing order of difficulty.

Question	Points	Score
1	25	
2	7	
3	8	
4	10	
5	10	
6	10	
7	15	
8	15	
Total:	100	

Question 1. Provide short answers for the following questions (no more than 1 line):

(a) (2 points) Let A denote the adjacency matrix for a graph $G = (V, E)$, then what does $A^3(i, j)$ denote? (where $A^3(i, j)$ represents $(i, j)^{th}$ entry of the A^3 matrix)

Solution: All walks of length 3 from i to j.
Grading Rubric: If path is mentioned instead of walk then 1 mark is deducted.

(b) (3 points) A weighted graph with distinct weight on all edges has a unique MST. (**True/False**) Give reason for your answer.

Solution: True
For any cut of the graph the unique lightest edge crossing that cut must belong to every MST. Since all weights are distinct the lightest crossing edge for each cut is unique, and that forces a unique choice of edges across all cuts, giving a unique MST.

(c) (2 points) For a directed graph $G = (V, E)$, what is the value of $\sum_{u \in V}$ outdegree(u)? (outdegree(u) denotes the outdegree of a vertex u)

Solution: $|E|$

(d) (3 points) Let e be some edge having minimum weight in G. Then e must be present in every MST of G. (**True/False**). Justify your answer.

Solution: False. Consider a cycle with all edges having the same weight.

(e) (2 points) Solve the following recurrence relation and give a Θ bound for $T(n)$: $T(n) = 8n + T(n/10) + T(8n/9)$. You need not show the steps.

Solution: $T(n) = \Theta(n)$
Grading Rubric: No analysis is required. But if incorrect analysis is given then 1 mark is deducted.

(f) (2 points) Here is an array which has just been partitioned by the first step of quicksort : 3, 0, 2, 4, 5, 8, 7, 6, 9. Which of the elements in the array could be the pivot?

Solution: 4, 5, or 9
Grading Rubric: 1 mark given if one or two pivots are correct.

(g) (2 points) Given a sorted array A, the number of occurrences of a given number x in A can be computed in time $O(\log n)$. (**True/False**). Justification is not needed.

Solution: True

(h) (2 points) Finding the maximum element in a min-heap will require $\Omega($________________$)$ comparisons.

Solution: n

(i) (2 points) Let T be a shortest path tree of a weighted directed graph, with positive edge weights, and rooted at s. Consider any path in T from s to a leaf. What can we say about the order of the sequence of vertices in this path?

Solution: The vertices in this sequence are in increasing order of their distance from s in the graph.

(j) (1 point) Suppose the numbers 7, 5, 1, 8, 3, 6, 0, 9, 4, 2 are inserted in that order into an initially empty binary search tree. What is the in-order traversal sequence of the resultant tree?

Solution: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

(k) (1 point) The time taken to build a heap from a set on n integers is $O($____________$)$. (Give your answer in O-notation.)

Solution: $O(n)$

(l) (3 points) For each of the following three sorting algorithms, mention whether they are In Place and/or Stable sorting algorithms – MergeSort, QuickSort, RadixSort

Sorting Algorithm	In Place (yes/no)	Stable (yes/no)
MergeSort	**no**	**yes**
QuickSort	**yes**	**no**
RadixSort	**no**	**yes**

Question 2. (7 points) Given k sorted arrays where each array can have a maximum of n elements. Design an $O(nk \log k)$ algorithm to merge the arrays and print the output. Provide time complexity analysis for your algorithm.

Solution: Here are minimum two possible approaches. Marks are awarded for other approaches too. Step marks are given accordingly.

Solution 1: Divide and Conquer based approach

1. Club the arrays into groups of two and merge them.
2. Repeat the above step until a single array remains.

Time Analysis

Let cn be the time taken to merge two sorted arrays of length n. Then the first iteration of step 1 takes time $cn \cdot k/2$, second iteration will take $2cn \cdot k/4 = cn \cdot k/2$ and so on. Since there are k sorted arrays initially and in each iteration the number of sorted arrays are halved, the number of iterations will be $\log k$. Therefore total time taken is $cnk \log k/2 = O(nk \log k)$.

