|  |
| --- |
| **Computer Engineering Department - ITU** |
| **CE200L: Data Structures & Algorithms Lab** |

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated: 22/09/2022** |
| **Teaching Assistant: Muhammad Sufyan Ashraf** | **Semester: Fall 2022** |
| **Lab Engineer: Nadir Abbas** | **Batch: BSCE2021** |

# **Lab 4B. Use of Radix Sort Algorithm to Sort an Array list**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Roll number** | **Report**  **(out of 100)** | **Scaled to 10** | **Total**  **(out of 10)** |
| NIMRA MAQBOOL | BSCE21012 |  |  |  |

Checked on: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Objective**

The objective of this lab is to provide the knowledge of basic data structures and their implementations.

## **Equipment and Component**

|  |  |  |
| --- | --- | --- |
| **Component Description** | **Value** | **Quantity** |
| Computer | Available in lab | 1 |

## **Conduct of Lab**

1. Students are required to perform this experiment individually.
2. In case the lab experiment is not understood, the students are advised to seek help from the course instructor, lab engineers, assigned teaching assistants (TA) and lab attendants.

## **Theory and Background**

The variable that is used to hold the memory address of another variable is called a **pointer** variable or simply a pointer. The data type of the variable (whose address a pointer is to hold) and the pointer variable must be the same. A pointer variable is declared by placing a asterisk (\*) after data type or before variable name in data type statement. E.g. if pointer variable “p” is to hold memory address of an integer variable it is declared as:

**int \*p;**

or to hold address of a float type variable we can declare as:

**float \*rep;**

A **dynamic array** is quite similar to a regular array, but its size is modifiable during program runtime. Dynamic Array elements occupy a contiguous block of memory. Once an array has been created, its size cannot be changed. However, a dynamic array is different. A dynamic array can expand its size even after it has been filled. During the creation of an array, it is allocated a predetermined amount of memory. This is not the case with a dynamic array as it grows its memory size by a certain factor when there is a need.

**Templates** are a feature of the C++ programming language that allows functions and classes to operate with generic types. This allows a function or class to work on many different data types without being rewritten for each one.

**Radix sort** is an integer sorting algorithm that sorts data with integer keys by grouping the keys by individual digits that share the same significant position and value (place value). Radix sort uses counting sort as a subroutine to sort an array of numbers.

**Lab Task**

**Task A**

You need to continue using the code produced in last lab.

**// Add function to sort elements of the array using radix sort algorithm**

void radixSort ()

