**SORTING ALGORITMS**



Session: 2021 – 2025

**Submitted by:**

Muhammad Hamza 2021-CS-178

**Supervised by:**

Mam Maida

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

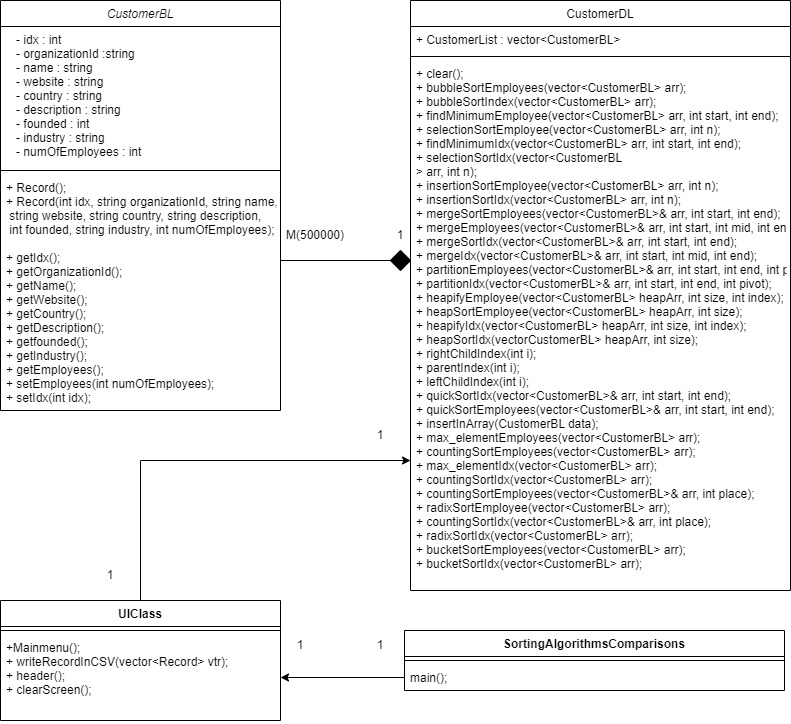
# **Table of Contents**

|  |  |
| --- | --- |
| **PAGE** | **CONTENT** |
| 3 - 3 | Short Description |
| 3 - 3 | Class Diagram |
| 4 - 4 | Wire Frame |
| 5 - 5 | Execution Time Analysis |
| 5 - 6 | Discussion on Execution Time |
| 6 - 38 | CLI Project source code |

