CADT IDT វិទ្យាស្ថានបច្ចេកវិទ្យាឌីជីថល Institute of Digital Technology

Course T1Y2: Advanced Algorithms

Lecturer: Bou Channa

Student's name: Chea Ilong

ID: 100022

Group: 1 SE Gen10

Lab3: Sorting Algorithms

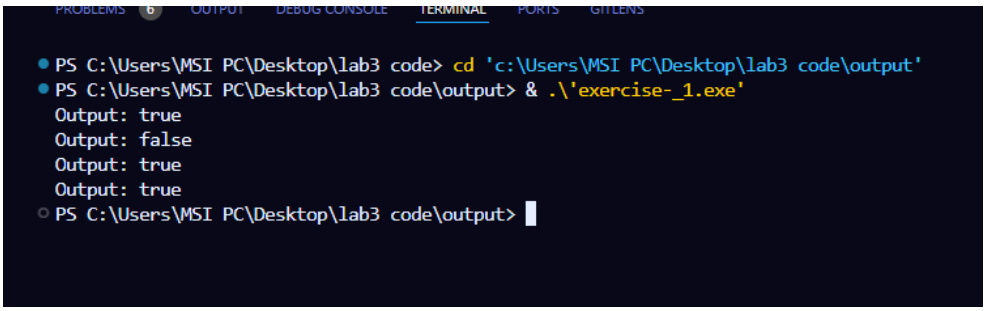
Sorting algorithms is a method that is use to arrange an array or a set of data in a specific order. There are various algorithms that have difference approach, efficiencies and use case like bubble sort, selection sort, insertion sort and so on.

Exercise1:

Source code:

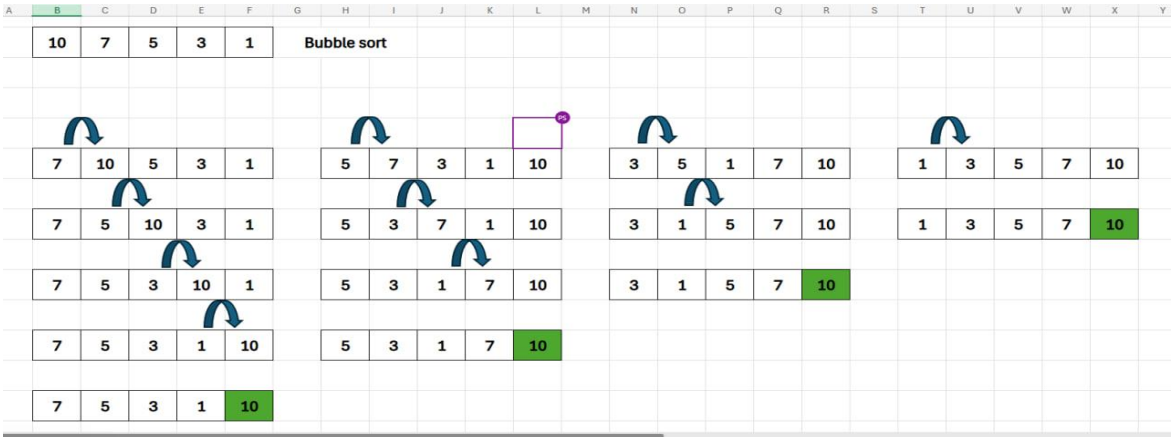
```cpp
#include <iostream>
#include <vector>
using namespace std;

bool issorted(vector<int> &array)
{
    int n = array.size();
    for (int i = 0; i < n-1; i++)
    {
        if (array[i] > array[i + 1])
        {

            return false;
        }
    }
    return true;
}

int main()
{

    vector<int> input1 = {1, 2, 8, 10, 49};
    vector<int> input2 = {1, 2, 8, 6, 49};
    vector<int> input3 = {3, 3, 3, 3};
    vector<int> input4 = {};

    cout << boolalpha;
    cout << "Output: " << issorted(input1) << endl;
    cout << "Output: " << issorted(input2) << endl;
    cout << "Output: " << issorted(input3) << endl;
    cout << "Output: " << issorted(input4) << endl;

    return 0;
}
```

