COT4400: Analysis of Algorithms
Spring 2025 - Individual Project

Objective:

This project is designed to assess your ability to analyze complex computing problems; apply computer science theory, principles of computing, and related disciplines to identify effective solutions; demonstrate proficiency in formulating and solving engineering problems using principles of engineering, science, and mathematics; and apply software development fundamentals to create computing-based solutions. Through this project, you will strengthen essential skills in problem analysis, complexity evaluation, and algorithm optimization.

Instructions:

- Prepare a Jupyter Notebook using the provided template and complete all sections as outlined below. Ensure that each section is thorough and meets the specified requirements.
- You may use Google Colab to develop your notebook and then download it as a Jupyter Notebook (.ipynb) file for submission.
- Review the attached example template for guidance. Please note that the examples may not cover all required sections for this project — make sure to update and adapt the template as needed.
- Ensure clarity and completeness in each section, including detailed explanations and well-commented code or pseudocode.
- Parts of this project may be moved to **GradeScope**. If this happens, we will make an announcement to inform you.
- Start early! Attempting to complete the project on the due date will be extremely challenging.
- Review the example pages provided for Greedy and Divide & Conquer approaches below. They include code you need such as performance testing or plotting a graph
    - Greedy Approach: https://colab.research.google.com/drive/1z728K8h80ur7RwMAL5V9qqJ2aGuoAwXM?usp=sharing
    - *Divide & Conquer: https://colab.research.google.com/drive/1jZb6y5FiEQUmVOcFIhOzAiiuIOurbAaN?usp=sharing*
- What & How to Submit
    - Ensure that all code cells run without errors in your Colab-hosted Jupyter Notebook. After running the whole Jupyter Notebook page, download the page (.ipynb extension)
    - Name your file as "LastName-FirstName-ID.ipynb" and then submit it.
    - Make sure you submit the correct file.

Important: Failure to follow these submission instructions will result in a deduction of minimum 10 points.

Drone-based Pollution Cleanup Optimization

Problem Description:

An environmental protection agency has deployed one AI-powered drone to clean pollution hotspots scattered across a region. Each hotspot requires a specific amount of energy to clean and has a varying importance level based on environmental impact.

Hotspot Characteristics:

- Priority Score: Indicates the environmental importance of cleaning the hotspot (higher means more critical).
- Cleaning Effort: Total energy required to fully clean the hotspot (in energy units).
- Distance from Dock: Round-trip energy cost to travel from the drone's docking station to the hotspot and back (in energy units).

Drone Specifications (Per Mission):

- Battery Capacity: 1000 energy units per mission.
- Cleaning Efficiency: 1 energy unit cleans 1 unit of pollution.
- Travel Cost: Round-trip travel consumes the specified number of energy units.

Operational Rules:

- The drone starts and ends each mission at the docking station.
- It can visit multiple hotspots in a single mission.
- Total energy usage (travel + cleaning) must not exceed 1000 units.
- After cleaning a hotspot, the drone returns to the docking station to unload dirt. Every trip to another hotspot begins from the docking station.
- The drone can partially clean a hotspot, earning a priority score proportional to the fraction cleaned.
  (Example: Cleaning 50% of a hotspot with priority 100 earns 50 points.)
- Objective: Maximize the total collected priority score over one mission, which may include visits to multiple hotspots.
- Use the hotspot dataset provided at the end of this document. It is also provided in the attached Jupyter Notebook page template.

Project Tasks (100 Points Total):

a) Solution Description — Brute Force Approach (10 Points)

- Explain the brute force method to solve this problem.
- Provide clear pseudocode following class standards for structure and formatting.
- Note: You must present brute force only here, submitting optimized solutions for this part will result in point deductions.

b) Complexity Analysis - Brute Force (5 Points)

- Analyze the asymptotic time and space complexity of the brute force solution based on your pseudocode. Plot a graph demonstrating how time complexity increases as the number of hotspots grows based on the theoretical asymptotic complexity you found.
- Graphs must be labeled and explained briefly (e.g., what they show, observed trends).

c) Proof of Correctness (10 Points)