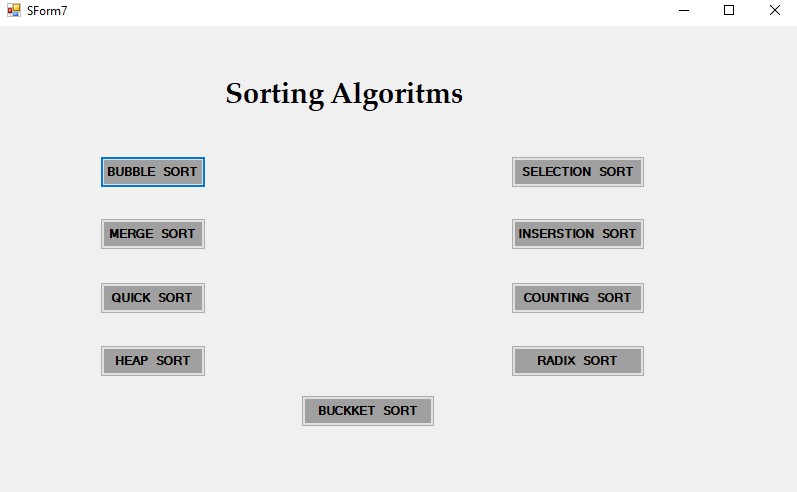
**Description:**

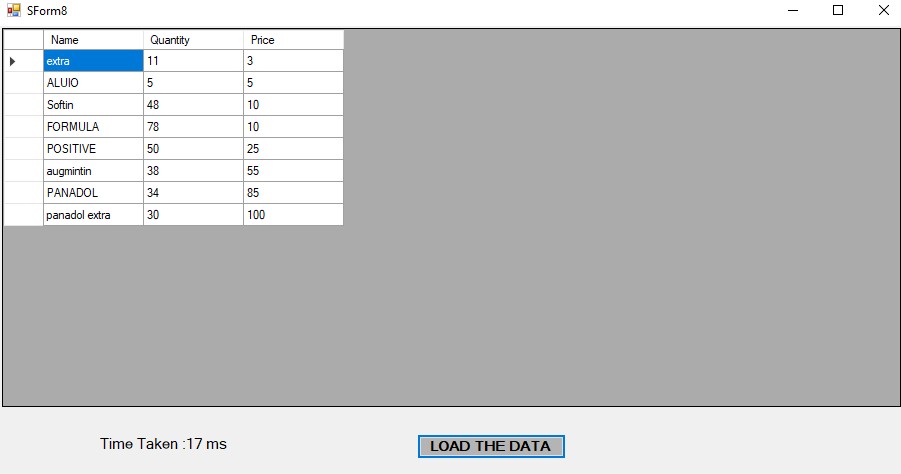
This project is basically the comparison of different Sorting Algorithms. Its demonstrate that which algorithm is best time complexity wise and also space complexity wise.

**Class Diagram:**

****

**WireFrame:**

****



**Execution Time Analysis:**

## **Unsorted Data**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Record** | **100** | **1000** | **10000** | **100000** | **500000** |
| Bubble Sort | 1.9444 ms | 183.558 ms | 20823.4 ms |  |  |
| Selection Sort | 0.004 ms | 0.0005ms | 62139355700 ns |  |  |
| Insertion Sort | 1184400 ns | 36510800 ns | 4116725800 ns |  |  |
| Merge Sort | 19579900 ns | 1040440000 ns | 126566178000 ns |  |  |
| Quick Sort | 13185000 ns | 577243000 ns | 92928978700 ns |  |  |
| Heap Sort | 911500 ns | 4332900 ns | 53924000 ns |  |  |
| Counting Sort | 1197900 ns | 4747700 ns | 40231400 ns |  |  |
| Radix Sort | 925300 ns | 8038700 ns | 147592800 ns |  |  |
| Bucket Sort | 2477100 ns | 5694800 ns | 35676200 ns |  |  |

## **Sorted Data**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Record** | **100** | **1000** | **10000** | **100000** | **500000** |
| Bubble Sort | 0.018 ms | 0.2001 ms | 3.4891 ms |  |  |
| Selection Sort | 459.761ms | 62478.8ms | 64383355400 ns |  |  |
| Insertion Sort | 1127200 ns | 37272500 ns | 5108290300 ns |  |  |
| Merge Sort | 16895900 ns | 969305000 ns | 125207280800 ns |  |  |
| Quick Sort | 9324900 ns | 557495200 ns | 89133480400 ns |  |  |
| Heap Sort | 851300 ns | 5320500 ns | 55894500 ns |  |  |
| Counting Sort | 889000 ns | 3526200 ns | 35706200 ns |  |  |
| Radix Sort | 725600 ns | 5510900 ns | 84901800 ns |  |  |
| Bucket Sort | 802300 ns | 3965400 ns | 34524400 ns |  |  |

# **Discussion on execution time:**

**Bubble sort:**

Bubble sort is quite well for sorted data and also for unsorted data but not large data because it takes a lot of comparison that’s why its time complexity in O(n^2).

**Selection sort:**

Selection sort is good for large unsorted data due to we swap the elements only in one time in the interation but for sorted is not enough good.

**Insertion sort:**

Insertion sort is good for sorted and unsorted data because its takes less comparison from the above algorithms.

**Merge Sort:**

In merge sort, data is divided into smaller parts and after sorting data is merged back. It performed better than bubble, heap and bucket sort for smaller unsorted data set as well as larger data sets.

**Quick sort:**

Quick sort is also a good sort but in this sort we hold 3 pointers and with the help of these pointers we sort the data. Quick is well for small data and for large data quick quiet less efficiency than bucket sort etc.

**Heap sort:**

Heap sort is good for large record as well for small record that why its time complexity is nlogn.

**Counting sort:**

Its performance was better than bubble, selection, heap and bucket sort for smaller unsorted data set but its performance reduced as compared to bucket sort for bigger data sets.

**Radix sort:**

Radix Sort proved to be one of the best sorting algorithms on small as well as large data sets for unsorted as well as sorted data. Although, its performance was less efficient in smaller sorted data when compared to bubble and insertion sort.