Output:



Exercise2: Bubble Sort

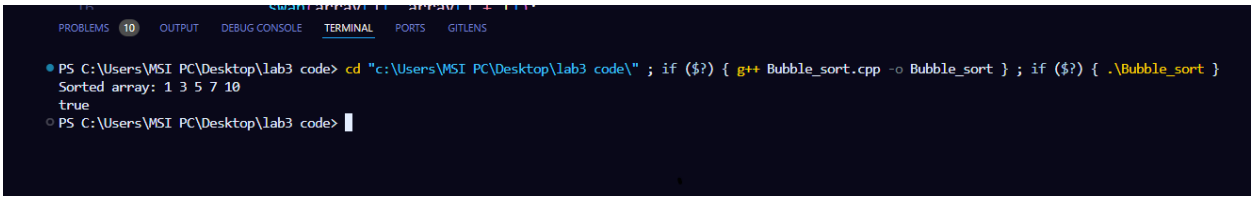
Bubble sort is a simple algorithm that is very common and easy to implement. It is use for sorting array of a set of data to arrange them into an order that is ascending or descending order. It works by repeatedly comparing the adjacent elements and swapping them if they are in the wrong arrange order.



Source Code:

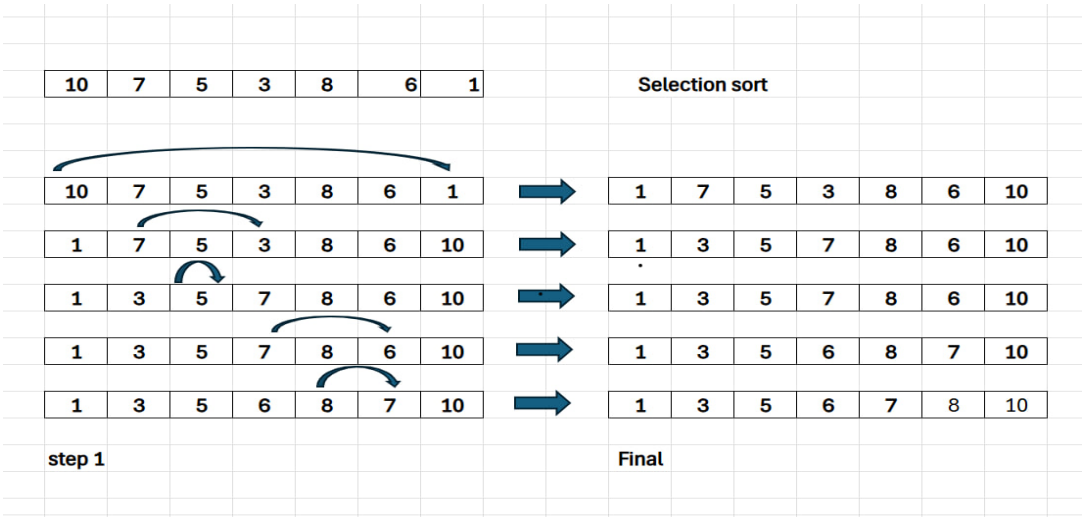
```cpp
#include <iostream>
#include <vector>
using namespace std;

void bubblesort(vector<int> &array)
{

    int n = array.size();

    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (array[j] > array[j + 1])
            {
                swap(array[j], array[j + 1]);
            }
        }
    }
}
bool issort(vector<int> &array)
{
    int n = array.size();
    for (int i = 0; i < n - 1; i++)
    {
        if (array[i] > array[i + 1])
        {
            return false;
        }
    }
    return true;
}

int main()
{

    vector<int> input = {10, 7, 5, 3, 1};

    bubblesort(input);

    cout << "Sorted array: ";

    for (int num : input)
    {
        cout << num << " ";
    }
    cout << endl;

    cout << boolalpha;
    cout << issort(input) << endl;

    return 0;
}

```

Output:



Exercise3: Selection Sort:

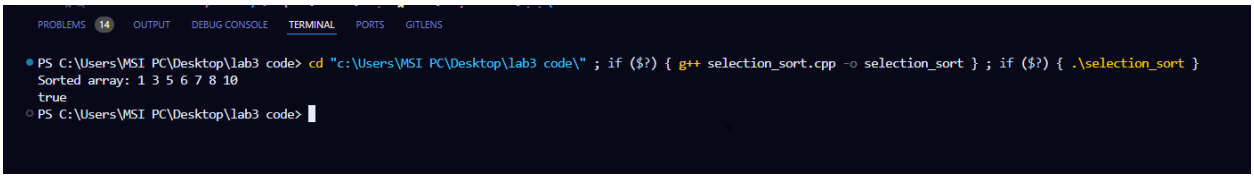
Selection sort Selection Sort is a comparison-based sorting algorithm. It sorts an array by repeatedly selecting the smallest (or largest) element from the unsorted portion and swapping it with the first unsorted element. This process continues until the entire array is sorted.



Source code:

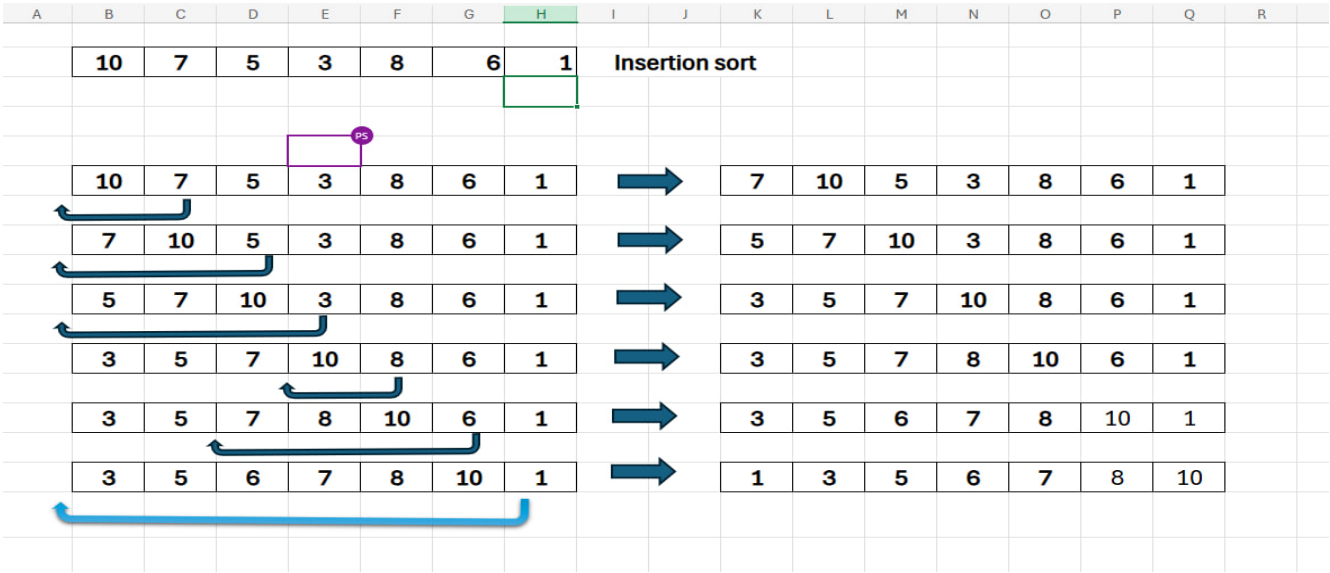
```cpp
#include <iostream>
#include <vector>
using namespace std;

void selectionsort(vector<int> &array)
{

    int n = array.size();
    for (int i = 0; i < n - 1; i++)
    {
        int minIndex = i;
        int minValue = array[i];
        for (int j = i + 1; j < n; j++)
        {
            if (array[j] < minValue)
            {
                minIndex = j;
                minValue = array[j];
            }
        }
        array[minIndex] = array[i];
        array[i] = minValue;
    }
}
bool issort(vector<int> &array)
{
    int n = array.size();
    for (int i = 0; i < n - 1; i++)
    {
        if (array[i] > array[i + 1])
        {
            return false;
        }
    }
    return true;
}

int main()
{

    vector<int> input = {10, 7, 5, 3, 8, 6, 1};

    selectionsort(input);

    cout << "Sorted array: ";

    for (int num : input)
    {
        cout << num << " ";
    }
    cout << endl;

    cout << boolalpha;
    cout << issort(input) << endl;

    return 0;
}