- Select a key function or loop from your brute force pseudocode.
- Prove its correctness using loop invariants, induction, or other formal methods covered in class.

d) Implementation - Brute Force (10 Points)

- Implement the brute force algorithm in Python inside a Jupyter Notebook.
- Comment code clearly and thoroughly, explaining each step and decision.
- The function should return the best total priority score and the optimal subset of hotspots that maximizes this priority. For example:

```
priority, subset = brute_max_priority(current_hotspots,battery_capacity)
```

e) Performance Testing - Brute Force (15 Points)

- Test the brute force solution on varying input sizes, starting from small sets (e.g., 2, 3, 4, 5 hotspots).
- Collect timing data for around 5 minutes and plot execution time vs. input size (10 points for performance data, 5 points for graph). Ensure you have the graph plotted with the collected data in your submission, we will **not** run it for 5 minutes.
- Graphs must be labeled and explained briefly (e.g., what they show, observed trends).

f) Optimal Algorithm - Greedy or Divide & Conquer (15 Points)

- Select and explain an optimized approach (clearly indicate whether it's Greedy or Divide & Conquer).
- Provide detailed pseudocode and explain steps:
    - If you use Greedy: Selection procedure, feasibility check, solution check.
    - If you use Divide & Conquer: Divide, conquer, combine.

g) Complexity Analysis - Optimized Algorithm (10 Points)

- Analyze asymptotic time and space complexity of your optimized solution (5 points).
- Plot time complexity graph as input size increases (5 points).
- Graphs must be labeled and explained briefly.

h) Implementation - Optimized Algorithm (15 Points)

- Implement the optimized algorithm in Python inside the same Jupyter Notebook.
- The function should return the best total priority score and the optimal subset of hotspots that maximizes this priority. For example:

```
priority, subset = greedy_max_priority(current_hotspots,battery_capacity)
```

- Comment code clearly and thoroughly, explaining each part of the process.
- Note: This part focuses only on the optimal solution implementation, not brute force.

i) Performance Testing and Comparison (10 Points)

- Run both algorithms on the same set of input sizes where feasible.
- Increase input size for optimized solution if needed to demonstrate its efficiency.
- Collect execution time data and plot comparison graph of both approaches (5 points for performance data, 5 points for graph).
- Graphs must be labeled and explained briefly (e.g., what they show, observed trends).

Important: **Ensure that all code cells execute without errors in your Colab-hosted Jupyter Notebook.** After running the entire notebook, download the **.ipynb file** along with its generated data and graphs, then submit it. **We will not run your notebook to collect data or generate graphs.**

Hotspot Dataset:

Hotspot ID	Priority Score	Cleaning Effort (Energy Units)	Distance from Dock (Round-Trip Energy Units)
1	90	40	20
2	70	30	10
3	120	80	25
4	60	20	30
5	100	50	15
6	80	60	20
7	150	70	30
8	50	25	10
9	110	55	18
10	95	45	22
11	85	35	12
12	130	90	28
13	75	40	16
14	105	60	24
15	65	30	14
16	115	70	26
17	55	20	8
18	140	85	32
19	100	50	20
20	125	75	30
21	80	50	14
22	68	38	1
23	114	74	18
24	67	27	22
25	94	50	15
26	78	70	11
27	156	77	38
28	52	19	0
29	116	64	17
30	104	52	28
31	91	35	3
32	138	98	36
33	78	30	18
34	108	50	29
35	58	39	9
36	113	78	30

37	63	21	0
38	148	83	42
39	108	43	15
40	117	70	24
41	90	48	13
42	71	39	17
43	112	90	21
44	65	28	35
45	101	50	5
46	71	57	17
47	148	78	21
48	50	24	3
49	109	52	27
50	91	49	16
51	91	35	17
52	135	87	31
53	72	34	23
54	114	50	27
55	62	38	6
56	111	69	19
57	61	24	2
58	131	92	23
59	92	56	24
60	117	85	22
61	80	36	23
62	79	36	3
63	129	71	26
64	56	16	28
65	109	59	12
66	88	53	21
67	140	72	35
68	54	24	2
69	110	50	16
70	101	41	24
71	86	35	2
72	120	86	29
73	67	45	13
74	111	70	25
75	60	27	15

