Student Assignment Brief

Contents

Assignment Information

Module Name:	Advanced Algorithms
Module Code:	ST5003CEM
Assignment Title:	Coursework
Assignment Due:	25th January 2026 [11:55 PM]

Assignment Credit:	10 credits
Word Count:	N/A
Assignment Type:	Coursework
Grading:	67%

Assessment Overview

You will be provided with an overall grade between 0% and 100%. You have one opportunity to pass the assignment at or above 40%.

Important Notice

The work you submit for this assignment must be your **own independent work**. More information is available in the 'Assignment Task' section of this assignment brief.

Assessed Module Learning Outcomes

On completion of this module the student should be able to:

1. Understand and select appropriate algorithms for solving a range of problems and reason about their complexity and efficiency.
2. Design and implement algorithms and data structures for novel problems.
3. Specify and implement methods to estimate solutions to intractable problems.
4. Describe the issue of data consistency in multi-threaded applications.
5. Design and implement a concurrent application.

Assignment Task

Question 1

a)

Optimizing Sensor Placement for Data Collection

Problem: Optimal Sensor Placement for Signal Strength

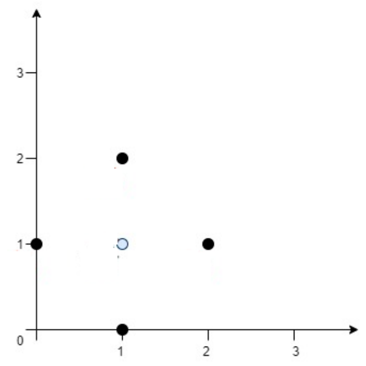
A research team is deploying a network of static environmental sensors across a remote area. Each sensor is located at a known geographical coordinate on a 2D plane. To maximize the efficiency of data collection and minimize signal attenuation, the team wants to establish a central data aggregation hub. The ideal location for this hub is one that minimizes the total sum of the Euclidean distances to all deployed sensors, as signal strength degrades with distance.

Given an array sensor_locations where sensor_locations[i] = [xi, yi] represents the coordinates of the i-th environmental sensor on the map, return the minimum total sum of the Euclidean distances from the data aggregation hub to all sensors.

In other words, you need to choose the optimal position for the data aggregation hub [hub_x, hub_y] such that the following formula is minimized:

$$\sum_{i=0}^{N-1} \sqrt{(sensor\_locations[i][0] - hub\_x)^2 + (sensor\_locations[i][1] - hub\_y)^2}$$

Example 1:

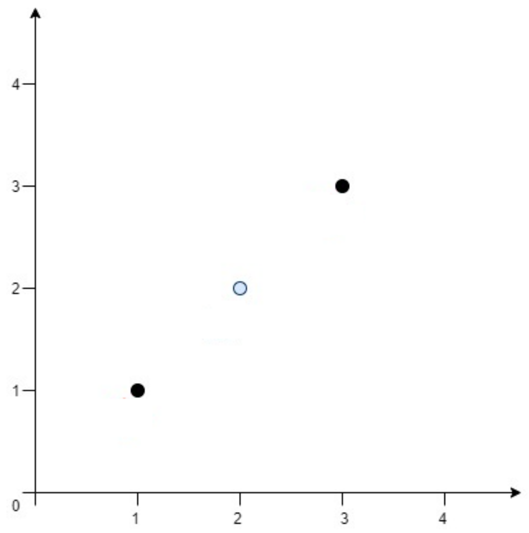


Input: sensor_locations = [[0,1],[1,0],[1,2],[2,1]]

Output: 4.00000

Explanation: Placing the hub at [hub_x, hub_y] = [1, 1] results in a distance of 1 to each sensor. The sum of all distances is 4, which is the minimum achievable.