**Bucket sort:**

Bucket sort is good for large data in term of time complexity and in term of space complexity bucket sort is worst algorithm and also not good for small data compare to other sorting algorithms.

**Full Code:**

**CustomerBL:**

class CustomerBL

{

public:

int index;

int no\_of\_employ;

string country;

string description;

string founded;

string name;

string industry;

string organization;

string website;

CustomerBL(string name, int idx, string country, string organization, string despri, string founded, string industry, string web, int no\_employ)

{

this->name = name;

this->index = idx;

this->country = country;

this->description = despri;

this->industry = industry;

this->website = web;

this->no\_of\_employ = no\_employ;

this->founded = founded;

this->organization = organization;

}

}

**CustomerDL:**

CustomerDL() {

}

vector<CustomerBL> customerList;

string parse(string line, int value)

{

const char quote = '"';

int l = line.length();

string parse = "";

int count = 1, qcount = 0;

for (int i = 0; i < l; i++)

{

if (line[i] == ',')

{

if (qcount % 2 == 0)

count++;

if (line[i + 1] == quote && count == value) {

i += 2;

while (line[i] != quote) {

parse = parse + line[i];

i++;

}

return parse;

}

}

else if (count == value)

{

parse = parse + line[i];

}

if (line[i] == quote) {

qcount++;

}

}

return parse;

}

bool readFromFile(string path) {

string line, word, temp, temp2;

int index;

string id;

string name;

string website;

string country;

string description;

string founded;

string industry;

int noOfEmployees;

stringstream ss;

ifstream fin;

fin.open(path);

getline(fin, line);

while (fin.good())

{

getline(fin, line);

if (!line.empty())

{

index = stoi(parse(line, 1));

id = parse(line, 2);

name = parse(line, 3);

website = parse(line, 4);

country = parse(line, 5);

description = parse(line, 6);

founded = parse(line, 7);

industry = parse(line, 8);

noOfEmployees = stoi(parse(line, 9));

CustomerBL customer(name,index, country,id, description, founded, industry,website, noOfEmployees);

customerList.push\_back(customer);

}

else {

fin.close();

}

}

return true;

}

void bubbleSortByEmploy()

{

int n = customerList.size();

for (int x = 0; x < n - 1; x++)

{

bool isSwapped = false;

for (int y = 0; y < n - x - 1; y++)

{

if (customerList[y].no\_of\_employ > customerList[y + 1].no\_of\_employ)

{

swap(customerList[y].no\_of\_employ, customerList[y + 1].no\_of\_employ);

isSwapped = true;

}

}

if (!isSwapped)

{

break;

}

}

}

void bubbleSortByIndex()

{

int n = customerList.size();

for (int x = 0; x < n - 1; x++)

{

bool isSwapped = false;

for (int y = 0; y < n - x - 1; y++)

{

if (customerList[y].index > customerList[y + 1].index)

{

swap(customerList[y].index, customerList[y + 1].index);

isSwapped = true;

}

}

if (!isSwapped)

{

break;

}

}

}

void printArray() {

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

cout << customerList[i].index<<" ";

cout << customerList[i].organization << " ";

cout << customerList[i].name << " ";

cout << customerList[i].website << " ";

cout << customerList[i].country << " ";

cout << customerList[i].description << " ";

cout << customerList[i].founded << " ";

cout << customerList[i].industry << " ";

cout << customerList[i].no\_of\_employ << ","<<endl;

}

}

void clear() {

CustomerDL::customerList.clear();

}

void selectionSortByEmploy() {

int n = customerList.size();

for (int x = 0; x < n - 1; x++)

{

int minIndex = findMinimumByEmploy(customerList, x, n);

swap(customerList[x].no\_of\_employ, customerList[minIndex].no\_of\_employ);

}

}

int findMinimumByEmploy(vector<CustomerBL> arr, int start, int end)

{

int min = arr[start].no\_of\_employ;

int minIndex = start;

for (int x = start; x < end; x++)

{

if (min > arr[x].no\_of\_employ)

{

min = arr[x].no\_of\_employ;

minIndex = x;

}

}

return minIndex;

}

void selectionSortByIndex() {

int n = customerList.size();

for (int x = 0; x < n - 1; x++)

{

int minIndex = findMinimumByIndex(customerList, x, n);

swap(customerList[x].index, customerList[minIndex].index);

}

}

int findMinimumByIndex(vector<CustomerBL> arr, int start, int end)