{

}

|  |
| --- |
| **FUNCTION.H:**  // // Created by Lenovo on 9/22/2022. //  #ifndef MAIN\_CPP\_FUNCTIONS\_H #define MAIN\_CPP\_FUNCTIONS\_H  #include <iostream>  using namespace std;  class mergeArray {  public:  int sizeOfArray; //declaring  int NoOfElement; //declaring  int \*array; public:   mergeArray(int s) {  sizeOfArray = s; //copying  NoOfElement = 0; //placing it to zero  array = new int[sizeOfArray];  for (int i = 0; i < sizeOfArray; i++) {  array[i] = 0; //setting values to zero  }  }   void add() {  cout << "ENTER NUMBERS IN ARRAY TO BE SORTED = ";  for (int i = 0; i < sizeOfArray; i++) {  cin >> array[i]; //taking elements in the array  }  }   void display() {  cout << "THE SORTED ARRAY =";  for (int i = 0; i < sizeOfArray; i++) {  cout << array[i] << " "; //displaying the array  }  cout << endl;  }   void merge(int \*arr, int lowerBound, int upperBound, int midTerm) {  int i = lowerBound;; //I have copied the lowerbound in i  int j = midTerm + 1; //I have copied the midTErm +1 in j  int z = lowerBound; //I have copied the lowerbound in z  int arr1[100];  while (j <= upperBound && i <=  midTerm) { //then applied an loop to check that if j and i are between the midterm and upperbound  if (arr[i] < arr[j]) { //checking which element is greater  arr1[z] = arr[i]; //putting the value  i++; //iterating i  } else {  arr1[z] = arr[j]; //putting this element  j++;  }  z++;  }  while (i <= midTerm) { //checking the i <= midterm  arr1[z] = arr[i]; //putting  z++;  i++;  }  while (j <= upperBound) { //checking that j is less than upper bound  arr1[z] = arr[j];  z++;  j++;  }  for (i = lowerBound; i < z; i++) {  arr[i] = arr1[i]; //copying  }  }   void mergeSort(int \*arr, int lowerBound, int upperBound) {  int midTerm; //declaring  if (lowerBound < upperBound) { //checking  midTerm = (lowerBound + upperBound) / 2; //calculating the mid of the array  mergeSort(arr, lowerBound, midTerm); //calling the function  mergeSort(arr, midTerm + 1, upperBound); //calling the function itself  merge(arr, lowerBound, upperBound, midTerm); //calling  }  }   int getMax(int arr[], int size1) {  int max;  max = arr[0];  for (int i = 0; i < size1; i++) {  if (max < arr[i]) {  max = arr[i]; //finding the max element  }  }  cout << "\nTHE MAX ELEMENT IN THE ARRAY = " << max << endl;  return max;  }   void countSort(int arr[], int size1) {   cout << "ENTER SIZE = ";  cin >> size1; //taking size  cout << "ENTER ELEMENTS = ";  for (int i = 0; i < size1; i++) {  cin >> arr[i]; //taking input from the user  }   cout << "THE ARRAY BEFORE SORTING = ";  for (int i = 0; i < size1; i++) {  cout << arr[i] << " "; //displaying  }  int max = getMax(arr, size1);   int count[max];  for (int i = 0; i <= max; ++i) {  count[i] = 0; //putting it equal to zero  }  for (int i = 0; i < size1; i++) {  count[arr[i]]++; //counting the no. of elements at that index  }  int j = 0; //incrementer for arr array in which the input is taken  int i = 0; //incrementer for count array  while (i <= max) { //to check that i is less than max  if (count[i] > 0) { //checking that the count is greater than zero  arr[j] = i; //putting the value at i  count[i] = count[i] - 1; //decrementing the counter at that index  j++;  } else {  i++; //if count is less than zero then increment the i and check the next index  }  }  cout << "THE ARRAY AFTER SORTING = ";  for (int i = 0; i < size1; i++) {  cout << arr[i] << " "; //displaying  }  cout << endl;   }   void count(int arr[], int size1, int i) {  int output[size1]; //declaring an array to store  int count[10] = {0}; //declaring array to count  int k = 0;  while (k < size1) {  ++count[(arr[k] / i) % 10]; //we are taking the mode to check the least element  k++;  }  int j=1;  while (j < 10) {  count[j] += count[j - 1]; //counting in the count array  j++;  }  for (int j = size1 - 1; j >= 0; j--) {  output[count[(arr[j] / i) % 10] - 1] = arr[j];  count[(arr[j] / i) % 10]--;  }  for (int j = 0; j < size1; j++) {  arr[j] = output[j]; //copying the output array in the original array  }  }   void radixSort(int arr[], int size1) {  int max = getMax(arr, size1); //calling the max function and storing in the max element  int i=1; //declaring  while(max/i>0){  count(arr, size1, i); //calling the count function  i\*=10; //to go to the previous digit  }  } };  **MAIN.CPP:**  // // Created by Lenovo on 9/22/2022. //  // // Created by Lenovo on 9/20/2022. // #include <iostream> #include "Functions.h"  using namespace std;  // int main() {   int opt;  cout << "PLEASE ENTER YOUR CHOICE." << endl; //displaying options  cout << "1.MERGE SORT." << endl;  cout << "2.COUNT SORT." << endl;  cout << "3.RADIX SORT." << endl;  cout << "4.EXIT." << endl;  cin >> opt;   if (opt == 1) {  mergeArray M(5);  int size; //declaring  int \*array1 = new int[size]; //making an  cout << "ENTER SIZE OF ARRAY = ";  cin >> size; //taking size of array  cout << "ENTER ELEMENTS OF ARRAY = ";  for (int i = 0; i < size; i++) {  cin >> array1[i]; //taking input  }  M.mergeSort(array1, 0, size - 1); //calling function  cout << "SORTED ARRAY AFTER MERGE SORT = ";  for (int i = 0; i < size; i++) {  cout << array1[i] << " "; //displaying  }  cout << endl << endl;  }  if (opt == 2) {  mergeArray M(5);  int size1;  int arr[size1];  M.countSort(arr, size1); //calling the function  }  if (opt == 3) {  mergeArray M(5);  int size1; //declaring  cout<<"ENTER SIZE OF ARRAY = ";  cin>>size1; //taking size  int arr[size1];  cout<<"ENTER ELEMENTS = ";  for(int j=0;j<size1;j++){  cin>>arr[j]; //taking input in the array  }  cout<<endl;  cout<<"ARRAY BEFORE SWAPPING = ";  for(int j=0;j<size1;j++){  cout<<arr[j]<<" "; //showing the elements of the array before swapping  }  cout<<endl;  M.radixSort(arr,size1); //calling function  cout<<"ARRAY AFTER SWAPPING = ";  for(int j=0;j<size1;j++){  cout<<arr[j]<<" "; //showing the elements of the array after swapping  }  cout<<endl;  }  if (opt == 4) {  cout << "YOU CHOOSE TO EXIT..." << endl;  exit(2);  }  return 0; }  **OUTPUT:** |