Example 2:



Input: sensor_locations = [[1,1],[3,3]]

Output: 2.82843

Explanation: The minimum possible sum of distances is

$\sqrt{2} + \sqrt{2} = 2.82843.$

[5 Marks]

b)

The Traveling Salesperson Problem (TSP) - Classic Application

Problem: You are a logistics manager for a delivery company, and you have a list of N customer locations (cities) you need to visit, starting and ending at your depot. Your goal is to find the shortest possible route that visits each customer exactly once.

Task:

1. Generate a TSP Instance: Create a set of N cities (e.g., N=20 to 50 for manageable computation) with random 2D coordinates (e.g., between 0 and 1000). The "distance" between any two cities is their Euclidean distance.
2. Implement Simulated Annealing: Apply the Simulated Annealing algorithm to find a near-optimal solution to this TSP instance.
    - State: A permutation of cities representing a tour.
    - Objective Function: Total Euclidean distance of the tour.
    - Neighborhood: Common neighborhood operations for TSP include:
        - 2-opt: Reverse a segment of the tour (e.g., A-B-C-D-E to A-B-D-C-E).
        - Swap: Swap two random cities in the tour.
    - Cooling Schedule: Experiment with at least two different cooling schedules (e.g., exponential T = T_initial * alpha^k vs. linear T = T_initial - beta * k).
    - Stopping Criteria: Max iterations, or temperature falling below a threshold.

[5 Marks]

Question 2

Scenario: The "Strategic Tile Shatter" Game

Problem: You're playing a new single-player arcade game called "Strategic Tile Shatter." In this game, you are presented with a row of n intricately designed ceramic tiles, each marked with a unique score multiplier. Your objective is to shatter all the tiles one by one.

When you shatter the i-th tile, you earn points based on the score multipliers of its adjacent tiles. Specifically, if you shatter tile i, you gain points equal to the product of the score multipliers of the tile immediately to its left, the tile i itself, and the tile immediately to its right.

A special rule applies to tiles at the ends:

If tile i-1 is out of bounds (meaning tile i is the leftmost tile remaining), its score multiplier is treated as 1.

If tile i+1 is out of bounds (meaning tile i is the rightmost tile remaining), its score multiplier is treated as 1.

Your challenge is to devise a strategy to shatter all the tiles in a specific order to maximize your total accumulated points.

Formal Definition:

Given an array tile_multipliers where tile_multipliers[i] is the score multiplier painted on the i-th tile (indexed from 0 to n-1), determine the maximum total points you can collect by shattering the tiles optimally.

Constraints:

This problem must be solved using a Dynamic Programming approach. You are expected to define your dp state, recurrence relation, base cases, and the order of computation.

Example 1:

Input: tile_multipliers = [3, 1, 5, 8]

Output: 167

Initial State: [3, 1, 5, 8]

Shatter 1: 3 * 1 * 5 = 15 points. Remaining: [3, 5, 8]

Shatter 5: 3 * 5 * 8 = 120 points. Remaining: [3, 8]

Shatter 3: 1 * 3 * 8 = 24 points. Remaining: [8] (Note: 1 is for the left out-of-bounds, 8 is to its right)

Shatter 8: 1 * 8 * 1 = 8 points. Remaining: [] (Note: 1 for both left and right out-of-bounds)

*Total Points = 15 + 120 + 24 + 8 = 167

Example 2:

Input: tile_multipliers = [1, 5]

Output: 10

*Shatter 1: 1 * 1 * 5 = 5 points. Remaining: [5]

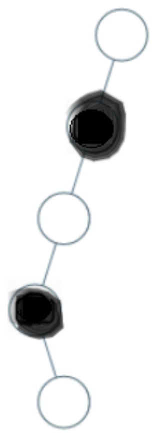
*Shatter 5: 1 * 5 * 1 = 5 points. Remaining: []

*Total Points = 5 + 5 = 10

[5 Marks]

Question 3

You are given a binary tree that represents cities where a certain corporation wishes to provide service by constructing a service center. Building service centers at any node, i.e. a city can give service to its immediate ancestor and descendant node (cities), returns the smallest number of service centers required by the corporation to provide service to all connected cities.