```

Output:



Exercise4: Insertion Sort:

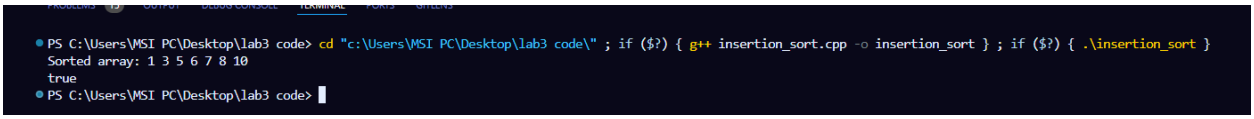
Insertion Sort is a straightforward sorting algorithm that arranges elements by repeatedly placing each item from an unsorted portion into its correct position within an already sorted section of the list. It works much like organizing playing cards in your hand: you separate the cards into two groups — one sorted and one unsorted. Then, you pick a card from the unsorted group and place it into its correct spot within the sorted group, continuing this process until all cards are sorted.



Source code:

```cpp
#include <iostream>
#include <vector>
using namespace std;

void insertionsort(vector<int> &array)
{

    int n = array.size();
    for (int i = 1; i < n ; i++) {
            int key = array[i];
            int j = i - 1;
            while (j >= 0 && array[j] > key) {
                array[j + 1] = array[j];
                j = j -1;
            }
            array[j + 1] = key;
        }
}
bool issort(vector<int> &array)
{
    int n = array.size();
    for (int i = 0; i < n - 1; i++)
    {
        if (array[i] > array[i + 1])
        {
            return false;
        }
    }
    return true;
}

int main()
{

    vector<int> input = {10, 7, 5, 3, 8, 6, 1};

    insertionsort(input);

    cout << "Sorted array: ";

    for (int num : input)
    {
        cout << num << " ";
    }
    cout << endl;

    cout << boolalpha;
    cout << issort(input) << endl;

    return 0;
}

```

Output:



Exercise5: Performance Analysis

We will measure the performance of the following sorting algorithms on the same set of values

