**Lab Sheet 8**

**Sort System**

**Introduction**

Efficient search and sort algorithms are fundamental in data structures, providing the backbone for many real-world applications. This project-based assignment focuses on creating an efficient search and sort system for an e-commerce platform. The system will handle various products, allowing users to search and sort items based on different criteria. This project will utilize advanced sorting and searching algorithms to ensure optimal performance.

**Objective**

The objective of this assignment is to develop an efficient e-commerce search and sort system using various sorting and searching algorithms. Students will implement elementary and advanced sorting algorithms, as well as search algorithms, to handle product data efficiently.

**Introduction to Data Structures and Algorithms**

1. Sorting Algorithms:
   * Elementary Sorts:
     + Insertion Sort: A simple comparison-based sorting algorithm that builds the final sorted array one item at a time.
     + Selection Sort: An in-place comparison-based sorting algorithm that divides the input list into two parts: the sublist of items already sorted and the sublist of items remaining to be sorted.
     + Bubble Sort: A simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order.
   * Advanced Sorts:
     + Merge Sort: A divide-and-conquer algorithm that divides the array into halves, sorts them, and then merges them back together.
     + Quicksort: A highly efficient sorting algorithm that works by partitioning the array into sub-arrays and then sorting them independently.
     + Heapsort: A comparison-based sorting technique based on a binary heap data structure.
   * Non-Comparison-Based Sorts:
     + Counting Sort: An integer sorting algorithm that operates by counting the number of objects that have distinct key values.
     + Radix Sort: A non-comparative integer sorting algorithm that sorts data with integer keys by grouping keys by the individual digits.
     + Bucket Sort: A sorting algorithm that distributes elements into a number of buckets, each of which is then sorted individually.
2. Searching Algorithms:
   * Sequential Search: A basic search algorithm that checks each element in the list until the desired element is found.
   * Binary Search: An efficient search algorithm that finds the position of a target value within a sorted array. It comes in two forms:
     + Recursive Binary Search: Uses a recursive approach to divide the search interval in half.
     + Iterative Binary Search: Uses an iterative approach to divide the search interval in half.

**Problem Description**

1. Product Data Management:
   * Create a product class that includes attributes such as product ID, name, category, price, and rating.
   * Implement a system to add, update, and delete products from the product list.
2. Sorting Functionality:
   * Implement the following sorting algorithms to sort products based on price, rating, and name:
     + Insertion Sort
     + Selection Sort
     + Bubble Sort
     + Merge Sort
     + Quicksort
     + Heapsort
     + Counting Sort (for integer values like price)
     + Radix Sort (for integer values like price)
     + Bucket Sort (for integer values like price)
3. Searching Functionality:
   * Implement the following searching algorithms to find products based on product ID and name:
     + Sequential Search
     + Binary Search (both recursive and iterative for product ID)
4. E-Commerce System:
   * Develop a user interface to manage and visualize the product list.
   * Provide functionalities to add, update, delete, search, and sort products.
   * Allow users to interact with the system (e.g., perform searches and sorts based on different criteria).

**Instructions**

1. Product Class Implementation:
   * Create a class Product with attributes such as product ID, name, category, price, and rating.
   * Implement methods to add, update, and delete products.

class Product {

public:

int id;

std::string name;

std::string category;

float price;

float rating;

Product(int id, std::string name, std::string category, float price, float rating)

: id(id), name(name), category(category), price(price), rating(rating) {}

};

1. Sorting Algorithm Implementation:
   * Implement each of the sorting algorithms listed above.
   * Create a method to sort the product list based on different attributes using the chosen sorting algorithm.
2. Searching Algorithm Implementation:
   * Implement each of the searching algorithms listed above.
   * Create a method to search for products based on product ID and name using the chosen searching algorithm.
3. E-Commerce System Development:
   * Develop a user interface (UI) using a framework such as Qt for C++.
   * Provide functionalities to manage and visualize the product list.
   * Allow users to perform searches and sorts interactively.

**Code:**

#include <iostream>

#include <vector>

#include <algorithm>

class Product {

public:

int id;

std::string name;

std::string category;

float price;

float rating;

Product(int id, std::string name, std::string category, float price, float rating)

: id(id), name(name), category(category), price(price), rating(rating) {}

};

// Function to display products

void displayProducts(const std::vector<Product>& products) {

for (const auto& product : products) {

std::cout << "ID: " << product.id << ", Name: " << product.name

<< ", Category: " << product.category << ", Price: " << product.price

<< ", Rating: " << product.rating << "\n";

}

}

// Sorting Algorithms

void insertionSort(std::vector<Product>& products, char criteria) {

for (int i = 1; i < products.size(); i++) {

Product key = products[i];

int j = i - 1;

while (j >= 0 && ((criteria == 'p' && products[j].price > key.price) ||

(criteria == 'r' && products[j].rating > key.rating) ||

(criteria == 'n' && products[j].name > key.name))) {

products[j + 1] = products[j];

j--;

}

products[j + 1] = key;