Grading Rubric

1. 4 marks for the correct algorithm. Atmost 2 marks are awarded if the divide or merge step is not clearly explained.
2. 2 marks for the time complexity analysis. A formal analysis is required, though it is not necessary to derive recurrence relation. Atmost 1 mark awarded if someone has directly written there are $logk$ steps without any explanation or written recurrence relation directly without any justification. Zero marks in this part for incorrect algorithm or unclear time complexity analysis.

Solution 2 : Heap based approach

1. Build a min-heap of size k which contains the first (smallest) element from every k arrays. Each element is stored in form of a pair (v, j), where v is the value of the element from j^{th} array.
2. Declare an empty list L (output list).
3. Repeat the following until the heap becomes empty.
   a. Extract the minimum element from the heap via EXTRACT-MIN operation. Say item (v, j) is returned. Insert v into the list L.
   b. Get the next element from the j^{th} array and insert it into in the heap using INSERT operation. Ignore the step if the j^{th} array contains no next element (becomes empty).

Time Analysis

Step 1 takes $O(k)$ time to build a heap of k elements. Note the heap will be atmost of size k. Step 3a will take $O(logk)$ to perform EXTRACT-MIN and Step 3b will also take $O(logk)$ time for INSERT operation. Since nk elements are inserted and extracted atmost once, hence, total time complexity will be $O(nklogk)$.

Grading Rubric

(a) 4 marks for the correct algorithm: 1 mark of stating that a heap of size k is used, 1 mark for stating that the heap elements are of the form (v, j) or stating that indices of array are kept tracked of along with value of elements, 1 mark for step 3a and 1 mark for step 3b.

(b) 2 marks for the time complexity analysis. If time complexity of any step in the algorithm is not clearly stated, then atmost 1 mark may be awarded. Zero marks in this part for incorrect algorithm or unclear time complexity analysis.

Question 3. (8 points) Given a connected, unweighted, undirected graph $G = (V, E)$ and two distinct vertices, s (source) and t (target), design an $O(m)$ time algorithm that finds the number of distinct shortest length paths from s to t, where $n = |V|$ and $m = |E|$. Provide a brief time complexity analysis for your algorithm as well.

Solution: Perform a breadth-first search (BFS) starting from s. BFS naturally explores the graph in increasing order of distance, so the first time we reach any vertex we know that we have discovered a shortest path to it.

During the BFS, for each vertex we keep track of two pieces of information: (1) the length of the shortest path discovered so far from s to that vertex, and (2) how many different shortest paths from s reach that vertex.

Initialize the distance to every vertex as ∞ and giving the source s a distance of zero. We also initialize the number of shortest paths to each vertex as zero, except for the source, which has exactly one shortest path (the trivial path of length zero). We place the source in a queue and start BFS.

When we remove a vertex u from the queue, we examine each of its neighbors v. If a neighbor has never been seen before, then the shortest path to it must go through u. We set its distance to be one more than the distance to u, set the number of shortest paths to match the number of shortest paths to u, and add it to the queue. If the neighbor has already been seen but the distance we discovered for it is exactly one more than the distance to u, then we have found an additional shortest path to that neighbor (via u). In this case, we add the number of shortest paths to u to the count stored for that neighbor.

We continue this process until BFS finishes. At that point, the value stored for the target vertex t tells us exactly how many shortest paths exist from s to t.

Time complexity is same as BFS. Since the graph has only one component, time is $O(m)$.

Question 4. (10 points) Prove that the following three statements are equivalent for a graph $G = (V, E)$ where $|V| = n$ and $|E| = m$.

1. G is connected and acyclic.
2. G is connected and $m = n - 1$.
3. There is a unique path between every pair of vertices in G.

Solution: We prove that the three statements are equivalent by showing (1) $\Rightarrow$ (3) $\Rightarrow$ (1) and (1) $\Leftrightarrow$ (2).

(1) $\Rightarrow$ (3). Assume that G is connected and acyclic. Take any two vertices $u, v \in V$. Since G is connected, there is at least one u–v path. Suppose, for contradiction, that there are two distinct simple paths P_1 and P_2 between u and v. Let x be the first vertex where P_1 meets P_2 when traversing from u. The subpaths of P_1 and P_2 between consecutive common vertices form a cycle, contradicting acyclicity. Thus, between any two vertices there is at most one path. Combining this with connectivity, we conclude that there is *exactly* one path between every pair of vertices. Hence (3) holds.