#### **Assessment Rubric for Lab**

**Method for assessment:**

Lab reports and instructor observation during lab sessions. Outcome assessed:

a. Ability to conduct experiments, as well as to analyze and interpret data (P) b. Ability to function on multi-disciplinary teams (A)

c. Ability to use the techniques, skills, and modern engineering tools necessary for engineering practice (P)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Performance metric** | **Task** | **CLO** | **Description** | **Max marks** | **Exceeds expectation** | **Meets expectation** | **Does not meet expectation** | **Obtained marks** |
| 1. Realization of experiment (a) | 1 | 1 | Functionality | 40 | Executes without errors excellent user prompts, good use of symbols, spacing in output. Through testing has been completed (35-40) | Executes without errors, user prompts are understandable, minimum use of symbols or spacing in output. Some testing has been completed (20-34) | Does not execute due to syntax errors, runtime errors, user prompts are misleading or non-existent. No testing has been completed (0-19) |  |
| 2. Teamwork (b) | 1 | 3 | Group Performance | 5 | Actively engages and cooperates with other group member(s) in effective manner (4-5) | Cooperates with other group member(s) in a reasonable manner but conduct can be improved (2-3) | Distracts or discourages other group members from conducting the experiment (0-1) |  |
| 3. Conducting experiment (a, c) | 1 | 1 | On Spot Changes | 10 | Able to make changes (8-10) | Partially able to make changes (5-7) | Unable to make changes (0-4) |  |
| 1 | 1 | Viva | 10 | Answered all questions (8-10) | Few incorrect answers (5-7) | Unable to answer all questions (0-4) |  |
| 4. Laboratory safety and disciplinary rules (a) | 1 | 3 | Code commenting | 5 | Comments are added and does help the reader to understand the code (4-5) | Comments are added and does not help the reader to understand the code (2-3) | Comments are not added (0-1) |  |
| 5. Data collection (c) | 1 | 3 | Code Structure | 5 | Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap (4-5) | Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables (2-3) | Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy (0-1) |  |
| 6. Data analysis (a, c) | 1 | 4 | Algorithm | 20 | Solution is efficient, easy to understand, and maintain (15-20) | A logical solution that is easy to follow but it is not the most efficient (6-14) | A difficult and inefficient solution (0-5) |  |
| 7. Computer use (c) | 1 | 2 | Documentation & Github Submissions | 5 | Timely (4-5) | Late (2-3) | Not done (0-1) |  |
|  | Max Marks (total): | | | 100 | Obtained Marks (total): | | |  |

Lab Engineer Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_