}

}

int partition(std::vector<Product>& products, int low, int high, char criteria) {

Product pivot = products[high];

int i = low - 1;

for (int j = low; j < high; j++) {

if ((criteria == 'p' && products[j].price <= pivot.price) ||

(criteria == 'r' && products[j].rating <= pivot.rating) ||

(criteria == 'n' && products[j].name <= pivot.name)) {

i++;

std::swap(products[i], products[j]);

}

}

std::swap(products[i + 1], products[high]);

return i + 1;

}

void quickSort(std::vector<Product>& products, int low, int high, char criteria) {

if (low < high) {

int pi = partition(products, low, high, criteria);

quickSort(products, low, pi - 1, criteria);

quickSort(products, pi + 1, high, criteria);

}

}

// Binary Search

int binarySearchById(const std::vector<Product>& products, int id, int low, int high) {

if (high >= low) {

int mid = low + (high - low) / 2;

if (products[mid].id == id)

return mid;

if (products[mid].id > id)

return binarySearchById(products, id, low, mid - 1);

return binarySearchById(products, id, mid + 1, high);

}

return -1;

}

// Sequential Search

int sequentialSearchByName(const std::vector<Product>& products, const std::string& name) {

for (int i = 0; i < products.size(); i++) {

if (products[i].name == name) return i;

}

return -1;

}

// Main menu to interact with the system

void menu() {

std::vector<Product> products;

int choice;

do {

std::cout << "\n1. Add Product\n2. Update Product\n3. Delete Product\n4. Display Products\n";

std::cout << "5. Sort Products\n6. Search Product\n0. Exit\nEnter choice: ";

std::cin >> choice;

if (choice == 1) {

int id;

std::string name, category;

float price, rating;

std::cout << "Enter Product ID: "; std::cin >> id;

std::cout << "Enter Product Name: "; std::cin >> name;

std::cout << "Enter Category: "; std::cin >> category;

std::cout << "Enter Price: "; std::cin >> price;

std::cout << "Enter Rating: "; std::cin >> rating;

products.emplace\_back(id, name, category, price, rating);

}

else if (choice == 2) {

int id;

std::cout << "Enter Product ID to update: ";

std::cin >> id;

int index = binarySearchById(products, id, 0, products.size() - 1);

if (index != -1) {

std::cout << "Enter New Name: "; std::cin >> products[index].name;

std::cout << "Enter New Category: "; std::cin >> products[index].category;

std::cout << "Enter New Price: "; std::cin >> products[index].price;

std::cout << "Enter New Rating: "; std::cin >> products[index].rating;

} else {

std::cout << "Product not found.\n";

}

}

else if (choice == 3) {

int id;

std::cout << "Enter Product ID to delete: ";

std::cin >> id;

int index = binarySearchById(products, id, 0, products.size() - 1);

if (index != -1) {

products.erase(products.begin() + index);

std::cout << "Product deleted.\n";

} else {

std::cout << "Product not found.\n";

}

}

else if (choice == 4) {

displayProducts(products);

}

else if (choice == 5) {

char criteria;

std::cout << "Choose sort criteria (p: Price, r: Rating, n: Name): ";

std::cin >> criteria;

std::cout << "Choose sort type (1: Insertion Sort, 2: Quick Sort): ";

int sortType;

std::cin >> sortType;

if (sortType == 1) {

insertionSort(products, criteria);

} else if (sortType == 2) {

quickSort(products, 0, products.size() - 1, criteria);

}

std::cout << "Products sorted.\n";

}

else if (choice == 6) {

int searchChoice;

std::cout << "Search by 1: ID, 2: Name: ";

std::cin >> searchChoice;

if (searchChoice == 1) {

int id;

std::cout << "Enter Product ID: ";

std::cin >> id;

int index = binarySearchById(products, id, 0, products.size() - 1);

if (index != -1)

std::cout << "Product Found: ID: " << products[index].id << ", Name: " << products[index].name << "\n";

else

std::cout << "Product not found.\n";

} else if (searchChoice == 2) {

std::string name;

std::cout << "Enter Product Name: ";

std::cin >> name;

int index = sequentialSearchByName(products, name);

if (index != -1)

std::cout << "Product Found: ID: " << products[index].id << ", Name: " << products[index].name << "\n";

else

std::cout << "Product not found.\n";

}

}

} while (choice != 0);

}

int main() {

menu();

return 0;

}

**Report: Implementing an Efficient E-Commerce Search and Sort System**

**Introduction**

In e-commerce platforms, efficient data retrieval and organization are essential for optimal user experience. Users should be able to search for products quickly and sort them based on various attributes such as price, rating, or category. This project-based assignment focuses on implementing efficient search and sort algorithms within an e-commerce system, leveraging data structures and algorithms to ensure quick and accurate handling of product data.

**Objective**

The main objective of this project is to develop an e-commerce search and sort system that efficiently manages and organizes product data. This system incorporates a range of sorting and searching algorithms to enable users to access and arrange product information based on different criteria.

