**Search and Sort Algorithms: Performance Comparison**

**Overview**

This repository contains two Python program structures to compare the performance of:

1. **Two Searching Algorithms**
2. **Two Sorting Algorithms**

The programs evaluate the efficiency of the chosen algorithms using three different types of arrays:

* **Ordered Array**: Elements are sorted in ascending order.
* **Reverse-Ordered Array**: Elements are sorted in descending order.
* **Unsorted Array**: Elements are in random order.

The results will include:

* **Number of swaps** for sorting algorithms.
* **Number of steps** for searching algorithms.
* A comparison of which algorithm performs better for each array type.

**Files in This Repository**

1. **search\_comparison.py**  
   A Python program structure to implement and compare two **searching algorithms**. It tracks the number of steps performed by each algorithm when searching for a target value.
2. **sort\_comparison.py**  
   A Python program structure to implement and compare two **sorting algorithms**. It tracks the number of swaps made by each algorithm when sorting the arrays.

**Task Objective**

**Search Algorithms**

* Students are required to choose **two searching algorithms** from the list below:
  + Sentinel Linear Search
  + Meta Binary Search
  + One-Sided Binary Search
  + Ternary Search
  + Jump Search
  + Interpolation Search
  + Exponential Search
  + Fibonacci Search
  + The Ubiquitous Binary Search

**Sort Algorithms**

* Students are required to choose **two sorting algorithms** from the list below:
  + **Comparison-Based**: Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Quick Sort, Heap Sort, Cycle Sort, 3-way Merge Sort.
  + **Non-Comparison Based**: Counting Sort, Radix Sort, Bucket Sort, TimSort, Comb Sort, Pigeonhole Sort.
  + **Hybrid**: IntroSort, Tim Sort.

**How to Use**

1. **Clone the Repository**:

bash

Copy code

git clone https://github.com/<your-username>/<repository-name>.git

cd <repository-name>

1. **Open the Files**:
   * search\_comparison.py contains the structure for comparing searching algorithms.
   * sort\_comparison.py contains the structure for comparing sorting algorithms.
2. **Implement Algorithms**:  
   Replace the placeholder logic with the two algorithms of your choice for both searching and sorting.
3. **Run the Programs**:  
   Execute each file to test your algorithms and generate results.

bash

Copy code

python search\_comparison.py

python sort\_comparison.py

1. **Analyze the Output**:  
   The programs will:
   * Count and display the number of **steps** for searching algorithms.
   * Count and display the number of **swaps** for sorting algorithms.
   * Compare the performance of the algorithms for all three types of arrays.

**Program Details**

**search\_comparison.py**

* **Purpose**: Compare the performance of two searching algorithms.
* **Inputs**:
  + Three arrays: Ordered, Reverse-Ordered, Unsorted.
  + A target value to search.
* **Outputs**:
  + Number of steps taken by each algorithm.
  + The better-performing algorithm for each array type.

**sort\_comparison.py**

* **Purpose**: Compare the performance of two sorting algorithms.
* **Inputs**:
  + Three arrays: Ordered, Reverse-Ordered, Unsorted.
* **Outputs**:
  + Number of swaps performed by each algorithm.
  + The better-performing algorithm for each array type.

**How to Contribute**

Feel free to fork this repository, implement your algorithms, and submit a pull request with your findings and improvements.

Presentation from the lesson: <https://www.canva.com/design/DAGW6qBdj0Q/lwhB9rMjZ9oj9oOzuE5YCg/edit?utm_content=DAGW6qBdj0Q&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton>