Sorting Algorithms

Session: 2021 – 2024

Submitted by:

Muhammad Danish 2021-CS-167

Supervised by:

Mam Maida Shahid

Department of Computer Science

University of Engineering and Technology

Lahore Pakistan

Contents

Short Description of Project: 4
Class Diagram: 4
Wireframes: 5
Execution Time Analysis: 6
For Sorted Data: 6
Discussion Paragraphs: 6
- Bubble Sort 6
- Insertion Sort: 7
- Selection Sort: 7
- Merge Sort: 7
- Quick Sort: 7
- Heap Sort: 8
- Counting Sort: 8
- Radix Sort: 8
- Bucket Sort: 8

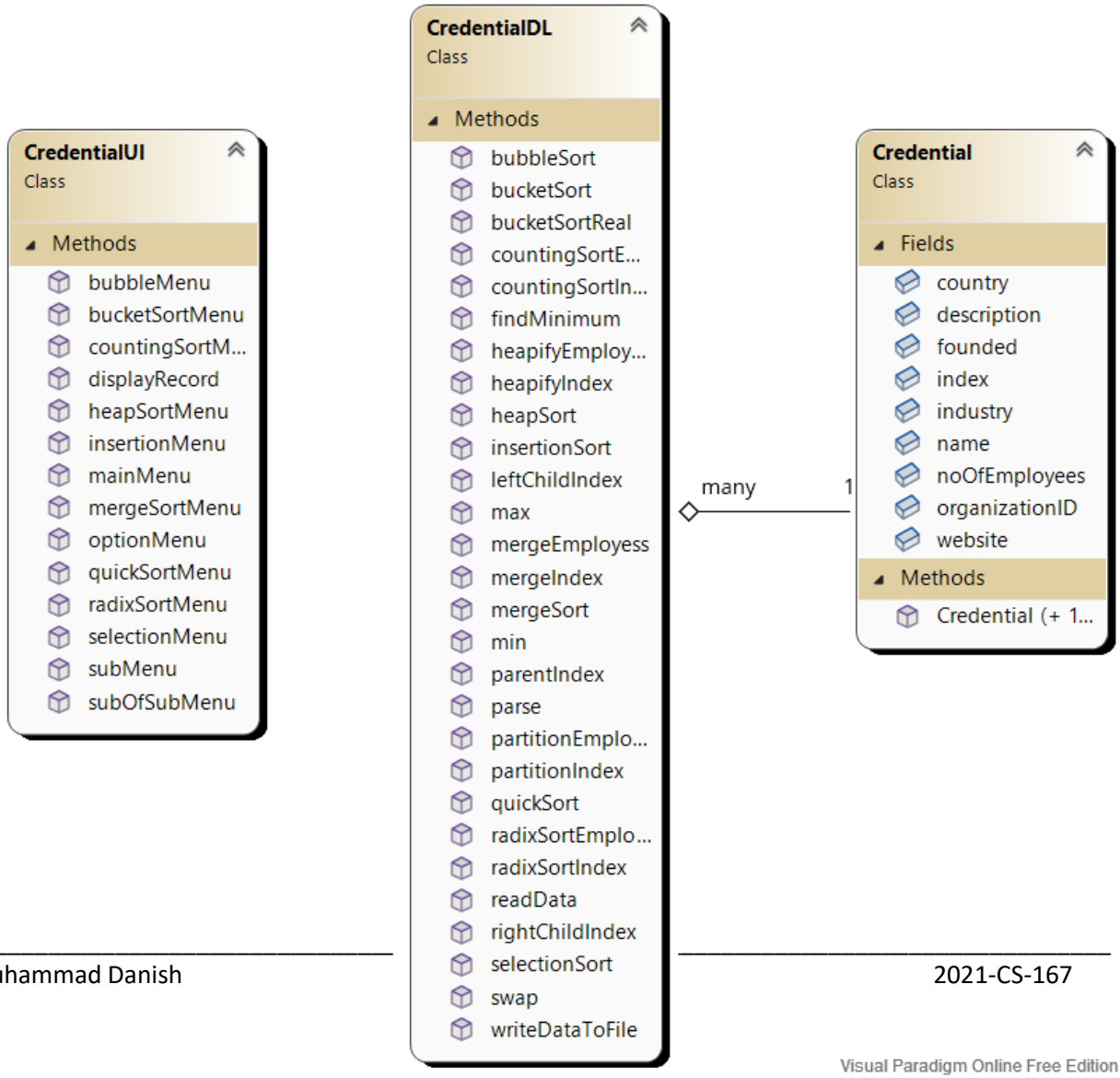
For Unsorted Data: 9
Discussion paragraphs: 9
- Bubble Sort: 9
- Insertion Sort: 9
- Selection Sort: 9
- Merge Sort: 9
- Quick Sort: 10
- Heap Sort: 10
- Counting Sort: 10
- Radix Sort: 10
- Bucket Sort: 10

Full Code of CLI Project 11
CredentialBL: 11
Credential DL 12
CredentialUI 26
Driven Program 34

Short Description of Project:

Basically, my project is all to all related to “Sorting Algorithms”. It Help us to recognized large amount of data into specific order. Sorting Algorithms take a list of data as input and return you the ordered array in specific order. Sorting plays a crucial role in all the algorithms that are related to data science. Total number of 9 Sorting Algorithms is discussed in the projects that help you to recognize its importance in different applications and solve complex problems related to sorting in easy way.

