Lab 3: Searching algorithm

1 Exercise

1. Modify binary search to count how many times a specific element appears in a sorted array. Requirements
   - Input: A sorted array and a target value.
   - Output: The count of occurrences of the target value.
2. Implement jump search where the step size can be adjusted dynamically based on the distribution of data in the array. Requirements
   - Input: A sorted array, a target value, and an adjustable step size.
   - Output: The index of the target value or -1 if not found.
3. Modify interpolation search to work with an array of strings sorted lexicographically. Requirements
   - Input: An array of strings and a target string.
   - Output: The index of the target string or -1 if not found.
4. Given an integer array containing n distinct numbers taken from 0, 1, 2, ..., n, find the one number that is missing from this sequence using binary search techniques. Requirements
   - Input: An unsorted integer array containing n distinct numbers.
   - Output: The missing number from the sequence.
5. Implement a multi-threaded searching system where multiple searching algorithms (linear, binary, jump) can be executed concurrently on different threads. Measure and compare their performance based on execution time and number of comparisons
6. Implement a bidirectional binary search algorithm that searches for a target element from both ends of a sorted array simultaneously. Analyze the performance compared to traditional binary search.
7. Write an algorithm to find the kth smallest element in an unsorted array using a modified binary search approach, ensuring optimal time complexity.

Develop a visualization tool that graphically represents different searching algorithms (e.g., binary search, linear search) in action. Allow users to input data and visualize the steps taken during the search process.

2 Homework

1. Create a K-D tree (k-dimensional tree) for efficiently searching points in a multi-dimensional space. Implement nearest neighbor search and discuss its efficiency compared to other search methods.
2. Write an algorithm to find the closest pair of points in a 2D space using a divide-and-conquer approach. Analyze the time complexity and discuss how it improves over brute force methods.

PREPARED BY Tran Vinh Khiem

3. Implement a search algorithm that uses exponential backoff strategies to handle search retries in a distributed system. Discuss how this approach can optimize network resource usage.

4. Develop a multi-threaded implementation of the k-nearest neighbors (KNN) search algorithm for large datasets. Analyze the impact of concurrency on performance. Dataset can be found here: `https://www.kaggle.com/datasets/gkalpolukcu/knn-algorithm-dataset`

5. Develop an algorithm that can perform searches on data that is continuously being updated (e.g., real-time stock prices). Ensure it maintains efficiency and accuracy during updates. Student can access dataset here: `https://www.kaggle.com/datasets/nguyenngocphung/stock-prices-vn30-indexvietnam`

6. Implement a pattern matching algorithm that utilizes searching techniques (like KMP or Rabin-Karp) to find occurrences of a substring within a larger string efficiently.

Notice

- Use C++ for practice.
- In the programming file, the student should include the following complete information:

```
//STT: 39 (Example)
//Full Name: X, With X is you, don't need to find X anywhere else.
//Session 01 - Exercise 01
//Notes or Remarks: ......
```

- Develop a visualization tool that graphically represents different exercises in action. Allow users to input data and visualize the steps taken during the search process.

References :

[1]. Skiena, S. S. (1998). The algorithm design manual (Vol. 2). New York: springer.
[2]. Pai, G. V. (2023). A Textbook of Data Structures and Algorithms, Volume 3: Mastering Advanced Data Structures and Algorithm Design Strategies. John Wiley & Sons.
[3]. Anggoro, W. (2018). C++ Data Structures and Algorithms: Learn how to write efficient code to build scalable and robust applications in C++. Packt Publishing Ltd.
[3].Leetcode
[4].Codeforce