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

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated: 15/09/2022** |
| **Teaching Assistant: Muhammad Sufyan Ashraf** | **Semester: Fall 2022** |
| **Lab Engineer: Nadir Abbas** | **Batch: BSCE2021** |

# **Lab 3B. Use of Quick Sort Algorithm to Sort an Array list**

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

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.

**Quicksort** is a popular sorting algorithm that is often faster in practice compared to other sorting algorithms. It utilizes a divide-and-conquer strategy to quickly sort data items by dividing a large array into two smaller arrays.

**Lab Task**

**Task A**

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

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

void quickSort ()

{

}

|  |
| --- |
| **FUNCTION.H:**  // // Created by Lenovo on 9/16/2022. //  #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 bubbleSort() {  int temp; //declaring temp  for (int i = 0; i < sizeOfArray - 1; i++) { //applying loop till size-1  for (int j = i + 1;  j < sizeOfArray; j++) { //applying loop starting from 1 index after till size  if (array[j] < array[i]) { //comparing the indexes  temp = array[j]; //swapping the elements if the next index is greater than the previous one  array[j] = array[i];  array[i] = temp;  }  }  }  }   void insertionSort() {  int temp;  for (int i = 1;  i < sizeOfArray; i++) { //applying loop till the size starting from 1  for (int j = i; j >  0; j--) { //applying inner loop starting from i to greater than zero  if (array[j - 1] > array[j]) { //comparing  temp = array[j - 1]; //swapping  array[j - 1] = array[j];  array[j] = temp;  }  }  }  }   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 takingPivotPointAndSeparating(int array2[], int lowerBound, int upperBound) {  //selecting the upperbound element as pivot value  int pivotPoint = array2[upperBound];  int temp; //declaring elements fro swapping  int temp1;  int i = (lowerBound - 1); //declaring i as lower-bound -1  int j = lowerBound;  while (j < upperBound) { //applying condition  if (array2[j] <= pivotPoint) { //checking the values (comparing values at indexes  i++; //if pivot elements is greater than swap the elements  temp1=array2[i]; //swapping the element having i index with the element having j  array2[i]=array2[j];  array2[j]=temp1;  }  j++; //iterating j  }   temp=array2[i+1];  array2[i+1]=array2[upperBound]; //swapping the pivot point with the greater element at i  array2[upperBound]=temp;  // return the separating point  return (i + 1);  }   void quickSort(int array2[], int lowerBound, int upperBound) {  if (lowerBound < upperBound) {  //we are finding the pivot point and dragging the smaller value on the left side and greater value on the right side  int pivotP = takingPivotPointAndSeparating(array2, lowerBound, upperBound);   //calling function for the left side of pivot point  quickSort(array2, lowerBound, pivotP - 1);   //calling function for the right side of pivot point  quickSort(array2, pivotP + 1, upperBound);  }  } };  **MAIN.CPP:**  #include <iostream> #include "Functions.h"  using namespace std;  int main() {  int opt;  cout << "PLEASE ENTER YOUR CHOICE." << endl;  cout << "1.BUBBLE SORT." << endl;  cout << "2.INSERTION SORT." << endl; //displaying options  cout << "3.MERGE SORT." << endl;  cout << "4.QUICK SORT." << endl;  cout << "5.EXIT." << endl;  cin >> opt;  if (opt == 1) {  mergeArray M(5); //making an object  M.add();  M.bubbleSort(); //calling  M.display();  }  if (opt == 2) {  mergeArray M(5); //making an object  M.add();  M.insertionSort();  M.display();  }  if (opt == 3) {  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==4){  mergeArray M(5);  int size1; //declaring  int \*array1 = new int[size1]; //making an  cout << "ENTER SIZE OF ARRAY = ";  cin >> size1; //taking size of array  cout << "ENTER ELEMENTS OF ARRAY = ";  for (int i = 0; i < size1; i++) {  cin >> array1[i]; //taking input  }  M.quickSort(array1,0,size1-1);  cout << "SORTED ARRAY AFTER MERGE SORT = ";  for (int i = 0; i < size1; i++) {  cout << array1[i] << " "; //displaying  }  cout << endl << endl;  }  if (opt == 5) {  cout << "YOU CHOOSE TO EXIT..." << endl;  exit(2);  }  } |

#### **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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_