2024-02-16

Assignment 1

Luke Jurkowski

1. This assignment aims to enhance online shopping platforms' backend systems by leveraging advanced data structures and sorting algorithms to efficiently manage large volumes of product data, employing fundamental structures like arrays and linked lists for data management and implementing sorting algorithms such as Bubble Sort and Insertion Sort to optimize product organization and system performance.
2. Data Load:

```cpp
vector<Product> loadProductData(const string& filename) {
    ifstream file(filename);
    vector<Product> products;

    if (file.is_open()) {
        Product product;
        while (file >> product.ID >> product.Name >> product.Price >> product.Category) {
            products.push_back(product);
        }
        file.close();
    }
    else {
        cerr << "Unable to open file " << filename << endl;
    }

    return products;
}
```

Data Load Output:

```cpp
vector<Product> products = loadProductData("product_data.txt");
```

Bubble Sort:

```cpp
void bubbleSortByPrice(vector<Product>& products) {
    for (size_t i = 0; i < products.size() - 1; ++i) {
        for (size_t j = 0; j < products.size() - i - 1; ++j) {
            if (products[j].Price > products[j + 1].Price) {
                swap(products[j], products[j + 1]);
            }
        }
    }
}
```

Bubble Sort Output:

```cpp
cout << "Sorting Products by Price..." << endl; // (Sort)
bubbleSortByPrice(products);
cout << "Products Sorted by Price:" << endl;
printAllProducts(products);
```

Print Product Details:

```cpp
void printProduct(const Product& product) {
    cout << "ID: " << product.ID << ", Name: " << product.Name << ", Price: " << product.Price << ", Category: " << product.Category << endl;
}
```

Print All products:

```cpp
cout << "All Products:" << endl;
printAllProducts(products);
cout << endl;
```

3.

Best Case:

- The best-case scenario for Bubble Sort occurs when the array is already sorted. In this case, Bubble Sort will only need to make a single pass through the array without any swaps because all elements are in the correct order.
- Time Complexity: O(n)
- Best case, Bubble Sort will only require one pass through the array, resulting in linear time complexity.

Average Case:

- The average-case scenario for Bubble Sort occurs when the array is randomly shuffled or partially sorted. In this case, Bubble Sort will need to make multiple passes through the array, swapping adjacent elements until the array is sorted.
- Time Complexity: O(n^2)
- On average, Bubble Sort will require approximately n^2/2 comparisons and swaps to sort the array, resulting in quadratic time complexity.

Worst Case:

- The worst-case scenario for Bubble Sort occurs when the array is sorted in reverse order. In this case, Bubble Sort will need to make the maximum number of passes through the array, performing swaps at each step until the array is sorted.
- Time Complexity: O(n^2)
- Worst case, Bubble Sort will require n passes through the array, and each pass will involve n comparisons and swaps, resulting in quadratic time complexity.

4. C++ script located in repo

5. In conclusion we can see that bubble sorting is effective within certain contexts due to its restraints. Overall personally, this project helped make deeper connections with understanding with C++.