Class Diagram:



Wireframes:


Sorting
Algorithm
Hello!
welcome back!
Search here..
Muhammad Danish
ADMIN
All Oragnization Data
Sort
Load 100 Records
Load 1000 Records
Load 10000 Records
Load 100000 Record
Load 500000 Record
24859
Time Complexity(ms)
Index OrganizationID Name Website Country Description Industry Founded NoOfEmployee
1 E3ffDdFD0dbCf... Valenzuela-M... http://booker.... Ecuador Profound next ... Human Resour... 1980 7136
2 B47c7fCDAB94... "Martin Colem... https://www.m... Austria Distributed inta... Security / Inves... 2022 9082
3 32E00A5f218F4... "Waters Lozano... http://www.bol... Portugal Visionary logis... Market Research 2007 1799
4 7Ff45165a49Fd... Kerr-Hickman http://beard-g... Honduras Sharable non-... Computer Soft... 1983 9203
5 dc68feAC2D3... Montgomery LLC http://robinson... Western Sahara Exclusive dedi... Primary / Seco... 1988 1182
6 39a2534c9b6... Lowe-Cantu http://ray.biz/ Cape Verde Focused foregr... Online Publishi... 1988 7409
7 D3b16dfdbC3... "Mcconnell C... http://www.leb... Yemen Multi-layered ... Alternative Me... 1979 6701
8 25b8E298E8E1... Gilmore PLC https://riley.info/ Ecuador Streamlined m... Animation 1994 5715
9 e82c0f8bBfbC... Nolan PLC https://morale... Anguilla User-friendly in... Mechanical or... 1991 9735
10 C12FbF3B8FC... Valentine-Barton https://www.fe... Anguilla Fully-configura... Military Industry 1996 1656
11 698D69Fb7F0b... Lawson-Barker https://www.si... Gabon Balanced dida... Program Devel... 2000 9069
12 2f388fBA7ae83... "Powell Cole a... http://www.he... Togo Re-contextuali... Translation / Lo... 1974 1648
13 fbff8f06AF86d0f Decker-Edwards http://cowan.c... Kuwait Assimilated e... Railroad Manu... 2006 189
14 0E2DB5eeAEF7... Richmond Inc http://gallagh... United Kingdom Virtual fresh-thi... Sports 2018 4148
15 EC1FB946e8dB... Bonilla LLC https://richards... British Virgin Isl... Vision-oriente... Online Publishi... 1976 6299
16 e07c1d46b43E... Stanton LLC http://www.ros... Sao Tome and ... Progressive sys... Mechanical or... 2016 2111
17 4a4901C5Da1... Stevens-Zuniga https://www.o... Switzerland Balanced hum... Government A... 2012 5460
18 DCccc270d1C... Vega-Harrell http://graham-... India Fundamental ... E - Learning 1997 5132
19 CABfbbDC5aB... Rivers-Jacobs http://white.net/ Madagascar Proactive obje... Consumer Serv... 1973 5239
20 5b1f5D8091a2... Pope-Osborn http://www.gil... Argentina Cross-platform... Information Ser... 1982 5926
21 Ca59924d9e8... Valenzuela Gr... https://singh-h... Ethiopia Fundamental ... Mining / Metals 1970 9571
22 afc6B7384aD1... Fitzpatrick LLC http://www.pat... Moldova Intuitive multi-... Fine Art 1986 9467
23 CE98EC45ac5... Hubbard Inc https://hodge.... Turkey Cross-platform... E - Learning 1988 2835

Execution Time Analysis:

Basis on No of Employess	Bubble Sort	Insertion Sort	Selection Sort	Merge Sort	Quick Sort	Heap Sort	Counting Sort	Radix Sort	Bucket Sort
100 Record	0ms	0ms	1ms	0ms	0ms	0ms	0ms	oms	1ms
1000 Record	65ms	18ms	14ms	4ms	1ms	3ms	0ms	6ms	3ms
10000 Record	3206ms	3130ms	1265ms	35ms	19ms	30ms	10ms	24ms	48ms
100000 Record	17060ms	54900ms	302587ms	289ms	247ms	601ms	90ms	285ms	260ms
500000 Record	42,650,00ms	13,725,00ms	7564675ms	1768ms	1833ms	2672ms	249ms	1446ms	1406ms

Basis on No of Index	Bubble Sort	Insertion Sort	Selection Sort	Merge Sort	Quick Sort	Heap Sort	Counting Sort	Radix Sort	Bucket Sort
100 Record	0ms	0ms	1ms	0ms	0ms	0ms	0ms	0ms	0ms
1000 Record	0ms	0ms	30ms	1ms	10ms	1ms	0ms	0ms	0ms
10000 Record	1ms	2ms	1425ms	42ms	9ms	13ms	1ms	14ms	8ms
100000 Record	1ms	11ms	229227ms	295ms	151ms	301ms	40ms	143ms	42ms
500000 Record	90ms	96ms	5,730,675ms	1320ms	2085ms	1158ms	172ms	933ms	406ms

For Sorted Data:

Discussion Paragraphs:

- ## Bubble Sort

  Basically, bubble sort is straightforward, simple and slow sorting algorithm. The reason behind why it is slow is that the each element is compare to another element at basis on condition and if condition satisfies then it is swap with that element. In this way we sort in ascending and descending order according to need. But if data is already sorted then we don’t need to sort this to overcome the time complexity to O(n) we make bool type swap element and put it into require condition and set its value then after inner loop iteration if the value is not set then it means data is already sorted then we break the loop. It is efficient when comparisons are not costly.