76	106	74	25
77	58	23	1
78	131	78	42
79	93	50	14
80	127	72	21
81	97	40	13
82	75	20	3
83	120	81	27
84	66	25	32
85	106	45	14
86	79	70	29
87	153	77	32
88	46	31	11
89	109	54	10
90	93	49	29
91	77	32	3
92	122	85	25
93	76	49	8
94	115	52	20
95	75	36	23
96	108	70	19
97	60	16	15
98	134	92	38
99	108	51	14
100	123	65	33

hotspots = [(1, 90, 40, 20), (2, 70, 30, 10), (3, 120, 80, 25), (4, 60, 20, 30), (5, 100, 50, 15), (6, 80, 60, 20), (7, 150, 70, 30), (8, 50, 25, 10), (9, 110, 55, 18), (10, 95, 45, 22), (11, 85, 35, 12), (12, 130, 90, 28), (13, 75, 40, 16), (14, 105, 60, 24), (15, 65, 30, 14), (16, 115, 70, 26), (17, 55, 20, 8), (18, 140, 85, 32), (19, 100, 50, 20), (20, 125, 75, 30), (21, 80, 50, 14), (22, 68, 38, 1), (23, 114, 74, 18), (24, 67, 27, 22), (25, 94, 50, 15), (26, 78, 70, 11), (27, 156, 77, 38), (28, 52, 19, 0), (29, 116, 64, 17), (30, 104, 52, 28), (31, 91, 35, 3), (32, 138, 98, 36), (33, 78, 30, 18), (34, 108, 50, 29), (35, 58, 39, 9), (36, 113, 78, 30), (37, 63, 21, 0), (38, 148, 83, 42), (39, 108, 43, 15), (40, 117, 70, 24), (41, 90, 48, 13), (42, 71, 39, 17), (43, 112, 90, 21), (44, 65, 28, 35), (45, 101, 50, 5), (46, 71, 57, 17), (47, 148, 78, 21), (48, 50, 24, 3), (49, 109, 52, 27), (50, 91, 49, 16), (51, 91, 35, 17), (52, 135, 87, 31), (53, 72, 34, 23), (54, 114, 50, 27), (55, 62, 38, 6), (56, 111, 69, 19), (57, 61, 24, 2), (58, 131, 92, 23), (59, 92, 56, 24), (60, 117, 85, 22), (61, 80, 36, 23), (62, 79, 36, 3), (63, 129, 71, 26), (64, 56, 16, 28), (65, 109, 59, 12), (66, 88, 53, 21), (67, 140, 72, 35), (68, 54, 24, 2), (69, 110, 50, 16), (70, 101, 41, 24), (71, 86, 35, 2), (72, 120, 86, 29), (73, 67, 45, 13), (74, 111, 70, 25), (75, 60, 27, 15), (76, 106, 74, 25), (77, 58, 23, 1), (78, 131, 78, 42), (79, 93, 50, 14), (80, 127, 72, 21), (81, 97, 40, 13), (82, 75, 20, 3), (83, 120, 81, 27), (84, 66, 25, 32), (85, 106, 45, 14), (86, 79, 70, 29), (87, 153, 77, 32), (88, 46, 31, 11), (89, 109, 54, 10), (90, 93, 49, 29), (91, 77, 32, 3), (92, 122, 85, 25), (93, 76, 49, 8), (94, 115, 52, 20), (95, 75, 36, 23), (96, 108, 70, 19), (97, 60, 16, 15), (98, 134, 92, 38), (99, 108, 51, 14), (100, 123, 65, 33)]

Updates:

March 19th:

- Part e) Performance Testing - Brute Force text is updated for clarity.
- Hotspot dataset size increased to 100 hotspots
- Battery Capacity is now 1000
- “Operational Rules” is updated for clarity
- The following is added for clarification: “The function should return the best total priority score and the optimal subset of hotspots that maximizes this priority.” For example:

```
priority, subset = greedy_max_priority(current_hotspots,battery_capacity)
```