• Selection Sort

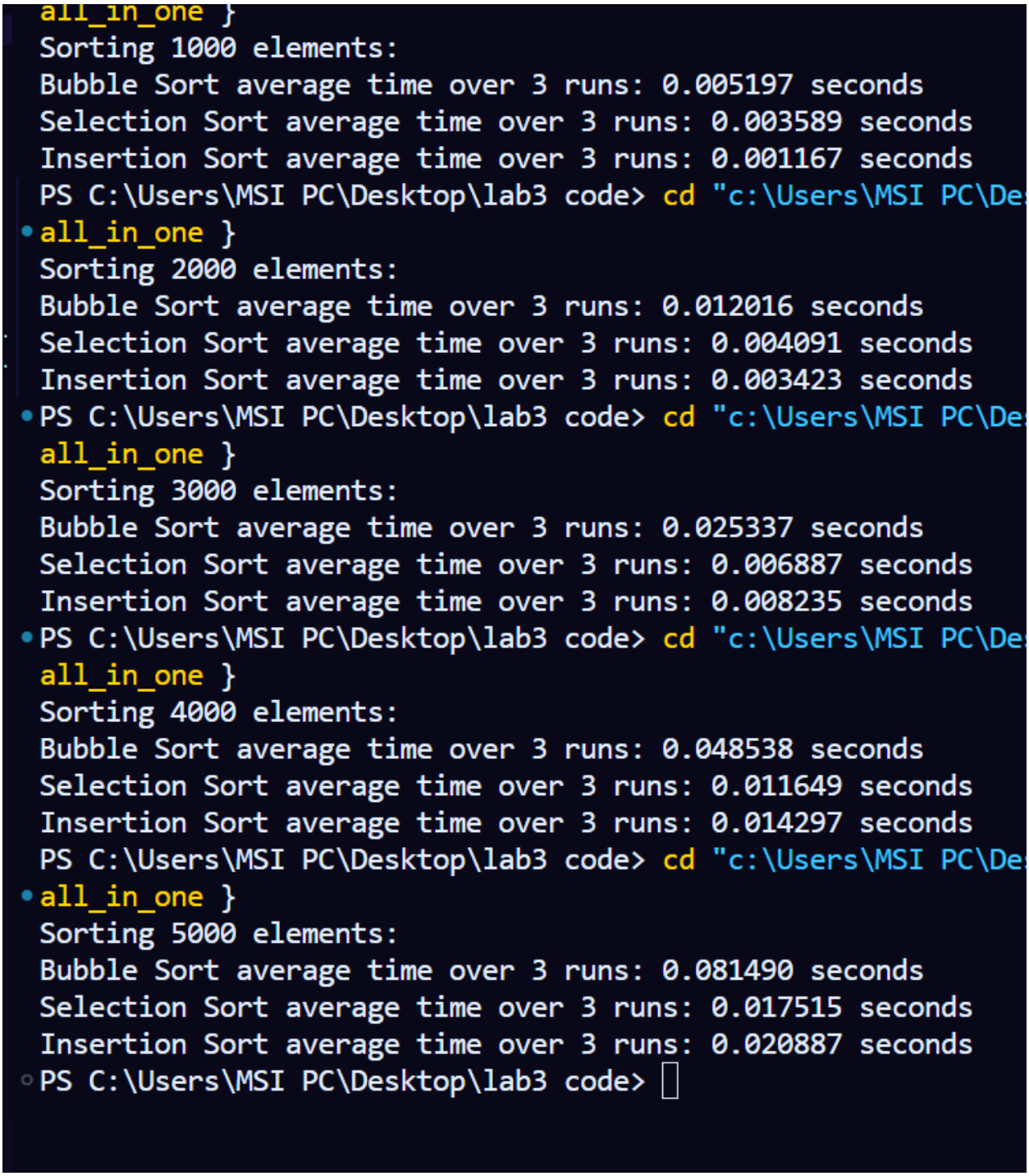
• Insertion Sort

• Bubble Sort

Source code:

```cpp
#include <iostream>
#include <vector>
#include <chrono>
#include <iomanip>
#include <cstdlib>

using namespace std;

// Bubble Sort
void bubblesort(vector<int> &array) {
    int n = array.size();
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (array[j] > array[j + 1]) {
                swap(array[j], array[j + 1]);
            }
        }
    }
}

// Selection Sort
void selectionsort(vector<int> &array) {
    int n = array.size();
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        int minValue = array[i];
        for (int j = i + 1; j < n; j++) {
            if (array[j] < minValue) {
                minIndex = j;
                minValue = array[j];
            }
        }
        array[minIndex] = array[i];
        array[i] = minValue;
    }
}

// Insertion Sort
void insertionsort(vector<int> &array) {
    int n = array.size();
    for (int i = 1; i < n; i++) {
        int key = array[i];
        int j = i - 1;
        while (j >= 0 && array[j] > key) {
            array[j + 1] = array[j];
            j = j - 1;
        }
        array[j + 1] = key;
    }
}

// Utility function to generate an array of random numbers
vector<int> generateRandomArray(int size) {
    vector<int> array(size);
    for (int i = 0; i < size; i++) {
        array[i] = rand() % 10000;  // Generate numbers between 0 and 9999
    }
    return array;
}

// Function to measure the execution time of a sorting algorithm using <chrono>
void measureSortingTime(void (*sortFunction)(vector<int>&), vector<int> array, const string &sortName, int repeats = 5) {
    double total_time = 0.0;

    for (int i = 0; i < repeats; i++) {
        vector<int> tempArray = array;  // Copy the array each time for accurate measurement
        auto start = chrono::high_resolution_clock::now();
        sortFunction(tempArray);
        auto end = chrono::high_resolution_clock::now();
        chrono::duration<double> time_taken = end - start;
        total_time += time_taken.count();
    }

    double average_time = total_time / repeats;
    cout << sortName << " average time over " << repeats << " runs: " << fixed << setprecision(6) << average_time << " seconds" << endl;