- **Insertion Sort:**
  Basically, Insertion sort is efficient for small list of data and also for sorted data. It time complexity is also related to Bubble sort. It is efficient when comparisons are costly, because it makes less comparison to sort data as compare to other sorting algorithms because it rotates data one at a time. It also efficient when data is sorted because the require condition is not satisfy for sorted data and it takes its best time O (n) to sort data. But it's also take O(n^2) time to sort data in worst case when all to all data is not sorted.

- **Selection Sort:**
  Basically, Selection Sort improves the performance of bubble sort but also slow because in selection sort the smallest element is selected and swap against the largest element. The swapping is O(n) in selection sort as compared to bubble sort O(n^2). For Sorted Data its also take O(n^2) time to sort data because it always find minimum and then swap it with largest number in list.

- **Merge Sort:**
  Basically, Merge Sort is well for very large sorting and also stable. It is fast recursive solution to sort data according to need. It follows divide and conquer rule to sort the data and divides list until it become unity then it merge the data according to require condition and then merge data in sorted way. For Sorted data it's also take same time as for unsorted data because it follows divide and conquer rule and take (nlogn) time to sort data.

- **Quick Sort:**
  Basically, Quick sort is fastest sorting algorithm in practice but unbalance partition lead to very slow sorting. For Sorted data if you always choose first element as a pivot then it takes O(n^2) time to sort data but if we choose randomly pivot then it can overcome the time complexity of sorting but may be it also leads to O(n^2) time.

- **Heap Sort:**
  Basically, Heap Sort is most efficient version of selection sort but also slows than quick and merge sort. For Sorted Data it also take same time as for unsorted data because the worst time complexity is (nlogn) time because it also perform same procedure for sorted data first find Max then swap it with last element.

- **Counting Sort:**
  Basically, Counting Sort is useful for repeated data to sort the data and also remain stable. For Sorted Data it also performs same procedure for unsorted because it is non-comparison sorting algorithm and also stable and unstable according to output loop we traverse. It take O(N+K) time to sort data where n is total input elements and K is total number of count list element.

- **Radix Sort:**
  Basically, Radix Sort is improve version of counting sort but also use subroutine counting sorts because it does not depends on that range of integers is wide or not. For Sorted Data it also perform same procedure as for unsorted data because it's a non-comparison base sorting algorithm and it also takes O(D*(N+K)) time to sort data.

- **Bucket Sort:**
  Basically, Bucket Sort is useful to sort data decimal form of data in efficient way but also useful for integer base data because it makes bucket and put data in that buckets according to require condition and then sort the bucket according to every efficient algorithm. It does not take same time for sorted as for unsorted data because it also a non-comparison based sorting algorithm. Its Worst Case is that when all the buckets are full then it takes O(n^2) time to sort data.

For Unsorted Data:

Discussion paragraphs:

- **Bubble Sort:**
  Basically, the reason behind why it is slow is that the each element is compare to another element at basis on condition and if condition satisfies then it is swap with that element. In this way we sort in ascending and descending order according to need. It is efficient when comparisons are not costly. For unsorted data it performs same procedure as above and takes O(n^2) to sort data.

- **Insertion Sort:**
  Basically, it worst time complexity is also related to Bubble sort. It is efficient when comparisons are costly, because it makes less comparison to sort data as compare to other sorting algorithms because it rotates data one at a time. It's also take O(n^2) time to sort data in worst case when all to all data is not sorted.

- **Selection Sort:**
  Basically, in selection sort the smallest element is selected and swap against the largest element. For Unsorted Data it's also take O(n^2) time to sort data because it always find minimum and then swap it with largest number in list.

- **Merge Sort:**
  Basically, It follows divide and conquer rule to sort the data and divides list until it become unity then it merge the data according to require condition and then merge data in sorted way. For Unsorted data it's also take same time as for sorted data because it follows divide and conquer rule and take (nlogn) time to sort data.

- **Quick Sort:**
  Basically, for unsorted data if you always choose first element as a pivot then it takes O(nlogn) time to sort data but if we choose randomly pivot then it can overcome the time complexity of sorting but may be it also leads to O(nlogn) time.

- **Heap Sort:**
  Basically, for unsorted data it also take same time as for sorted data because the Best time complexity is (nlogn) time because it also perform same procedure for sorted data first find Max then swap it with last element.

- **Counting Sort:**
  Basically, for unsorted data it also performs same procedure for sorted because it is non-comparison sorting algorithm and also stable and unstable according to output loop we traverse. It take O(N+K) time to sort data where n is total input elements and K is total number of count list element.

- **Radix Sort:**
  Basically, for unsorted data it also perform same procedure as for sorted data because it's a non-comparison base sorting algorithm and it also takes O(D*(N+K)) time to sort data.