(3) $\Rightarrow$ (1). Assume that there is a unique path between every pair of vertices in G. Uniqueness of paths immediately implies that G is connected. Suppose G contained a cycle C, and pick two distinct vertices x and y on C. Traversing C in the two different directions yields two distinct x–y paths, contradicting uniqueness. Hence G contains no cycles and is therefore acyclic. Thus (1) holds.

Therefore, (1) and (3) are equivalent.

(1) $\Rightarrow$ (2). Assume that G is connected and acyclic. Then G is a tree. We show that $m = n - 1$ by induction on n. If $n = 1$, then $m = 0 = n - 1$. For $n \geq 2$, any finite acyclic graph contains a vertex of degree 1 (a leaf). Remove this vertex and its incident edge. The remaining graph is connected and acyclic with $n - 1$ vertices, so by the inductive hypothesis it has $n - 2$ edges. Reinserting the removed vertex and edge gives $m = (n - 2) + 1 = n - 1$. Hence $m = n - 1$.

(2) $\Rightarrow$ (1). Assume that G is connected and $m = n - 1$. Any connected graph contains a spanning tree, which has exactly $n - 1$ edges. Since G already has $n - 1$ edges, all of them must belong to the spanning tree. Therefore G contains no additional edges and must itself be acyclic. Thus G is connected and acyclic.

Conclusion. All three statements are equivalent. $\square$

Question 5. Consider an undirected graph $G = (V, E)$ with non-negative edge weights ($w(e) \geq 0$). Suppose that you have computed a minimum spanning tree of G, and that you have also computed shortest paths to all nodes from a particular node $s \in V$. Now suppose each edge weight is increased by 1, that is, the new weights are $w'(e) = w(e) + 1$.

(a) (5 points) Does the minimum spanning tree change? Give an example where it changes or prove that it does not change.

Solution: Given an MST, if all edge weights are increased by 1, then the MST does not changes. .

Every spanning tree of G has $|V| - 1$ edges. Now increasing the weight of every edge by 1, increases the weight of every spanning tree by $|V| - 1$. Hence in T was an MST with respect to w, then T would be an MST with respect to w' as well.

Grading Rubric

The rigorous proof / explanation is required while solving this.

- +0.5 : For mentioning does not change.
- +4.5: For the mathematical details and logical explanation.
- At-least 1.5 points deducted if mathematical details using contradiction missing .
- At-least 1 point deducted if logical explanation using cut/cycle property missing or not used appropriately.
- At-least 1 point deducted if property of MST about order of weight preserving not mentioned.
- At least1 point deducted if uniqueness of MST not explained properly.
- 0 point awarded if the proof/explanation is missing.

(b) (5 points) Do the shortest paths change? Give an example where they change or prove they do not change.

Solution: Yes, it can change. Counterexample

Grading Rubric

- +0.5 for writing it does changes.
- +4.5 for giving an example correctly.
- At-least 2.5 points deducted if example is not clear.
- 0 point awarded if the example is missing.

Question 6. Alice has an infinite number of 1-cent and 2-cent coins. Each coin has two faces : Heads and Tails. Let $S(n)$ be the number of ways to create a linear arrangement of some of these coins such that their sum is n cents, given that the first coin in the arrangement should always have Heads facing up. All other coins could have either have Tails or Heads facing up.

For example : If $n = 2$, the possible arrangements are (1H,1H), (1H,1T), (2H) where H is heads and T is tails. Therefore there are 3 possible arrangements that sum up to 2 cents.

(a) (5 points) Derive the recurrence relation for $S(n)$.

Solution: $S(n) = 2S(n-1) + 2S(n-2)$, $S(2) = 3$ and $S(1) = 1$.

Grading Rubric

- +3 for the correct recurrence relation. +1 for every base case, $S(1)$, and $S(2)$. $S(0)$ does not hold any significance in this problem.

(b) (5 points) Give an algorithm to compute $S(n)$ given n, in $O(\log n)$ time. Show the time complexity analysis of your algorithm.

