University of Burgundy

Software Engineering

Tutorial 5

Lab Report-5

Author:
PARMAR Hardiksinh

Supervisor:
Dr.Yohan Fougerolle

November 25, 2018

1 Sorting Algorithms:: header file

```
#ifndef TUT5_H
#define TUT5_H
class Node
{
public:
    int Data;
    Node *left;
    Node *right;
};


class CArray
{
public:
    // 1. Preliminary work}

    int* Arr;
    unsigned int noelements;

    CArray();
    CArray(unsigned int);

    // Randomly initializes the array between 10-20 elements
    void Builder();
    // Randomly initializes the array with given size
    void Builder(unsigned int);

    // To Display the array
    void Display() const;

    // 2. A First and Simple Algorithm: Bubble Sort
    void Swap(unsigned int, unsigned int);
    void BubbleSort();

    // 3. Quicksort
    void Recursively_Sort(int*, unsigned int, unsigned int);
    int Recursive_Sort_Partition(int*, unsigned int, unsigned int);
    void QuickSort();

    // 4. Selection Sort
    void SelectionSort();

    // 5. Insertion Sort
    void InsertionSort();

    // 6. Sort using binary trees
    Node *CreateNode(int);
    void Store_value(Node *, int *, int &);
    Node *Insert_value(Node *, int );
    void BinarySort();


};

#endif // TUT5_H
```