{

int min = arr[start].index;

int minIndex = start;

for (int x = start; x < end; x++)

{

if (min > arr[x].index)

{

min = arr[x].index;

minIndex = x;

}

}

return minIndex;

}

void insertionSortByEmploy()

{

int n = customerList.size();

for (int x = 1; x < n; x++)

{

int key = customerList[x].no\_of\_employ;

int y = x - 1;

while (y >= 0 && customerList[y].no\_of\_employ > key)

{

customerList[y + 1].no\_of\_employ = customerList[y].no\_of\_employ;

y--;

}

customerList[y + 1].no\_of\_employ = key;

}

}

void insertionSortByIndex()

{

int n = customerList.size();

for (int x = 1; x < n; x++)

{

int key = customerList[x].index;

int y = x - 1;

while (y >= 0 && customerList[y].index > key)

{

customerList[y + 1].index = customerList[y].index;

y--;

}

customerList[y + 1].index = key;

}

}

void mergeSortByEmploy(vector<CustomerBL> &customerListss, int start,int end)

{

if (start < end)

{

int mid = (start + end) / 2;

mergeSortByEmploy(customerListss, start, mid);

mergeSortByEmploy(customerListss, mid + 1, end);

mergeByEmploy(customerListss, start, mid, end);

}

}

void mergeByEmploy(vector<CustomerBL> &arr, int start, int mid, int end) {

int i = start;

int j = mid + 1;

queue<CustomerBL> tempArr;

while (i <= mid && j <= end) {

if (arr[i].no\_of\_employ < arr[j].no\_of\_employ)

{

tempArr.push(arr[i]);

i++;

}

else

{

tempArr.push(arr[j]);

j++;

}

}

while (i <= mid) {

tempArr.push(arr[i]);

i++;

}

while (j <= end) {

tempArr.push(arr[j]);

j++;

}

for (int x = start; x <= end; x++) {

arr[x] = tempArr.front();

tempArr.pop();

}

}

void mergeSortByIndex(vector<CustomerBL>& customerListss, int start, int end)

{

if (start < end)

{

int mid = (start + end) / 2;

mergeSortByIndex(customerListss, start, mid);

mergeSortByIndex(customerListss, mid + 1, end);

mergeByIndex(customerListss, start, mid, end);

}

}

void mergeByIndex(vector<CustomerBL>& arr, int start, int mid, int end) {

int i = start;

int j = mid + 1;

queue<CustomerBL> tempArr;

while (i <= mid && j <= end) {

if (arr[i].index < arr[j].index)

{

tempArr.push(arr[i]);

i++;

}

else

{

tempArr.push(arr[j]);

j++;

}

}

while (i <= mid) {

tempArr.push(arr[i]);

i++;

}

while (j <= end) {

tempArr.push(arr[j]);

j++;

}

for (int x = start; x <= end; x++) {

arr[x] = tempArr.front();

tempArr.pop();

}

}

void writeDataIntoFile(string path) {

std::ofstream myfile;

myfile.open(path);

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

myfile << customerList[i].index << "," << customerList[i].organization << "," << customerList[i].name << "," << customerList[i].website << "," << customerList[i].country << "," << customerList[i].description << "," << customerList[i].founded << "," << customerList[i].industry << "," << customerList[i].no\_of\_employ << "\n";

}

myfile.close();

}

int partitionByEMP(int start, int end,int pivot)

{

int left = start;

int right = end;

while (left <= right) {

while (left <= end && customerList[left].no\_of\_employ < customerList[pivot].no\_of\_employ ) {

left++;

}

while (right >= start && customerList[right].no\_of\_employ > customerList[pivot].no\_of\_employ) {

right--;

}

if (left < right) {

swap(customerList[left], customerList[right]);

}

}

swap(customerList[pivot], customerList[right]);

return right;

}

int partitionByIDX(int start, int end, int pivot)

{

int left = start;

int right = end;

while (left <= right) {

while (left <= end && customerList[left].index < customerList[pivot].index) {

left++;

}

while (right >= start && customerList[right].index > customerList[pivot].index) {

right--;

}

if (left < right) {

swap(customerList[left], customerList[right]);

}

}

swap(customerList[pivot], customerList[right]);

return right;

}

void quickSortByEmploy(int start, int end)

{

if (start < end)

{

int pivot = start;

int mid = partitionByEMP(start + 1, end, pivot);

cout << endl;

quickSortByEmploy(start, mid - 1);

quickSortByEmploy(mid + 1, end);