Solution: Note that

$$\begin{bmatrix} S(n) \\ S(n-1) \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} S(n-1) \\ S(n-2) \end{bmatrix}$$

Therefore,

$$\begin{bmatrix} S(n) \\ S(n-1) \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 1 & 0 \end{bmatrix}^{n-2} \begin{bmatrix} S(2) \\ S(1) \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 1 & 0 \end{bmatrix}^{n-2} \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

Using matrix exponentiation,

$$\begin{bmatrix} 2 & 2 \\ 1 & 0 \end{bmatrix}^{n-2}$$

can be computed in $O(\log n)$ time.

Grading Rubric

- +2 for the first step, and +2 for the second step as shown above. +1 for time complexity analysis.

Question 7. Given an array of positive integers $A[0 \dots n-1]$, design a data structure tree to support the following operations:

1. UPDATE(i, x): Set $A[i] \leftarrow x$.
2. GCD(l, r): Return $\gcd(A[l], A[l+1], \dots, A[r])$.

Each operation must run in $O(\log n)$ time. Assume n is a power of two.

(a) (4 points) Describe the data structure that you will use. How is the data structure constructed and what is the time complexity of constructing the data structure.

Solution: Node Information
Each node of the segment tree corresponds to an interval $[L, R]$ and stores:

$$g = \gcd(A[L], A[L+1], \dots, A[R]).$$

Building the Tree
For a leaf node $[i, i]$, store:

$$g = A[i].$$

For an internal node with children storing g_1 and g_2:

$$g = \gcd(g_1, g_2).$$

Since each merge takes $O(1)$ time, the build runs in $O(n)$ time.

(b) (5 points) Give an algorithm for the UPDATE(i, x) operation.

Solution: Point Update
To perform UPDATE(i, x):

1. Change the leaf storing $A[i]$ to x.
2. Recompute all ancestor nodes using

$$\text{g} = \gcd(\text{left.g}, \text{right.g}).$$

This updates $O(\log n)$ nodes.

(c) (6 points) Give an algorithm for the GCD(l, r) operation.

Solution: Range GCD Query
To answer QUERY(l, r), we recursively visit nodes whose intervals overlap with $[l, r]$. If a node is fully inside the query range, we use its stored gcd value.
Since at most $O(\log n)$ nodes are combined and each merge computes a gcd in constant time, the query runs in $O(\log n)$.

Question 8. You are given n jobs. Each job i has:

- a deadline $d_i \in \mathbb{N}$,
- a profit $r_i > 0$,
- and requires exactly 1 unit of time to complete.

A job completed at time $t \leq d_i$ yields profit r_i, otherwise it yields 0 profit.

(a) (5 points) Design a greedy algorithm to select and schedule jobs in order to maximize the total profit.

Solution: Algorithm:
The greedy idea is to:

- schedule jobs in decreasing order of profit, and
- place each job as late as possible (before its deadline).

(b) (10 points) Give the proof of correctness of your algorithm.

Solution: Proof of Correctness:

We show that choosing the job with maximum profit among all remaining jobs is safe.

Lemma 1. *Let J be the job with maximum profit. There exists an optimal schedule that includes J.*

Proof. Consider any optimal schedule S.

Case 1: S already contains J. Nothing to prove.

Case 2: S does not contain J. Let k be the job scheduled in S at or before deadline d_J whose time slot we will replace. Since J has maximum profit,

$$r_J \geq r_k.$$

If we replace job k with job J in that slot, the schedule remains feasible (since the slot is $\leq d_J$), and the profit does not decrease.

Thus we obtain another optimal schedule that includes J. □

This proves that scheduling the highest-profit job first is always safe. By the lemma, there exists an optimal solution that schedules the highest-profit job.

We show that scheduling it in the *latest* feasible slot is also safe.

Lemma 2. *If job J has deadline d_J, scheduling it in the latest available slot $t \leq d_J$ preserves optimality.*

Proof. Suppose S is an optimal schedule that includes job J, but places it in an earlier slot $t' < t$. Slot t must be occupied in S. Swap the job in slot t with job J:

- J can be moved to t since $t \leq d_J$.
- The job originally at slot t can move to slot t' (or earlier) because $t' < t$.

Thus feasibility is maintained and total profit is unchanged.

By repeated swapping if necessary, we can transform S into one that places J at slot t. □

Combining both lemmas, we conclude that, scheduling jobs in decreasing order of profit and placing each as late as possible yields an optimal schedule.