**Overview of Data Structures and Algorithms**

**Sorting Algorithms**

This system employs both elementary and advanced sorting techniques:

1. **Elementary Sorting Algorithms**:
   * **Insertion Sort**: Builds the sorted list one element at a time, useful for small datasets or nearly sorted data.
   * **Selection Sort**: Divides the list into sorted and unsorted sections, selecting the smallest element from the unsorted part.
   * **Bubble Sort**: Repeatedly steps through the list, swapping adjacent elements if they are in the wrong order.
2. **Advanced Sorting Algorithms**:
   * **Merge Sort**: Divides the array into halves, recursively sorts each half, and merges them back.
   * **Quicksort**: Efficiently partitions the array into subarrays around a pivot and then sorts each subarray.
   * **Heapsort**: Uses a binary heap data structure to efficiently sort elements.
3. **Non-Comparison-Based Sorts**:
   * **Counting Sort**: Counts occurrences of each element for integer-based data.
   * **Radix Sort**: Sorts by individual digits, ideal for numeric data with fixed-length keys.
   * **Bucket Sort**: Distributes elements into several "buckets," each sorted individually, ideal for data uniformly distributed across a range.

**Searching Algorithms**

To retrieve product data, the system incorporates sequential and binary search methods:

1. **Sequential Search**: Checks each element in the list one by one, effective for unsorted lists or smaller datasets.
2. **Binary Search**: Efficiently finds the target in a sorted array by halving the search interval. Implemented in both:
   * **Recursive** and **Iterative** forms for flexibility and different use cases.

**Problem Description**

The system organizes product information, including a product’s ID, name, category, price, and rating, and allows users to interact with the data using various search and sort functions. The core features are:

1. **Product Data Management**:
   * A Product class is designed with attributes such as ID, name, category, price, and rating.
   * Methods are implemented to add, update, and delete products from the list.
2. **Sorting Functionality**:
   * Sorting algorithms are applied to arrange products based on attributes such as price, rating, or name.
   * The flexibility of choosing the appropriate sorting algorithm is provided to suit different data sizes and structures.
3. **Searching Functionality**:
   * Searching algorithms enable users to find products based on their ID or name.
   * The binary search is used for IDs (when the list is sorted), while sequential search is applied for names or unsorted lists.
4. **User Interaction and E-Commerce System**:
   * The user interface provides interactive options for adding, updating, deleting, sorting, and searching products.
   * This allows users to manage the product list intuitively, viewing sorted lists and accessing specific products efficiently.

**System Design**

**Product Class Design**

The Product class encapsulates essential product information such as product ID, name, category, price, and rating. This design follows object-oriented principles, enabling modularity and code reuse.

**Sorting and Searching Methods**

Each sorting and searching algorithm is implemented in a method tailored to handle specific attributes of the products. These methods ensure that products can be organized and retrieved based on user needs:

* Sorting methods are selected based on the nature of the dataset and the attribute being sorted (e.g., price or rating).
* Searching methods optimize retrieval efficiency, especially in cases with large datasets where binary search provides faster lookups for sorted lists.

**User Interface Design**

The system's UI (designed to be compatible with a framework like Qt) allows users to visualize and interact with the product list. Functionalities include:

* **Add/Update/Delete**: Basic product management, allowing users to modify the list of available products.
* **Sort**: Users can select a sorting algorithm and criteria to view products organized in a preferred order.
* **Search**: Provides both sequential and binary search options to locate products by name or ID efficiently.

**Test Cases and Expected Outputs**

Testing the system ensures the accuracy and reliability of search and sort operations. Below are some sample test cases:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Input** | **Expected Output** | **Description** |
| Insertion Sort by Price | List of products | List sorted by price | Verifies price-based sorting |
| Binary Search by ID | Product ID = 101 | Details of Product with ID 101 | Confirms ID search functionality |
| Quicksort by Rating | List of products | List sorted by rating | Validates rating-based sorting |
| Sequential Search by Name | Name = "ProductA" | Details of ProductA | Checks name-based search |
| Merge Sort by Name | List of products | List sorted by name | Ensures name-based sorting |

Each test case is designed to validate the sorting and searching algorithms' performance under different conditions and ensure the system meets expected outcomes.

**Expected Learning Outcomes**

This project provides valuable insights into data structures and algorithm implementation within a real-world context. Expected outcomes include:

1. **Understanding Sorting Algorithms**: Hands-on experience with both basic and advanced sorting algorithms and their suitability for various datasets.
2. **Efficiency in Searching**: Mastery of searching algorithms, enabling quick data retrieval.
3. **Data Management Skills**: Ability to manage and organize data effectively within an interactive system.
4. **Real-World Application of Algorithms**: Applying theoretical concepts to practical, real-world problems in the context of an e-commerce platform.

**Conclusion**

The project on implementing an efficient search and sort system for an e-commerce platform integrates theoretical knowledge with practical skills. Through various sorting and searching algorithms, it ensures quick access and organization of product data. This assignment not only provides essential algorithmic knowledge but also enhances skills in system design, data management, and UI interaction, paving the way for efficient e-commerce solutions.