Input: tree= {0,0, null, 0, null, 0, null, null, 0}

Output: 2

[5 Marks]

Question 4

Smart Energy Grid Load Distribution Optimization (Nepal)

In a progressive move towards sustainable energy usage, a city in Nepal has implemented a smart grid. This system dynamically allocates energy from various sources—solar, hydro, and diesel generators—to meet the power demands of its districts on an hourly basis.

As a systems engineer, your task is to develop a load balancing algorithm using advanced techniques like Dynamic Programming and Greedy Strategy. The goal is to optimally distribute energy to each district for every hour of the day.

Inputs

Hourly Energy Demand Table (in kWh):

Hour	District A	District B	District C
06	20	15	25
07	22	16	28
...	...	...	...

Export to Sheets

Energy Source Table:

Source ID	Type	Max Capacity/hr (kWh)	Available Hours	Cost/kWh (Rs.)
S1	Solar	50	06–18	1.0
S2	Hydro	40	00–24	1.5
S3	Diesel	60	17–23	3.0

Export to Sheets

Here's a rephrased version of the smart energy grid scenario, with costs expressed in Nepali Rupees (Rs.):

⍰ Real-World Context: Smart Energy Grid Load Distribution Optimization (Nepal)

In a progressive move towards sustainable energy usage, a city in Nepal has implemented a smart grid. This system dynamically allocates energy from various sources—solar, hydro, and diesel generators—to meet the power demands of its districts on an hourly basis.

As a systems engineer, your task is to develop a load balancing algorithm using advanced techniques like Dynamic Programming and Greedy Strategy. The goal is to optimally distribute energy to each district for every hour of the day.

🔢 Inputs

Hourly Energy Demand Table (in kWh):

Hour	District A	District B	District C
06	20	15	25
07	22	16	28
...	...	...	...

Export to Sheets

Energy Source Table:

Source ID	Type	Max Capacity/hr (kWh)	Available Hours	Cost/kWh (Rs.)
S1	Solar	50	06–18	1.0
S2	Hydro	40	00–24	1.5
S3	Diesel	60	17–23	3.0

Export to Sheets

Constraints:

Each district's demand must be met within ±10%.

Each energy source has limited capacity and specific hours of availability.

The objective is to minimize total cost and reduce diesel usage.

📌 Tasks (Marks total — distributed per task)

Model the Input Data (2 marks)

- Prepare or load the demand and source tables in your code using arrays or from JSON/CSV files.

Design an Hourly Allocation Algorithm (5 marks)

- Using Dynamic Programming, model each hour's assignment to fulfill all district demands by checking different combinations of energy sources under capacity and time constraints.

Implement Greedy Source Prioritization (3 marks)

- Within each hour, implement a greedy approach to always choose the cheapest available sources first before using costly sources like diesel.

Handle Approximate Demand Satisfaction (3 marks)

- Incorporate logic to allow for ±10% flexibility if an exact match of demand is not possible within source limits.

Output Table of Results (3 marks)

- Display results for each hour and district, showing energy drawn from each source, total used, actual demand, and % fulfilled.

Analyze Cost and Resource Usage (4 marks)

Report:

Total cost of distribution.

% of energy fulfilled by renewable sources.

Which hours/districts used diesel, and why.

Comment briefly on your algorithm's efficiency and trade-offs.