}

}

void quickSortByIDX(int start, int end)

{

if (start < end)

{

int pivot = start;

int mid = partitionByIDX(start + 1, end, pivot);

cout << endl;

quickSortByIDX(start, mid - 1);

quickSortByIDX(mid + 1, end);

}

}

int parentIndex(int i)

{

return (i - 1) / 2;

}

int leftChildIndex(int i)

{

return (2 \* i) + 1;

}

int rightChildIndex(int i)

{

return (2 \* i) + 2;

}

void count\_sortBYEMP() {

int large = -1;

int count1 = 0;

int temp = customerList.size();

int index = 0;

int z;

vector <CustomerBL> output(temp);

for (int x = 0; x < customerList.size(); x++) {

if (customerList[x].no\_of\_employ > large) {

large = customerList[x].no\_of\_employ;

}

}

vector<int> count(large + 1);

for (int x = 0;x < customerList.size(); x++) {

for (int y = 0; y < customerList.size(); y++) {

if (customerList[x].no\_of\_employ == customerList[y].no\_of\_employ) {

count1 = count1 + 1;

count.insert(count.begin() + customerList[x].no\_of\_employ, count1);

}

}

count1 = 0;

}

for (int y = 1; y < count.size(); y++) {

count[y] = count[y] + count[y - 1];

}

for (int x = customerList.size() - 1; x > -1; x--) {

index = count[customerList[x].no\_of\_employ] - 1;

count[customerList[x].no\_of\_employ] --;

output[index] = customerList[x];

}

for (int x = 0; x < output.size(); x++) {

customerList[x] = output[x];

}

}

void count\_sortBYINDEX() {

int large = -1;

int count1 = 0;

int temp = customerList.size();

int index = 0;

int z;

vector <CustomerBL> output(temp);

for (int x = 0; x < customerList.size(); x++) {

if (customerList[x].index > large) {

large = customerList[x].index;

}

}

vector<int> count(large + 1);

for (int x = 0;x < customerList.size(); x++) {

for (int y = 0; y < customerList.size(); y++) {

if (customerList[x].index == customerList[y].index) {

count1 = count1 + 1;

count.insert(count.begin() + customerList[x].index, count1);

}

}

count1 = 0;

}

for (int y = 1; y < count.size(); y++) {

count[y] = count[y] + count[y - 1];

}

for (int x = customerList.size() - 1; x > -1; x--) {

index = count[customerList[x].index] - 1;

count[customerList[x].index] --;

output[index] = customerList[x];

}

for (int x = 0; x < output.size(); x++) {

customerList[x] = output[x];

}

}

void radix\_sortbyEMP(int place) {

int count1 = 0;

int index = 0;

int temp = customerList.size();

vector<int>count(10);

vector<CustomerBL> output(temp);

for (int x = 0;x < customerList.size(); x++) {

for (int y = 0; y < customerList.size(); y++) {

if ((customerList[x].no\_of\_employ / place) % 10 == (customerList[y].no\_of\_employ / place) % 10) {

count1 = count1 + 1;

count[(customerList[x].no\_of\_employ / place) % 10] = count1;

}

}

count1 = 0;

}

for (int y = 1; y < count.size(); y++) {

count[y] = count[y] + count[y - 1];

}

for (int x = customerList.size() - 1;x > -1; x--) {

index = count[(customerList[x].no\_of\_employ / place) % 10] - 1;

count[(customerList[x].no\_of\_employ / place) % 10]--;

output[index] = customerList[x];

}

for (int x = 0; x < output.size(); x++) {

customerList[x] = output[x];

}

}

void radix\_sortbyIndex(int place) {

int count1 = 0;

int index = 0;

int temp = customerList.size();

vector<int>count(10);

vector<CustomerBL> output(temp);

for (int x = 0;x < customerList.size(); x++) {

for (int y = 0; y < customerList.size(); y++) {

if ((customerList[x].index / place) % 10 == (customerList[y].index / place) % 10) {

count1 = count1 + 1;

count[(customerList[x].index / place) % 10] = count1;

}

}

count1 = 0;

}

for (int y = 1; y < count.size(); y++) {

count[y] = count[y] + count[y - 1];