}

int main() {
    srand(time(0));  // Initialize random seed

    const int N = 5000;  // Increase size of the arrays for more measurable times

    // Generate an identical array for each algorithm
    vector<int> array1 = generateRandomArray(N);
    vector<int> array2 = array1;  // Copy array1
    vector<int> array3 = array1;  // Copy array1

    // Measure and print sorting times
    cout << "Sorting " << N << " elements:" << endl;
    measureSortingTime(bubblesort, array1, "Bubble Sort", 3);
    measureSortingTime(selectionsort, array2, "Selection Sort", 3);
    measureSortingTime(insertionsort, array3, "Insertion Sort", 3);

    return 0;
}
```

Output:


all_in_one }
Sorting 1000 elements:
Bubble Sort average time over 3 runs: 0.005197 seconds
Selection Sort average time over 3 runs: 0.003589 seconds
Insertion Sort average time over 3 runs: 0.001167 seconds
PS C:\Users\MSI PC\Desktop\lab3 code> cd "c:\Users\MSI PC\De
all_in_one }
Sorting 2000 elements:
Bubble Sort average time over 3 runs: 0.012016 seconds
Selection Sort average time over 3 runs: 0.004091 seconds
Insertion Sort average time over 3 runs: 0.003423 seconds
PS C:\Users\MSI PC\Desktop\lab3 code> cd "c:\Users\MSI PC\De
all_in_one }
Sorting 3000 elements:
Bubble Sort average time over 3 runs: 0.025337 seconds
Selection Sort average time over 3 runs: 0.006887 seconds
Insertion Sort average time over 3 runs: 0.008235 seconds
PS C:\Users\MSI PC\Desktop\lab3 code> cd "c:\Users\MSI PC\De
all_in_one }
Sorting 4000 elements:
Bubble Sort average time over 3 runs: 0.048538 seconds
Selection Sort average time over 3 runs: 0.011649 seconds
Insertion Sort average time over 3 runs: 0.014297 seconds
PS C:\Users\MSI PC\Desktop\lab3 code> cd "c:\Users\MSI PC\De
all_in_one }
Sorting 5000 elements:
Bubble Sort average time over 3 runs: 0.081490 seconds
Selection Sort average time over 3 runs: 0.017515 seconds
Insertion Sort average time over 3 runs: 0.020887 seconds
PS C:\Users\MSI PC\Desktop\lab3 code>

Array Size	Bubble sort	Selection sort	Insertion sort
1000	0.005197s	0.003589s	0.001167s
2000	0.012016s	0.004091s	0.003423s
3000	0.025337s	0.006887s	0.008235s
4000	0.048538s	0.011649s	0.014297s
5000	0.081490s	0.017515s	0.020887s