� Sample Descriptive Calculation (for Hour 06)

Demand:

District A: 20 kWh

District B: 15 kWh

District C: 25 kWh

→ Total Demand = 60 kWh

Available Sources at Hour 06:

Solar: 50 kWh (Available, Cost = Rs. 1.0)

Hydro: 40 kWh (Available, Cost = Rs. 1.5)

Diesel: Not available at 06

Step-by-step Allocation (Greedy by Cheapest):

Start with Solar (50 kWh):

Allocate:

District A: 20 kWh (fully met)

District B: 15 kWh (fully met)

District C: 15 kWh (partial)

→ Total used from Solar = 20 + 15 + 15 = 50 kWh

→ Remaining for District C = 25 – 15 = 10 kWh

Next use Hydro (max 40 kWh available):

District C: 10 kWh (remaining demand fully met)

→ Total used from Hydro = 10 kWh

Final Allocation Table:

Hour	District	Solar	Hydro	Diesel	Total Used	Demand	% Met
06	A	20	0	0	20	20	100%
06	B	15	0	0	15	15	100%
06	C	15	10	0	25	25	100%

Export to Sheets

Cost Calculation:

- Solar: $50 \times \text{Rs. } 1.0 = \text{Rs. } 50$
- Hydro: $10 \times \text{Rs. } 1.5 = \text{Rs. } 15$

$\rightarrow$ **Total Cost = Rs. 65**

[20 Marks]

Question 5

a)

Advanced Algorithm Design with GUI Integration

Project Title: Interactive Emergency Network Simulator

Scenario Overview: You are tasked with building an interactive GUI-based tool for simulating and optimizing a national Emergency Communication & Response System. The tool must allow users (e.g., planners, analysts) to visualize and interact with a dynamic graph-based network and a tree-based command structure, applying advanced algorithmic techniques to solve real-time problems.

System Requirements:

- A graphical interface that allows:
- Drawing cities (nodes) and roads (edges) with weights.
- Highlighting critical hubs and supply centers.
- Visualizing communication paths and failures.
- A command hierarchy viewer using tree structures to show organizational flow.
- User inputs to simulate failures, add/remove roads, and trigger algorithmic analysis.

Tasks:

Q1. Dynamic MST Visualization [6 marks]

- Implement an interactive feature where the user can:
- Click to generate and display the Minimum Spanning Tree connecting all supply centers and critical hubs.
- View updated MST when new edges or nodes are added.
- Show screenshots and explain the backend logic (e.g., Kruskal's or Prim's algorithm and time complexity).

Q2. Reliable Path Finder with GUI Controls [5 marks]

- Create a visual control panel for:
- Selecting any two nodes.
- Displaying K most reliable disjoint paths between them, avoiding marked "vulnerable roads".
- Describe how your algorithm integrates with the GUI and how users interact with it.

Q3. Command Hierarchy Optimizer [4 marks]

- Allow users to:
- View the current binary command tree.
- Click "Optimize" to rebalance the tree and minimize the longest communication path from HQ.
- Include visual before/after trees and explain the rebalancing algorithm.

Q4. Failure Simulation & Rerouting Module [5 marks]

- Build a simulation panel where:
- Users can disable a node (simulate failure).
- Automatically visualize:
- Nodes affected (disconnected).
- Recomputed shortest paths.
- Increase in delivery/communication time.
- Support your implementation with a GUI screenshot and logic explanation.

Bonus Task [+2 Marks]:

- Add a Graph Coloring visualizer for assigning frequencies to hubs such that no adjacent hubs use the same channel.
- Optional, creative implementation.