}

for (int x = customerList.size() - 1;x > -1; x--) {

index = count[(customerList[x].index / place) % 10] - 1;

count[(customerList[x].index / place) % 10]--;

output[index] = customerList[x];

}

for (int x = 0; x < output.size(); x++) {

customerList[x] = output[x];

}

}

void heap\_sortbyIndex() {

for (int x = (customerList.size()) / 2 - 1; x >= 0; x--) {

heapifyByIndex(customerList.size(), x);

}

for (int x = customerList.size() - 1; x > 0; x--) {

swap(customerList[0], customerList[x]);

heapifyByIndex(x, 0);

}

}

void heapifyByIndex(int size, int index) {

int maxindex;

while (true) {

int leftindex = leftChildIndex(index);

int rightindex = rightChildIndex(index);

if (rightindex >= size) {

if (leftindex >= size) {

return;

}

else {

maxindex = leftindex;

}

}

else {

if (customerList[leftindex].index >= customerList[rightindex].index) {

maxindex = leftindex;

}

else {

maxindex = rightindex;

}

}

if (customerList[index].index < customerList[maxindex].index) {

swap(customerList[index], customerList[maxindex]);

index = maxindex;

}

else {

return;

}

}

}

void heap\_sortbyEMP() {

for (int x = (customerList.size()) / 2 - 1; x >= 0; x--) {

heapifyByEMP(customerList.size(), x);

}

for (int x = customerList.size() - 1; x > 0; x--) {

swap(customerList[0], customerList[x]);

heapifyByEMP(x, 0);

}

}

void heapifyByEMP(int size, int index) {

int maxindex;

while (true) {

int leftindex = leftChildIndex(index);

int rightindex = rightChildIndex(index);

if (rightindex >= size) {

if (leftindex >= size) {

return;

}

else {

maxindex = leftindex;

}

}

else {

if (customerList[leftindex].no\_of\_employ >= customerList[rightindex].no\_of\_employ) {

maxindex = leftindex;

}

else {

maxindex = rightindex;

}

}

if (customerList[index].no\_of\_employ < customerList[maxindex].no\_of\_employ) {

swap(customerList[index], customerList[maxindex]);

index = maxindex;

}

else {

return;

}

}

}

void bucketSortByEMP() {

vector<vector<float>> bucket(customerList.size());

for (int x = 0; x < customerList.size(); x++)

{

bucket[int(customerList[x].no\_of\_employ \* customerList.size())].push\_back(customerList[x].no\_of\_employ);

}

for (int x = 0; x < bucket.size(); x++)

{

sort(bucket[x].begin(), bucket[x].end());

}

int index = 0;

for (int x = 0; x < bucket.size(); x++)

{

for (int y = 0; y < bucket[x].size(); y++)

{

customerList[index].no\_of\_employ = bucket[x][y];

index++;

}

}

}

**CustomerUI:**

class CustomerUI

{

public:

static int MainMenu() {

cout << "1. BUBBLE SORT." << endl;

cout << "2. SELECTION SORT." << endl;

cout << "3. INSERTION SORT." << endl;

cout << "4. MERGE SORT." << endl;

cout << "5. QUICK SORT." << endl;

cout << "6. HEAP SORT." << endl;

cout << "7. COUNTING SORT." << endl;

cout << "8. RADIX SORT." << endl;

cout << "9. BUCKET SORT." << endl;

cout << "10. EXIT." << endl;

cout << "Enter your Option........." << endl;

int op;

cin >> op;

return op;

}

static int FirstMenu() {

cout << "1. Load 100 Records." << endl;

cout << "2. Load 1000 Records." << endl;

cout << "3. Load 10000 Records." << endl;

cout << "4. Load 100000 Records." << endl;

cout << "5. Load 500000 Records." << endl;

cout << "6. Back." << endl;

int op;

cin >> op;

return op;

}

static int SecondMenu() {

cout << "1. SORT THE DATA BY NO. OF EMPLOYS." << endl;

cout << "2. SORT THE DATA BY INDEX." << endl;

cout << "3. Back." << endl;

int op;

cin >> op;

return op;

}

**Main:**

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <sstream>

#include "CustomerBL.h"

#include "conio.h"

#include "CustomerDL.h"

#include "CustomerUI.h"

#include "chrono"

#include <cstdio>

using namespace std;

int main()

{

CustomerUI obj;

static string path;

string path1 = "Sorted-File";

CustomerDL obj1;

int count = obj1.customerList.size();

while (true)