2 Sorting Algorithms:: source file

```
#include "tut5.h"
#include <iostream>

using namespace std;


//------------------ 1. Preliminary work -----------------

// Build Default constructor
CArray::CArray()
{
    this -> Builder();
}

// Build Parameterized constructor
CArray::CArray(unsigned int noelements)
{
    this -> Builder(noelements);
}

// Initializes the array
void CArray::Builder()
{
    int a;
    cout << "Enter the Number of Element " << endl;
    cin >> a;


    this -> noelements = a;

    this -> Arr = new int [this -> noelements];

    for(unsigned int i = 0; i < this -> noelements; i++)
    {
        this -> Arr[i] = rand() % 10;
        // Randomly assign values for array
    }
}

// Initializes the size of the Array which is given by user
void CArray::Builder(unsigned int noelements)
{
    this -> noelements = noelements;

    this -> Arr = new int[noelements];

    for(unsigned int i = 0; i < this -> noelements; i++)
    {
        this -> Arr[i] = rand() % 10;
    }
}

// Display the Array
void CArray::Display() const
{
```

```
    cout << "Values Stored in the Array:" << endl;


    for(unsigned int i = 0; i < this -> noelements; i++)
    {
        cout << this -> Arr[i] << " ";
    }

    cout << endl;

    for(unsigned int i = 0; i < this->noelements; i++)
    {
    cout << "Index is " << i << " & Value is " <<  this -> Arr[i]
    << endl;
    }
    cout << endl;
}



//-------2. A First and Simple algorithm: Bubble Sort --------
// On each pass, bubble sort scans the array, comparing
each pair of adjacent elements.
// If two adjacent elements are out of order, they are swapped.
// As long as at least one swap is performed along the scan,
another pass is computed
// First Swaps the values of the Array with Index
void CArray::Swap(unsigned int Index1, unsigned int Index2)
{
    int Bubble = 0;
    Bubble = this -> Arr [Index1];
    this -> Arr[Index1] = this -> Arr[Index2];
    this -> Arr[Index2] = Bubble;
}

// Perform Bubble sort
void CArray::BubbleSort()
{
    for(unsigned int i = 0; i < this -> noelements; i++)
    {
        for(unsigned int j = i + 1; j < this -> noelements; j++)
        {
            if(this -> Arr[i] > this -> Arr[j])
                this -> Swap(i, j);
        }
    }
}

//---------- 3. Quicksort --------------------

// Quicksort works in a "divide and conquer" manner
// split the initial list of numbers into parts around a "pivot";
// all the values in the first part are less than the pivot;
// all the values in the second part are greater than or equal
to the pivot.
// Recursively sort the two parts
```

```
// left is the Index of the Left Element of the subarray
// right is the Index of the Right Element of the subarray
// Number of Elements in subarray = right-left+1

void CArray::Recursively_Sort(int* Element, unsigned int left,
unsigned int right)
{
  if(left < right) //  If array has two or more elements
  {
  int Pivot_X=this->Recursive_Sort_Partition(Element,left,right);

    if(Pivot_X != 0)
        this->Recursively_Sort(Element, left, Pivot_X - 1);
        // Elements smaller than the pivot

        this->Recursively_Sort(Element, Pivot_X + 1, right);
        // Elements bigger than the pivot
    }

}

// The Final step is to move the pivot between the two regions
by swapping
int CArray::Recursive_Sort_Partition(int* Element,
unsigned int left, unsigned int right)
{
    int Pivot = Element[right];

    unsigned int Index = left;

    for(unsigned int i = left; i < right; i++)
    {
        if(Element[i] <= Pivot)
        {
            this -> Swap(i, Index); // If swapped , Increment
            Index++;
        }
    }

    this -> Swap(Index, right); // Move Pivot to end

    return Index;
}

void CArray::QuickSort()
{
    this -> Recursively_Sort(this -> Arr, 0, this -> noelements - 1);

}

//------------------  4. Selection Sort -------------------
// In selection sort the array is divided into two parts
// The first part that is sorted and the second part
that is not sorted
// Initially the sorted part is empty and the unsorted
part consists of the whole array
```

```
// In each step, the algorithm searches through the unsorted part,
// Finds the smallest element and puts it at the end of
the sorted part

void CArray::SelectionSort()
{
  unsigned int Sorted_Part = 0;

  for(unsigned int i = 0; i < this -> noelements; i++)
   {
    Sorted_Part = i;

    for(unsigned int j = i + 1; j < this -> noelements; j++)
      if(this -> Arr[j] < this -> Arr[Sorted_Part])
      // Finds the smallest element
               Sorted_Part = j;

       this -> Swap(i, Sorted_Part);
    }
}

//---------------------- 5. Insertion Sort ----------------------------
// The initialization of the algorithm is similar to the
selection sort
// Dividing the array into a sorted and an unsorted part
// Each step of the algorithm picks the first item of the
unsorted array and
// Inserts it into the right slot of the sorted array

void CArray::InsertionSort()
{
    int Insert = 0;
    unsigned int Sorted_null = 0;

    for(unsigned int i = 1; i < this -> noelements; i++)
    {
     Insert = this -> Arr[i];
        // Value will be inserted into the array
     Sorted_null = i;
        // position i as the null Index

     while(Sorted_null > 0 && Insert < this -> Arr[Sorted_null -1])
        {
        // Shift the larger value up
    this -> Arr[Sorted_null] = this -> Arr[Sorted_null - 1];
            Sorted_null--;
        }
        // Inserts it into the right slot of the sorted array
    this -> Arr[Sorted_null] = Insert;
    }
}



//--------------------6. Sort using binary trees //-------------------

// Create a new node
```

```
Node *CArray::CreateNode(int done)
{
    Node *create = new Node;
    create -> Data = done;
    create -> left = create -> right = nullptr;
    return create;
}

// Store sorted elements in an array
void CArray::Store_value(Node *root, int *store, int &m)
{
    if (root != nullptr)
    {
        Store_value(root -> left, store, m);
        store[m++] = root -> Data;
        Store_value(root -> right, store, m);
    }
}

// Insert values to the new node
Node *CArray::Insert_value(Node *node, int data)
{
    // If the node is empty, return a new Node
    if (node == nullptr)
        return CreateNode(data);

    // Down the tree
    if (data < node -> Data)
        node -> left  = Insert_value(node -> left, data);

    else if (data > node -> Data)
        node -> right = Insert_value(node -> right, data);

    return node;
}

/*
// The Pre-Order traversal:at each node the root is evaluated first
// then the left sub tree, then the right subtree.
void CArray::PreOrder(Node *node, int data)
{
    if(node -> Data != 0)
        cout<<"array"<<node->data<<endl;

    if(node -> left != 0)
        node -> left = PreOrder(node -> left, data);

    if(node->right != 0)
        node -> right = PreOrder(node -> right, data);
*/

// Binary Sort Algorithm
void CArray::BinarySort()
{
    Node *root = nullptr;

    root = Insert_value(root, this -> Arr[0]);
```

```
    for (unsigned int i=0; i < this -> noelements; i++)
        Insert_value(root, this -> Arr[0]);

    // Store inoder traversal of the BST
    int n = 0;
    Store_value(root, this -> Arr, n);
}
```