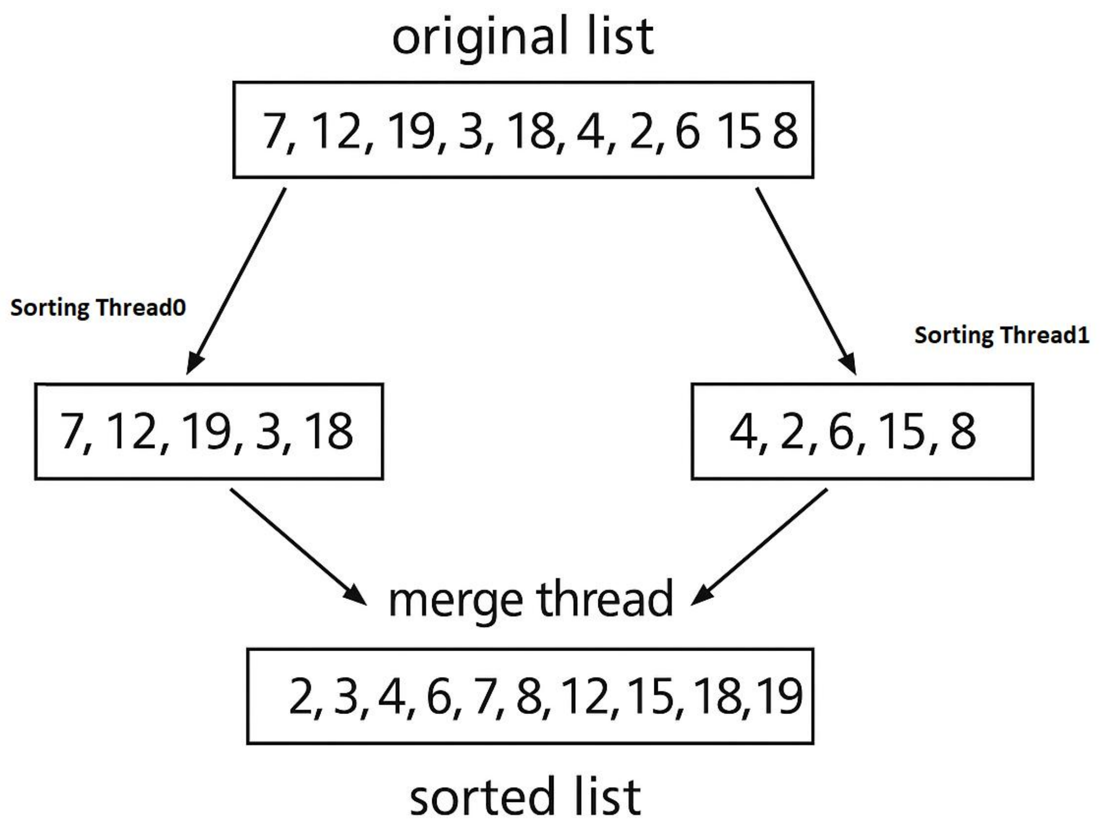
[20 Marks]

b)

Multithreaded Sorting Application

Write a multithreaded sorting program that works as follows: A list of integers is divided into two smaller lists of equal size. Two separate threads (which we will term sorting threads) sort each sublist using a sorting algorithm of your choice. The two sublists are then merged by a third thread—a merging thread—which merges the two sublists into a single sorted list. Because global data are shared across all threads, perhaps the easiest way to set up the data is to create a global array. Each sorting thread will work on one half of this array. A second global array of the same size as the unsorted integer array will also be established. The merging thread will then merge the two sublists into this second array. Graphically, this program is structured as in Figure Below.

This programming task will require passing parameters to each of the sorting threads. In particular, it will be necessary to identify the starting index from which each thread is to begin sorting. The parent thread will output the sorted array once all sorting threads have exited.



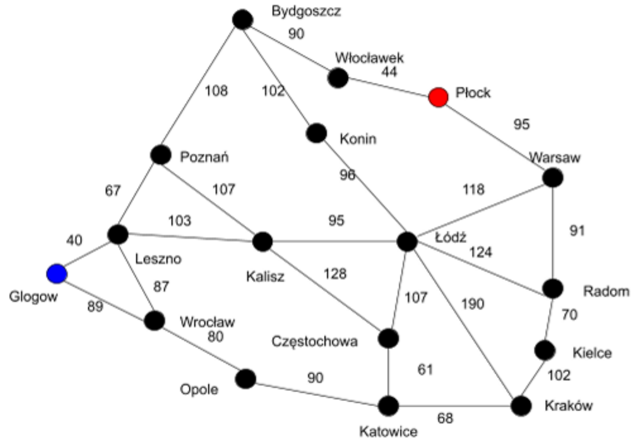
[10 Marks]