- **Bucket Sort:**
  Basically, It does not takes same time for unsorted as for sorted data because it also a non-comparison based sorting algorithm. Its Best Case is that when all the buckets are full then it takes O(N+K) time to sort data for best case.

Full Code of CLI Project

CredentialBL:

```
#pragma once
#include <iostream>
using namespace std;
class Credential
{
public:
    int index;
    int founded;
    int noOfEmployees;
    string organizationID;
    string name;
    string website;
    string country;
    string description;
    string industry;

    Credential()
    {

    }
    Credential(int index, int founded, int noOfEmployees, string organizationID, string name, string website, string
country, string description, string industry)
    {
        this->index = index;
        this->founded = founded;
        this->noOfEmployees = noOfEmployees;
        this->organizationID = organizationID;
        this->name = name;
        this->website = website;
        this->country = country;
        this->description = description;
        this->industry = industry;
    }
};
```

Credential DL

```
#include<iostream>
#include"Credential.h"
#include <vector>
#include<string>
#include<fstream>
#include"queue"
#include<cmath>
#pragma once
using namespace std;

class CredentialDL
{
 public:
   static vector<Credential> readData(string path)
   {
      vector<Credential> record;
      ifstream myFile;
      myFile.open(path,ios::in);
      string line;
      getline(myFile, line);
      while (myFile.good())
      {

         getline(myFile, line);
         if (!line.empty())
         {
            int index = stoi(parse(line, 1));
            string organizationID = parse(line, 2);
            string name = parse(line, 3);
            string website = parse(line, 4);
            string country = parse(line, 5);
            string description = parse(line, 6);
            int founded = stoi(parse(line, 7));
            string industry = parse(line, 8);
            int noOfEmployees = stoi(parse(line, 9));
            Credential cre(index, founded, noOfEmployees, organizationID, name, website, country,
description, industry);
            record.push_back(cre);

         }
         else
         {
            myFile.close();
         }
      }
```

```
    myFile.close();
    return record;
}


static string parse(string line, int find)
{
    bool flag = true;
    string parse = "";
    int commas = 1;
    int count = 0;
    for (int i = 0; line[i] != '\0'; i++)
    {
        if (line[i] == ',')
        {
            if (flag)
            {
                commas++;
            }
        }
        else if (commas == find)
        {
            parse = parse + line[i];
        }
        if (line[i] == '"')
        {
            flag = false;
            count++;
        }
        if (count == 2)
        {
            flag = true;
            count = 0;
        }

    }
    return parse;
}

static vector<Credential> bubbleSort(vector<Credential> record, bool flag)
{
    int size = record.size();
    for (int x = 0; x < size - 1; x++)
    {
        bool isSwapped = false;
        for (int y = 0; y < size - x - 1; y++)
        {
            int temp1, temp2;
```

```
            if (flag)
            {
               temp1 = record[y].index;
               temp2 = record[y + 1].index;
            }
            else
            {
               temp1 = record[y].noOfEmployees;
               temp2 = record[y + 1].noOfEmployees;
            }
            if (temp1 > temp2)
            {
               swap(record[y], record[y + 1]);
               isSwapped = true;
            }
         }
         if (!isSwapped)
         {
            break;
         }
      }
      return record;
   }

   static int findMinimum(vector<Credential> record, int start, int end, bool flag)
   {
      int min;
      if (flag)
      {
         min = record[start].index;
      }
      else
      {
         min = record[start].noOfEmployees;
      }

      int minIndex = start;
      for (int x = start; x < end; x++)
      {
         int temp;
         if (flag)
         {
            temp = record[x].index;
         }
         else
         {
            temp = record[x].noOfEmployees;
         }
```

```
        if (min > temp)
        {
            min = temp;
            minIndex = x;
        }
    }
    return minIndex;
}

static vector<Credential> selectionSort(vector<Credential> record, bool flag)
{
    int size = record.size();
    for (int x = 0; x < size - 1; x++)
    {
        int minIndex = findMinimum(record, x, size, flag);
        swap(record[x], record[minIndex]);
    }
    return record;
}

static vector<Credential> insertionSort(vector<Credential> record, bool flag)
{
    int size = record.size();
    for (int x = 1; x < size; x++)
    {
        int y, key;
        if (flag)
        {
            key = record[x].index;
            y = x - 1;
            while (y >= 0 && record[y].index > key)
            {
                record[y + 1] = record[y];
                y--;
            }
        }
        else
        {
            key = record[x].noOfEmployees;
            y = x - 1;
            while (y >= 0 && record[y].noOfEmployees > key)
            {
                record[y + 1] = record[y];
                y--;
            }
