36.

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

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated: 08/09/2022** |
| **Teaching Assistant: Muhammad Sufyan Ashraf** | **Semester: Fall 2022** |
| **Lab Engineer: Nadir Abbas** | **Batch: BSCE2021** |

# **Lab 2B. Comparison among Bubble Sort, Selection Sort, and Insertion Sort**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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.

A **Sorting** Algorithm is used to rearrange a given array or list elements according to a comparison operator on the elements. **Bubble** sort repeatedly compares and swaps(if needed) adjacent elements in every pass. **Insertion** Sort is a simple comparison-based sorting algorithm. It inserts every array element into its proper position. **Selection** sort selects i-th smallest element and places at i-th position. This algorithm divides the array into two parts: sorted (left) and unsorted (right) subarray. It selects the smallest element from unsorted subarray and places in the first position of that subarray (ascending order). It repeatedly selects the next smallest element.

**Lab Task**

**Task A**

Design a class to keep a list of similar data in the form of an array, the data type of array will be taken from templates.

**Data Members:**

// Add variable for array, no of elements in the list, and capacity

**Member Functions:**

**//Add function to expand an array by increasing size by 10**

void expand(int)

{

}

**// Add function to contract an array by deleting value from last index**

void deleteFromLast()

{

}

**// Add function to contract an array by deleting value from any index**

void deleteFromIndex(int)

{

}

**// Add function to add element at start index of the array**

void prepend(T)

{

}

**// Add function to add element at last index of the array**

void append(T)

{

}

**// Add function to add element at specific index in array**

void addAtIndex(T)

{

}

**// Add function to sort elements of the array**

void sortArray()

{

}

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

void bubbleSort ()

{

}

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

void insertionSort ()

{

}

|  |
| --- |
| **FUNCTION.H:**  // // Created by Lenovo on 9/8/2022. //  #ifndef ITU\_ALL\_DATA\_FUNCTIONS\_H #define ITU\_ALL\_DATA\_FUNCTIONS\_H #include <iostream>  using namespace std;  class sortArrays { //making class private:  int sizeOfArray; //declaring  int NoOfElement; //declaring  int \*array; public:  sortArrays(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 insertionSort(){  Int temp;  For(int i=1;i<size;i++}{  Temp=arr[i];  J=i-1;  While(j>=0 && arr[j]>temp){  arr[j+1]=arr[j];  j--;  }  Arr[j+1]=temp; }};  **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;  cout<<"3.EXIT."<<endl;  cin>>opt;  if(opt==1){  sortArrays S (5); //making an object  S.add();  S.bubbleSort(); //calling  S.display();  }  if(opt==2){  sortArrays S (5);  S.add();  S.insertionSort();  S.display();  }  if(opt==3){  cout<<"YOU CHOOSE TO EXIT..."<<endl;  exit(2);  } } |

**Task B**

Find time complexity of bubble sort, selection sort, and insertion sort algorithms and write its advantages.

|  |
| --- |
| **BUBBLE SORT:**  int temp;  for (int i = 0; i < sizeOfArray - 1; i++) {   for (int j = i + 1; j < sizeOfArray; j++) {   if (array[j] < array[i]) {   temp = array[j];  array[j] = array[i];  array[i] = temp;  }  } }  **COST REPITITION TOTAL**  **1 1 1**  **1 1 1**  **1 n-1 n-1**  **1 n n**  **1 n\*n n\*n**  **1 n\*n n\*n**  **So big-O is n^2 =O(n^2)**  **Advantages:**  **In best case the time complexity is o(n).**  **Otherwise, in average case is o(n^2), furthermore it requires little memory and is easy to use and can swap the 2 numbers when needed.**  **INSERTION SORT:**  for(int i=1;i<sizeOfArray;i++){ for(int j=i;j>0;j--){ if(array[j-1]>array[j]){   temp=array[j-1];   array[j-1]=array[j];  array[j]=temp;  }  } }  **COST REPITITION TOTAL**  **1 1 1**  1 n-1 n-1  1 n n  1 n n  1 n\*n n\*n  1 n\*n n\*n  So big-o is n^2 = O(n^2).  **Advantages:**  **The best-case time complexity is o(n) while the average case is o(n^2). And it is efficient for small data sets and is simple to implement too.** |

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