Task 6

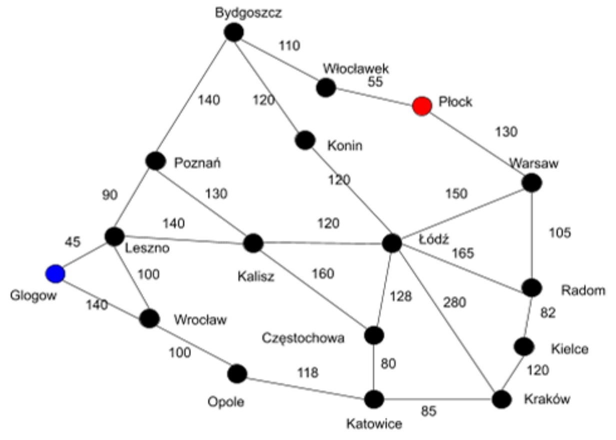
Problem Description

The diagram (a) below contains a simplified network mapping the geographical locations and distances between cities in Poland. There is a robot that can help to deliver parcels. Now, suppose you are developing an algorithm for this robot to achieve this task. The node highlighted in blue is your starting city, and the node in red is the goal city.

The diagram (b) below contains the straight-line distances between two cities:



(b)



(a)

Assessment Requirements

1. Based on the provided diagram map (a), build the state space (5 Marks).

a. Solve the above problem using a depth-first-search (DFS) algorithm. Use open and closed containers to explain the algorithm. (5 Marks)

b. Solve the above problem using a breadth-first-search (BFS) algorithm. Use open and close containers to explain the algorithm. (5 Marks)

2. Based on the provided diagram map (b), design the heuristic function (10 marks). Solve the above problem using the A* algorithm.

3. Discuss the advantages and disadvantages of BFS, DFS and A* search algorithms using the problem you analysed and the resulting solutions/paths as the context for your discussion (5 Marks)

[30 Marks]

Submission Instructions

Requirement	Details
File Naming	NAME_studentID
File Format	.docx/.pdf format
Submission Method	Campus 4.0 platform (submission link provided 2 weeks before deadline)

Marking and Feedback

How will my assignment be marked?

Your assignment will be marked by the Module Team using standardized criteria.

How will I receive grades and feedback?

Provisional marks will be released once internally moderated. Feedback will be provided alongside grades release within 2 weeks (10 working days).

What will I be marked against?

Details of the marking criteria for this task can be found in the Assessment Marking Criteria section at the end of this brief.

Grade Requirements

You must achieve 40% or above to pass this assessment. Ensure you understand the marking criteria for successful completion.

Assignment Support and Academic Integrity

Getting Help

If you have any questions about this assignment, please meet your respective module leader or teacher for more information.

Language Standards

You are expected to use effective, accurate, and appropriate language within this assessment task.

Academic Integrity

The work you submit must be your own. All sources of information need to be acknowledged and attributed; therefore, you must provide references for all sources of information and acknowledge any tools used in the production of your work, **excluding Artificial Intelligence (AI)**.

We use detection software and make routine checks for evidence of academic misconduct. Definitions of academic misconduct, including plagiarism, self-plagiarism, and collusion can be found in Student handbook in Campus 4.0.

All cases of suspected academic misconduct are referred to for investigation, the outcomes of which can have profound consequences to your studies.

Support for Students with Disabilities

If you have a disability, long-term health condition, specific learning difference, mental health diagnosis or symptoms, contact the Student Support Office for assistance.

Unable to Submit on Time?

If events prevent you from submitting on time, guidance on extenuating circumstances is available in the Student Handbook or from the Student Support Office.