3 Sorting Algorithms:: main file

```
#include <iostream>
#include "tut5.h"
using namespace std;


int main()
{

    cout << "   LAB 5: Sorting Algorithms" << endl;
    cout << endl;


    // 1. Preliminary work
    cout << "          1. Preliminary work: " << endl;
    cout << endl;
    // Display the values of the array
    CArray Sort_Arr;
    Sort_Arr.Display();
    cout << endl;

    // 2. Bubble Sort

    cout << "          2. Bubble Sort:" << endl;
    cout << endl;
    cout << "Performing Bubblesort .... " << endl;
    cout << endl;
    cout << "After Bubblesort " << endl;
    Sort_Arr.BubbleSort();
    Sort_Arr.Display();
    cout << endl;

    // 3. Quicksort

    cout << "          3. Quick Sort:" << endl;

    cout << endl;
    cout << "Performing Quicksort .... " << endl;
    cout << endl;
    cout << "After quick sort " << endl;
    Sort_Arr.QuickSort();
    Sort_Arr.Display();
    cout << endl;

    // 4. Selection Sort

    cout << "          4. Selection Sort:" << endl;
```

```
    cout << endl;
    cout << "Performing Selection Sort ... " << endl;
    cout << endl;
    cout << "After Selection Sort " << endl;
    Sort_Arr.SelectionSort();
    Sort_Arr.Display();
    cout << endl;

    // 5. Insertion Sort

    cout << "           5. Insertion Sort:" << endl;
    cout << endl;
    cout << "Performing Insertion Sort ... " << endl;
    cout << endl;
    cout<< "After Insertion Sort " <<endl;
    Sort_Arr.InsertionSort();
    Sort_Arr.Display();
    cout << endl;


    // 6. Sort using binary trees
    cout << "           6. Sort using Binary trees:" << endl;
    cout << endl;
    cout << "Performing Binary Sort ... " << endl;
    cout << endl;
    cout<< "After Binary Sort " <<endl;
    Sort_Arr.BinarySort();
    Sort_Arr.Display();
    cout << endl;


    return 0;

}
```