        }
        record[y + 1] = record[x];
    }
```

```
    return record;
  }
  static void mergeEmployess(vector<Credential>& arr, int start, int mid, int end) {
    int i = start;
    int j = mid + 1;
    queue<Credential> tempArr;
    while (i <= mid && j <= end) {
      if (arr[i].noOfEmployees < arr[j].noOfEmployees)
      {
        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();
    }
  }
  static void mergeIndex(vector<Credential>& arr, int start, int mid, int end) {
    int i = start;
    int j = mid + 1;
    queue<Credential> 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();
    }
}
static void mergeSort(vector<Credential>& arr, int start, int end, bool flag)
{
    if (start < end)
    {
        int mid = (start + end) / 2;
        mergeSort(arr, start, mid, flag);
        mergeSort(arr, mid + 1, end, flag);
        if (flag) {
            mergeIndex(arr, start, mid, end);
        }
        else {
            mergeEmployess(arr, start, mid, end);
        }
    }
}

static int partitionIndex(vector<Credential>& arr, int start, int end, int pivot)
{
    int left = start;
    int right = end;
    while (left <= right)
    {
        while (left <= end && arr[left].index < arr[pivot].index)
            left++;
        while (right >= start && arr[right].index >= arr[pivot].index)
            right--;
        if (left < right)
            swap(arr[left], arr[right]);
    }
    swap(arr[right], arr[pivot]);
    return right;
}

static int partitionEmployees(vector<Credential>& arr, int start, int end, int pivot)
{
    int left = start;
```

```
    int right = end;
    while (left <= right)
    {
        while (left <= end && arr[left].noOfEmployees < arr[pivot].noOfEmployees)
            left++;
        while (right >= start && arr[right].noOfEmployees >= arr[pivot].noOfEmployees)
            right--;
        if (left < right)
            swap(arr[left], arr[right]);
    }
    swap(arr[right], arr[pivot]);
    return right;
}
static void quickSort(vector<Credential>& arr, int start, int end, bool flag)
{
    if (start < end)
    {
        int pivot = start;
        int mid;
        if (flag)
        {
            mid = partitionIndex(arr, start + 1, end, pivot);
        }
        else {
            mid = partitionEmployees(arr, start + 1, end, pivot);
        }
        quickSort(arr, start, mid - 1, flag);
        quickSort(arr, mid + 1, end, flag);
    }
}

static int parentIndex(int i)
{
    return (i - 1) / 2;
}
static int leftChildIndex(int i)
{
    return (2 * i) + 1;
}
static int rightChildIndex(int i)
{
    return (2 * i) + 2;
}
static void swap(Credential& a, Credential& b)
{
    Credential temp = a;
    a = b;
    b = temp;
```

```
}

static void heapifyIndex(vector<Credential>& heapArr, int size, int index) {
    int maxIndex;
    while (true) {
        int lIdx = leftChildIndex(index);
        int rIdx = rightChildIndex(index);
        if (rIdx >= size) {
            if (lIdx >= size)
                return;
            else
                maxIndex = lIdx;
        }
        else {
            if (heapArr[lIdx].index >= heapArr[rIdx].index)
                maxIndex = lIdx;
            else
                maxIndex = rIdx;
        }
        if (heapArr[index].index < heapArr[maxIndex].index) {
            swap(heapArr[index], heapArr[maxIndex]);
            index = maxIndex;
        }
        else
            return;
    }
}

static void heapifyEmployess(vector<Credential>& heapArr, int size, int index) {
    int maxIndex;
    while (true) {
        int lIdx = leftChildIndex(index);
        int rIdx = rightChildIndex(index);
        if (rIdx >= size) {
            if (lIdx >= size)
                return;
            else
                maxIndex = lIdx;
        }
        else {
            if (heapArr[lIdx].noOfEmployees >= heapArr[rIdx].noOfEmployees)
                maxIndex = lIdx;
            else
                maxIndex = rIdx;
        }
        if (heapArr[index].noOfEmployees < heapArr[maxIndex].noOfEmployees) {
            swap(heapArr[index], heapArr[maxIndex]);
            index = maxIndex;
```

```
        }
        else
            return;
    }
  }

  static void heapSort(vector<Credential>& heapArr, int size, bool flag)
  {
    for (int x = (size / 2) - 1; x >= 0; x--)
    {
      if (flag)
      {
        heapifyIndex(heapArr, size, x);
      }
      else {
        heapifyEmployess(heapArr, size, x);
      }
    }
    for (int x = size - 1; x > 0; x--)
    {
      swap(heapArr[0], heapArr[x]);
      if (flag)
      {
        heapifyIndex(heapArr, x, 0);
      }
      else {
        heapifyEmployess(heapArr, x, 0);
      }
    }
  }
  static bool writeDataToFile(vector<Credential> record, string path)
  {
    ofstream myFile;
    myFile.open(path, ios::out);
    myFile << "index" << "," << " Organization ID" << "," << "Name" << "," << "Website" << "," <<
"Country" << "," << "Description" << "," << "Founded" << "," << "Industry" << "," << "No of Employees"
<< endl;

    for (int i = 0; i < record.size(); i++)
    {
      myFile << record[i].index << "," << record[i].organizationID << "," << record[i].name << "," <<
record[i].website << ","
        << record[i].country << "," << record[i].description << "," << record[i].founded << "," <<
record[i].industry << "," << record[i].noOfEmployees << endl;