{

int op = obj.MainMenu();

if (op == 1) {

while (true)

{

op = obj.FirstMenu();

if (op == 1) {

path = "organizations-100.csv";

auto start = chrono::steady\_clock::now();

bool find = obj1.readFromFile(path);

auto end = chrono::steady\_clock::now();

auto diff = end - start;

if (find==true) {

cout << "Data Loaded"<<endl;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

op = obj.SecondMenu();

if (op == 1) {

start = chrono::steady\_clock::now();

obj1.bubbleSortByEmploy();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

cout << "Press any Key to Print the Record" << endl;

\_getch();

start = chrono::steady\_clock::now();

obj1.printArray();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

}

else if (op == 2) {

start = chrono::steady\_clock::now();

obj1.bubbleSortByIndex();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

cout << "Press any Key to Print the Record" << endl;

\_getch();

start = chrono::steady\_clock::now();

obj1.printArray();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

}

}

}

else if (op == 2) {

auto start = chrono::steady\_clock::now();

path = "organizations-1000.csv";

bool find = obj1.readFromFile(path);

auto end = chrono::steady\_clock::now();

auto diff = end - start;

if (find == true) {

cout << "Data Loaded" << endl;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

op = obj.SecondMenu();

if (op == 1) {

start = chrono::steady\_clock::now();

obj1.bubbleSortByEmploy();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

cout << "Press any Key to Print the Record" << endl;

\_getch();

start = chrono::steady\_clock::now();

obj1.printArray();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

}

else if (op == 2) {

start = chrono::steady\_clock::now();

obj1.bubbleSortByIndex();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

cout << "Press any Key to Print the Record" << endl;

\_getch();

start = chrono::steady\_clock::now();

obj1.printArray();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

}

}

}

else if (op == 3) {

path = "organizations-10000.csv";

auto start = chrono::steady\_clock::now();

bool find = obj1.readFromFile(path);

auto end = chrono::steady\_clock::now();

auto diff = end - start;

if (find == true) {

cout << "Data Loaded" << endl;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

op = obj.SecondMenu();

if (op == 1) {

start = chrono::steady\_clock::now();

obj1.bubbleSortByEmploy();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

cout << "Press any Key to Print the Record" << endl;

\_getch();

start = chrono::steady\_clock::now();

obj1.printArray();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

}

else if (op == 2) {

start = chrono::steady\_clock::now();

obj1.bubbleSortByIndex();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

cout << "Press any Key to Print the Record" << endl;

\_getch();

start = chrono::steady\_clock::now();

obj1.printArray();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

}

}

}

else if (op == 4) {

path = "organizations-100000.csv";

auto start = chrono::steady\_clock::now();

bool find = obj1.readFromFile(path);

auto end = chrono::steady\_clock::now();

auto diff = end - start;

if (find == true) {

cout << "Data Loaded" << endl;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

op = obj.SecondMenu();

if (op == 1) {

start = chrono::steady\_clock::now();

obj1.bubbleSortByEmploy();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

cout << "Press any Key to Print the Record" << endl;

\_getch();

start = chrono::steady\_clock::now();

obj1.printArray();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

}

else if (op == 2) {

start = chrono::steady\_clock::now();

obj1.bubbleSortByIndex();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

\_getch();

start = chrono::steady\_clock::now();

obj1.printArray();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

}

}

}

else if (op == 5) {

path = "organizations-500000.csv";

auto start = chrono::steady\_clock::now();

bool find = obj1.readFromFile(path);

auto end = chrono::steady\_clock::now();

auto diff = end - start;

if (find == true) {

cout << "Data Loaded" << endl;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

op = obj.SecondMenu();

if (op == 1) {

start = chrono::steady\_clock::now();

obj1.bubbleSortByEmploy();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

cout << "Press any Key to Print the Record" << endl;

\_getch();

start = chrono::steady\_clock::now();

obj1.printArray();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

}

else if (op == 2) {

start = chrono::steady\_clock::now();

obj1.bubbleSortByIndex();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

\_getch();

start = chrono::steady\_clock::now();

obj1.printArray();

end = chrono::steady\_clock::now();

diff = end - start;

cout << chrono::duration <double, milli>(diff).count() << "ms" << endl;

}

}

}

if (op == 6) {

obj1.clear();

break;

}

obj1.writeDataIntoFile(path1);

obj1.clear();

}

}