CISC-121 Project Guideline

Project Title: Python App for one Searching **or** one Sorting Algorithm visual simulation.
Mode: Open Book | At-Home

AI level: Up to Level 4
Submission: OnQ (GitHub repository link + Hugging Face app link)

Objective

Create a **Python app** that demonstrates **one searching *or* sorting algorithm** in a visually interactive way using a graphical user interface. Showcase your **computational thinking, algorithm design, testing, documentation skills, and knowledge of searching and sorting algorithms**.

Project Deliverables

Submit a **GitHub repository link to onQ** that includes:

File	**Description**
app.py	Your main application file
requirements.txt	List of dependencies (i.e., Python packages, e.g., Gradio)
README.md	Documentation matching marking criteria
Screenshots, demo GIF, or short video	To illustrate your app visually
Hugging Face app link	The deployed version of your app

Optional: Display the GitHub project link in your personal Queen's Student webpage (From homework 2). To showcase your portfolio of projects. And I would recommend doing so for your other projects at Queen's as well.

Required Workflow

Step 1 — Choose an Algorithm

Pick **one** algorithm to implement and visualize:

- **Searching:** Linear | Jump | Binary | Interpolation
- **Sorting:** Bubble | Selection | Insertion | Merge | Quick

Please describe *why* you chose it in your README (links to *Problem Breakdown* marks).

Step 2 — Plan Using Computational Thinking

Before coding, clearly outline:

- **Decomposition:** What smaller steps form your chosen algorithm?
- **Pattern Recognition:** How does it repeatedly reach, compare, or swap values?
- **Abstraction:** Which details of the process should be shown to the user and how to show it, and which details should be discarded (i.e., not shown)?
- **Algorithm Design:** How will input → processing → output flow to and from the user? Including the use of the graphical user interface (GUI).
- Note you are free to choose the datatypes and structures of input (e.g., integer and list or string and linked list, etc.) as long as it is explicitly stated.
- Include this plan as a short section with a flowchart diagram in your README.

Step 3 — Implement the Algorithm

Develop a working version of your chosen algorithm in Python.

- Comment key steps so the logic is easy to follow.
- Ensure the code is **readable and correct**, not just functional.
- The code structure does not have to be OOP
- Handle user input and output to the GUI carefully (e.g., incorrect entries).

Step 4 — Add Interactivity with a Python UI Library

Use a beginner friendly UI library such as **Gradio**. Another option is **Tkinter,** but that is for desktop Python apps only and cannot be deployed on Hugging Face. So, please use **Gradio**.

- Create input boxes, e.g., for target value/s and search array, and result displays.
- Show each step or the final result clearly.
- Keep the interface simple enough that anyone can understand it.

Step 5 — Test and Verify

- Try different user inputs (including edge cases).
- Check if your algorithm returns correct results.
- Record a few example runs (screenshots or results).
- Document what you tested and what worked.

(Evidence here earns "Testing & Verification" marks.)

Step 6 — Document Everything in the README

Your README.md should be written so the TAs can grade each rubric category directly.

Use the template below to align your report perfectly:

Project_README_template.md

Example markdown README Outline

```
# Algorithm Name

## Demo video/gif/screenshot of test

## Problem Breakdown & Computational Thinking (You can add a flowchart and write the four pillars of computational thinking  briefly in bullets)

## Steps to Run

## Hugging Face Link

## Author & Acknowledgment
```

Marking Rubric (Total = 15 points)

Criteria	Points	Evaluators Criteria
Problem Breakdown & Computational Thinking	3	Clear logical plan, algorithm steps, and reasoning shown in README
Algorithm Implementation	4	Correct, efficient, well-commented Python logic
Use of Python Libraries for Graphical User Interface (UI)	2	Gradio (or similar) is used effectively and is user-friendly
Testing & Verification	3	Demonstrated evidence of testing and correctness, i.e., screenshots/gifs/videos
Documentation & Report	3	Complete README, clear setup, GitHub, and Hugging Face link
Total	**15**	

Sample Demos:

1. Sorting (Bubble, Selection, Insertion, Merge, Quick, Counting, Radix) - VisuAlgo
2. Bubble Sort Game | Algorithms | Computing
3. Searching (Linear and Binary) | USFCA

Example App:

https://huggingface.co/spaces/Rahatara/SimulateSearch/blob/main/app.py

Useful Resources:

Beginners' Guide to Hugging Face and Gradio

https://medium.com/@turna.fardousi/build-share-python-apps-instantly-with-gradio-and-hugging-face-cb36d100d5e9

Tips

- Think like a teacher: your app should **help someone learn** the algorithm.
- Keep it simple and readable — focus on clarity, not complexity.
- Use comments and descriptive labels for your interface.
- Explain what the user should enter and what they will see.
- Document your logic in your blog post or README.
- Creativity is welcome while maintaining correctness. Visualization, gamification, or sound effects would improve your mark.