    }
    myFile.close();
    return true;
  }
```

```
static int max(vector<Credential> arr, bool flag)
{
   int max;
   if (flag)
   {
      max = arr[0].index;
   }
   else
   {
      max = arr[0].noOfEmployees;
   }
   for (int i = 1; i < arr.size(); i++)
   {
      int value;
      if (flag)
      {
         value = arr[i].index;
      }
      else
      {
         value = arr[i].noOfEmployees;
      }
      if (max < value)
      {
         max = value;
      }
   }
   return max;
}

static int min(vector<Credential> arr, bool flag)
{
   int min;
   if (flag)
   {
      min = arr[0].index;
   }
   else
   {
      min = arr[0].noOfEmployees;
   }
   for (int i = 1; i < arr.size(); i++)
   {
      int value;
      if (flag)
      {
         value = arr[i].index;
```

```
        }
        else
        {
            value = arr[i].noOfEmployees;
        }
        if (min > value)
        {
            min = value;
        }
    }
    return min;
}


static void countingSortIndex(vector<Credential>& arr, int place, bool radix)
{
    int range = 0, maximum = 0, minimum = 0;
    if (radix)
    {
        range = 10;
    }
    else
    {
        maximum = max(arr, true);
        minimum = min(arr, true);
        range = maximum - minimum + 1;
    }

    vector<int> count(range);
    vector<Credential> output(arr.size());

    for (int i = 0; i < arr.size(); i++)
    {
        if (radix)
        {
            count[((arr[i].index / place) % 10)]++;
        }
        else
        {
            count[arr[i].index - minimum]++;
        }
    }
    for (int i = 1; i < count.size(); i++)
    {
        count[i] += count[i - 1];
    }
    for (int i = arr.size() - 1; i >= 0; i--)
    {
```

```
        if (radix)
        {
            output[count[((arr[i].index / place) % 10)] - 1] = arr[i];
            count[((arr[i].index / place) % 10)]--;
        }
        else
        {
            output[count[arr[i].index - minimum] - 1] = arr[i];
            count[arr[i].index - minimum]--;
        }
    }
    arr = output;
}

static void countingSortEmployee(vector<Credential> arr, int place, bool radix)
{
    int range = 0, maximum = 0, minimum = 0;
    if (radix)
    {
        range = 10;
    }
    else
    {
        maximum = max(arr, false);
        minimum = min(arr, false);
        range = maximum - minimum + 1;
    }
    vector<int> count(range);
    vector<Credential> output(arr.size());

    for (int i = 0; i < arr.size(); i++)
    {
        if (radix)
        {
            count[(arr[i].noOfEmployees / place) % 10]++;
        }
        else
        {
            count[arr[i].noOfEmployees - minimum]++;
        }

    }
    for (int i = 1; i < count.size(); i++)
    {
        count[i] += count[i - 1];
    }
    for (int i = arr.size() - 1; i >= 0; i--)
```

```
    {
        if (radix)
        {
            output[count[(arr[i].noOfEmployees / place) % 10] - 1] = arr[i];
            count[(arr[i].noOfEmployees / place) % 10]--;
        }
        else
        {
            output[count[arr[i].noOfEmployees - minimum] - 1] = arr[i];
            count[arr[i].noOfEmployees - minimum]--;
        }
    }
    arr = output;
}

static void radixSortIndex(vector<Credential>& arr)
{
    int maximum = max(arr, true);
    int place = 1;
    while (maximum / place > 0)
    {
        countingSortIndex(arr, place, true);
        place *= 10;
    }
}
static void radixSortEmployee(vector<Credential>& arr)
{
    int maximum = max(arr, false);
    int place = 1;
    while (maximum / place > 0)
    {
        countingSortEmployee(arr, place, true);
        place *= 10;
    }
}

static void bucketSort(vector<Credential>& arr, bool flag)
{
    int maximum;
    if (flag)
    {
        maximum = max(arr, true);
    }
    else
    {
        maximum = max(arr, false);
    }
```

```
    int n = ceil(sqrt(arr.size())) + 1;
    int indx = ceil((maximum + 1) / n);
    vector<vector<Credential>> bucket(maximum / indx + 1);

    for (int i = 0; i < arr.size(); i++) {
       int idx;
       if (flag)
       {
          idx = arr[i].index / indx;
       }
       else {
          idx = arr[i].noOfEmployees / indx;
       }
       bucket[idx].push_back(arr[i]);
    }

    for (int i = 0; i < bucket.size(); i++)
       if (flag)
       {
          countingSortIndex(bucket[i], 0, false);
       }
       else
       {
          countingSortEmployee(bucket[i], 0, false);
       }

    int index = 0;
    for (int i = 0; i < bucket.size(); i++)
       for (int j = 0; j < bucket[i].size(); j++)
          arr[index++] = bucket[i][j];
 }
 static void bucketSortReal(vector<Credential>&arr, bool flag)
 {
       int maximum;
       if (flag) {
          maximum = max(arr, true);
       }
       else {
          maximum = max(arr, false);
       }
       vector<vector<Credential>> bucket(maximum + 1);


       for (int i = 0; i < arr.size(); i++) {
          int idx;
          if (flag) {
             idx = arr[i].index;
          }
```

```
            else {
               idx = arr[i].noOfEmployees;
            }
            bucket[idx].push_back(arr[i]);
         }

         int index = 0;
         for (int i = 0; i < bucket.size(); i++)
            for (int j = 0; j < bucket[i].size(); j++)
               arr[index++] = bucket[i][j];
   }
};
```