****

4 Outputs of Sorting algorithms

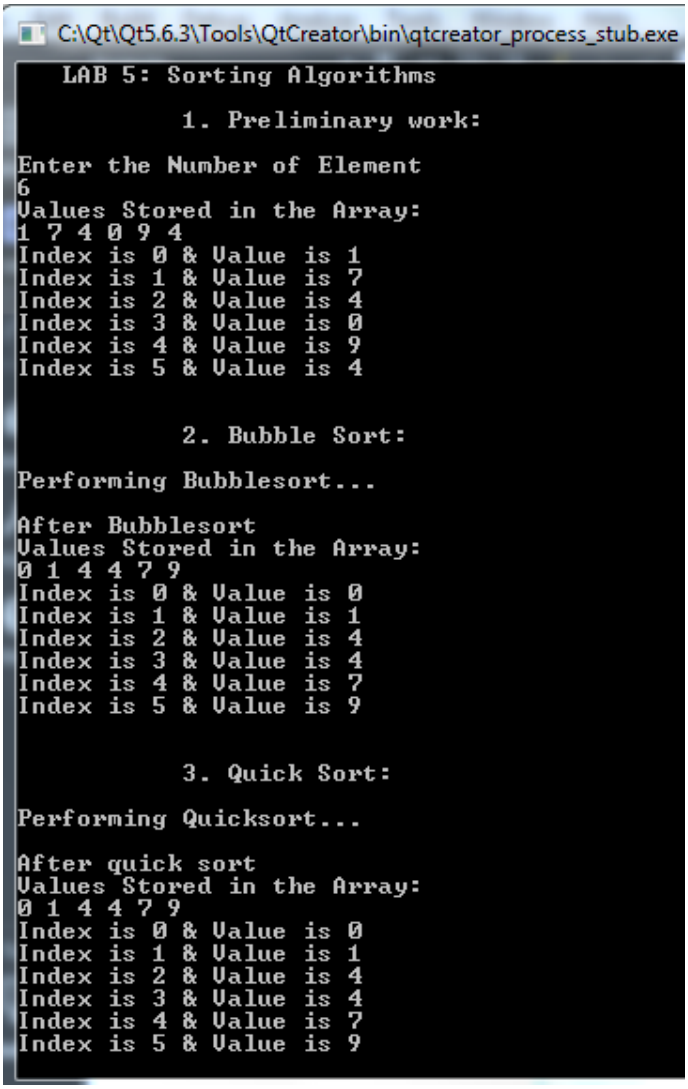

Figure 1: All outputs

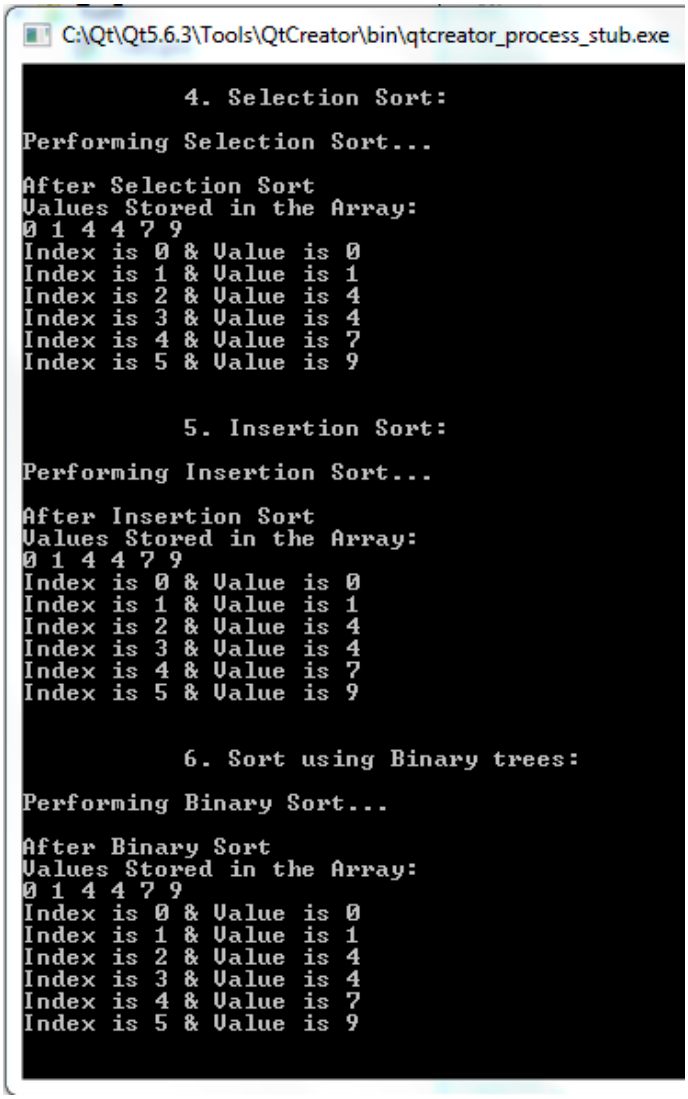

Figure 2: All outputs