Administration of Assessment

Module Leader Name:	Hikmat Saud
Module Leader Email:	Stw0032@softwarica.edu.np
Assignment Category:	Written
Attempt Type:	Standard
Component Code:	CW

Assessment Criteria

Marking Notes

1. All submitted coursework will be assessed via VIVA conducted at the end of this semester.
2. Each VIVA will last 20 minutes.
3. You will submit on the deadline a document (PDF or Word) on c4mpus.com containing all the coursework tasks solved and including a link to your GitHub Classroom repository shared via c4mpus.com .
4. During the VIVA you will be assessed with a few relevant random questions.
5. If you submit only some of the tasks, your mark will be proportional to that.
6. The marking criteria for each sub question from 1 to 3 and 6 is presented below

Assignment Component	Max Marks	Grading			
Algorithm Design & Implementation	3	Develops a logical and efficient approach to solve the problem (algorithm design). Chosen algorithm is fully functional without any error. (Demonstrates good clarity in understanding the problem statement). . Output is not formatted. (3 Marks)	Develops a logical approach to solve the problem (algorithm design). Implements the chosen algorithm is partially functional. May have minor syntax errors or logic flaws. (2 Marks)	Partial Completeness of algorithm with syntax and logical flaws, output is not formatted. (1 Marks)	Not provided any solution. (0 Marks)
Testing & Validation	2 Marks		Decision control logic, loop logic in program exhibits proper functional behavior and Output is obtained for varying sets of input data. (2 Marks)	Decision control logic, loop logic in program lacks proper functional behavior and Output is obtained for only few sets of input data.	No test cases were implemented to verify the program's behavior with different inputs. Additionally, the program doesn't

				(1 Mark)	produce any expected output, hindering its functionality. (0 Marks)
Additional Considerations	1 Mark			Completeness of code, well-structured code, consistent variable naming and formatting, well Commented code. Code readability, clarity, and adherence to coding standards. (1 Mark)	Missing comments, program lack readability and coding standards (0 Mark)

The marking criteria valid for GUI questions 4 and question 5.

Criteria	0	1	2	3	4	5
Feature complet e (20 %)	Not submitted	Only a few features implemented and are not executing	Many of the features are implemented but are not executing correctly	Many of the features are implemented and are executed correctly	Most of the features are implemented and are executed correctly	All features implemented and are executed correctly
Code aestheti c (15%)	Not submitted	Assignment submitted but not commented and formatted. variable's/classes /function are defined but meaningless	Lack of comments, formatted in Source code. Only a few classes and functions are defined but hard to read	Lack of comments, formatted in Source code, but meaningful variable/class/ function names are used few functions are defined.	Lack of comments, formatted in Source code, but meaningful variable/class/ function names are used. Code is easy to read	Source code is well commented, properly formatted, meaningful variable/function/class names are used. Code is easy to read and understand, having many pure functions.
GUI (20%)	Not submitted	Hard to use. Only some components are used and unmanaged	Few frames are difficult to use. UI components are used but unmanaged.	Some frames are difficult to use. UI components are used but unmanaged.	Easy to use, Proper use of various UI components. User Interaction is low	Easy to use, Proper use of various UI components, Clean and interactive UI
I/P Validati on (10%)	Not submitted	Only a few input fields are validated. Error message are not shown	Only a few input fields are validated. Error messages are shown in code format	Most input fields are properly validated. Error messages are shown in code format	Most input fields are properly validated. Error messages are properly shown in natural language	All input fields are properly validated. Error messages are properly shown in natural language.
Use of require d	Not submitted	Uses an incorrect or incomplete algorithm	Only few required algorithm are well implemented	Implemented all required algorithm with few errors or missing steps.	All algorithm are implemented without any error or missing steps.	The implemented algorithms achieve optimal solutions. The program exhibits highly efficient execution of all algorithms.

algorithm (20%)						
Viva (15%)	Not present (Assignment submitted but absent in viva)	Could not explain the reasoning behind the code. But answered only one viva question	Could explain basic terms but not about algorithm. But answered only two viva question	Could explain reasoning behind the code, including use of loops, conditions, algorithms. answered only three viva question	Could explain reasoning behind the code, including use of loops, conditions, algorithms. answered only four viva question	Could explain reasoning behind the code, including use of loops, conditions, algorithms. Answered all five questions