CredentialUI

```
#include<iostream>
#include "Credential.h"
#include "SystemTime.h"
#include "CredentialDL.h"
#pragma once
using namespace std;
class CredentialUI
{
public:
   static int mainMenu()
   {
      int option;
      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 -> EXIT" << endl;
      cout << "Select one option -> ";
      cin >> option;
      return option;
   }
   static int subMenu()
   {
      int option;
      cout << "1 -> Bubble Sort Algorithm" << endl;
      cout << "2 -> Insertion Sort Algorithm" << endl;
      cout << "3 -> Selection Sort Algorithm" << endl;
      cout << "4 -> Merge Sort Algorithm" << endl;
      cout << "5 -> Quick Sort Algorithm" << endl;
      cout << "6 -> Heap Sort Algorithm" << endl;
```

```
    cout << "7 -> Counting Sort Algorithm" << endl;
    cout << "8 -> Radix Sort Algorithm" << endl;
    cout << "9 -> Bucket Sort Algorithm" << endl;
    cout << "10 -> Back" << endl;
    cout << "Select one option -> ";
    cin >> option;
    return option;
  }

  static int subOfSubMenu()
  {
    int option;
    cout << "1 -> Base On Indexes" << endl;
    cout << "2 -> Base On Number Of Employess" << endl;
    cout << "3 -> Back" << endl;
    cout << "Select one option -> ";
    cin >> option;
    return option;
  }

  static void displayRecord(vector<Credential> record)
  {
    for (int i = 0; i < record.size() - 1; i++)
    {
      cout << record[i].index << " " << record[i].organizationID << " " << record[i].name << " " <<
record[i].country << " " << record[i].website
        << " " << record[i].founded << " " << record[i].industry << " " << record[i].description << " " <<
record[i].noOfEmployees << endl;
    }
  }

  static void bubbleMenu(vector<Credential> record, vector<Credential>& sortRecord)
  {
    int subOfSubOption = 0;
    while (subOfSubOption != 3)
    {
      system("CLS");
      subOfSubOption = CredentialUI::subOfSubMenu();
      system("CLS");
      SystemTime time;
      time.BeforeOperation();
      if (subOfSubOption == 1)
      {
        sortRecord = CredentialDL::bubbleSort(record, true);
      }
      else if (subOfSubOption == 2)
      {
        sortRecord = CredentialDL::bubbleSort(record, false);
```

```
        }
        else
        {
            break;
        }
        time.AfterOperation();
        time.TimeDifference();
        system("pause");
        CredentialDL::writeDataToFile(sortRecord, "Danish.csv");
    }
}

static void insertionMenu(vector<Credential> record, vector<Credential>& sortRecord)
{
    int subOfSubOption = 0;
    while (subOfSubOption != 3)
    {
        system("CLS");
        subOfSubOption = CredentialUI::subOfSubMenu();
        system("CLS");
        SystemTime time;
        time.BeforeOperation();
        if (subOfSubOption == 1)
        {
            sortRecord = CredentialDL::insertionSort(record, true);

        }
        else if (subOfSubOption == 2)
        {
            sortRecord = CredentialDL::insertionSort(record, false);

        }
        else
        {
            break;
        }
        time.AfterOperation();
        time.TimeDifference();
        system("pause");
        CredentialDL::writeDataToFile(sortRecord, "Danish.csv");
    }

}

static void selectionMenu(vector<Credential> record, vector<Credential>& sortRecord)
{
```

```
    int subOfSubOption = 0;
    while (subOfSubOption != 3)
    {
        system("CLS");
        subOfSubOption = CredentialUI::subOfSubMenu();
        system("CLS");
        SystemTime time;
        time.BeforeOperation();
        if (subOfSubOption == 1)
        {
            sortRecord = CredentialDL::selectionSort(record, true);

        }
        else if (subOfSubOption == 2)
        {
            sortRecord = CredentialDL::selectionSort(record, false);
        }
        else
        {
            break;
        }
        time.AfterOperation();
        time.TimeDifference();
        system("pause");
        CredentialDL::writeDataToFile(sortRecord, "Danish.csv");
    }
}

static void mergeSortMenu(vector<Credential> record)
{
    int subOfSubOption = 0;
    while (subOfSubOption != 3)
    {
        system("CLS");
        subOfSubOption = CredentialUI::subOfSubMenu();
        system("CLS");
        int size = record.size() - 1;
        SystemTime time;
        time.BeforeOperation();
        if (subOfSubOption == 1)
        {
            CredentialDL::mergeSort(record, 0, size, true);

        }
        else if (subOfSubOption == 2)
        {
            CredentialDL::mergeSort(record, 0, size, false);
        }
```

```
        else
        {
            break;
        }

        time.AfterOperation();
        time.TimeDifference();
        system("pause");
        CredentialDL::writeDataToFile(record, "Danish.csv");
    }
}

static void quickSortMenu(vector<Credential> record)
{
    int subOfSubOption = 0;
    while (subOfSubOption != 3)
    {
        system("CLS");
        subOfSubOption = CredentialUI::subOfSubMenu();
        system("CLS");
        int size = record.size() - 1;
        SystemTime time;
        time.BeforeOperation();
        if (subOfSubOption == 1)
        {
            CredentialDL::quickSort(record, 0, size, true);

        }
        else if (subOfSubOption == 2)
        {
            CredentialDL::quickSort(record, 0, size, false);
        }
        else
        {
            break;
        }

        time.AfterOperation();
        time.TimeDifference();
        system("pause");
        CredentialDL::writeDataToFile(record, "Danish.csv");
    }
}

static void heapSortMenu(vector<Credential> record)
{
    int subOfSubOption = 0;
    while (subOfSubOption != 3)
```

```
    {
        system("CLS");
        subOfSubOption = CredentialUI::subOfSubMenu();
        system("CLS");
        int size = record.size() - 1;
        SystemTime time;
        time.BeforeOperation();
        if (subOfSubOption == 1)
        {
            CredentialDL::heapSort(record, size, true);

        }
        else if (subOfSubOption == 2)
        {
            CredentialDL::heapSort(record, size, false);
        }
        else
        {
            break;
        }

        time.AfterOperation();
        time.TimeDifference();
        system("pause");
        CredentialDL::writeDataToFile(record, "Danish.csv");
    }
}

static void countingSortMenu(vector<Credential> record)
{
    int subOfSubOption = 0;
    while (subOfSubOption != 3)
    {
        system("CLS");
        subOfSubOption = CredentialUI::subOfSubMenu();
        system("CLS");
        SystemTime time;
        time.BeforeOperation();
        if (subOfSubOption == 1)
        {
            system("pause");
            CredentialDL::countingSortIndex(record, 0, false);

        }
        else if (subOfSubOption == 2)
        {
            CredentialDL::countingSortEmployee(record, 0, false);
        }
```

```
        else
        {
           break;
        }

        time.AfterOperation();
        time.TimeDifference();
        system("pause");
        CredentialDL::writeDataToFile(record, "Danish.csv");
     }
  }

  static void radixSortMenu(vector<Credential> record)
  {
     int subOfSubOption = 0;
     while (subOfSubOption != 3)
     {
        system("CLS");
        subOfSubOption = CredentialUI::subOfSubMenu();
        system("CLS");
        SystemTime time;
        time.BeforeOperation();
        if (subOfSubOption == 1)
        {
           CredentialDL::radixSortIndex(record);

        }
        else if (subOfSubOption == 2)
        {
           CredentialDL::radixSortEmployee(record);
        }
        else
        {
           break;
        }

        time.AfterOperation();
        time.TimeDifference();
        system("pause");
        CredentialDL::writeDataToFile(record, "Danish.csv");
     }
  }

  static void bucketSortMenu(vector<Credential> record)
  {

     int subOfSubOption = 0;
     while (subOfSubOption != 3)
```

```
    {
        system("CLS");
        subOfSubOption = CredentialUI::subOfSubMenu();
        system("CLS");
        SystemTime time;
        time.BeforeOperation();
        if (subOfSubOption == 1)
        {
            CredentialDL::bucketSort(record,true);
        }
        else if (subOfSubOption == 2)
        {
            CredentialDL::bucketSort(record,false);
        }
        else
        {
            break;
        }
        time.AfterOperation();
        time.TimeDifference();
        system("pause");
        CredentialDL::writeDataToFile(record, "Danish.csv");
    }
}

static void optionMenu(vector<Credential> record, vector<Credential>& sortRecord)
{
    int subOption = 0;
    while (subOption != 10)
    {
        system("CLS");
        subOption = subMenu();
        if (subOption == 1)
        {
            bubbleMenu(record, sortRecord);
        }
        else if (subOption == 2)
        {
            insertionMenu(record, sortRecord);
        }
        else if (subOption == 3)
        {
            selectionMenu(record, sortRecord);
        }
        else if (subOption == 4)
        {
            sortRecord = record;
            mergeSortMenu(sortRecord);
```

```
        }
        else if (subOption == 5)
        {
           sortRecord = record;
           quickSortMenu(sortRecord);
        }
        else if (subOption == 6)
        {
           sortRecord = record;
           heapSortMenu(sortRecord);
        }
        else if (subOption == 7)
        {
           sortRecord = record;
           countingSortMenu(sortRecord);
        }
        else if (subOption == 8)
        {
           sortRecord = record;
           radixSortMenu(sortRecord);
        }
        else if (subOption == 9)
        {
           sortRecord = record;
           bucketSortMenu(record);
        }
     }
     return;
  }
};
```

Driven Program

```
#include "SortingAlgorithm.h"
#include "Credential.h"
#include"CredentialDL.h"
#include"CredentialUI.h"
#include<vector>
#include"SystemTime.h"
using namespace std;
int main()
{
   vector<Credential> record;
   vector<Credential> sortRecord;
   int option = 0;
   while (option != 5)
   {
      system("CLS");
      option = CredentialUI::mainMenu();
      if (option == 1)
```

```
        {

            record = CredentialDL::readData("organizations-100.csv");
            cout << record.size();
            CredentialUI::optionMenu(record, sortRecord);
        }
        else if (option == 2)
        {
            record = CredentialDL::readData("organizations-1000.csv");
            CredentialUI::optionMenu(record, sortRecord);
        }
        else if (option == 3)
        {
            record = CredentialDL::readData("organizations-10000.csv");
            CredentialUI::optionMenu(record, sortRecord);
        }
        else if (option == 4)
        {
            record = CredentialDL::readData("organizations-100000.csv");
            CredentialUI::optionMenu(record, sortRecord);
        }
        else if (option == 5)
        {
            record = CredentialDL::readData("organizations-500000.csv");
            CredentialUI::optionMenu(record, sortRecord);
        }
    }
};
```