Muhammad Kashif (I21-0851)

Section: P

Assignment No: 2

Submitted To: Ma’am Behjat Zuhaira



National University of Computing and Emerging Sciences, Islamabad

15th Oct, 2023

**Report on Selection Sort Algorithm**

**Code to generate Dataset**

#include <iostream>

#include <fstream>

#include <cstdlib>

#include <ctimes>

#include <algorithm>

#include <string>

using namespace std;

struct Student {

    string id;

    int rollNo;

    int subject1;

    int subject2;

    int subject3;

    int subject4;

    int subject5;

};

// Function to generate a unique random 5-digit roll number

int generateUniqueRollNo(int usedRollNumbers[], int& usedRollCount) {

    int rollNo;

    do {

        rollNo = rand() % 90000 + 10000;  // Generate a random 5-digit number

    } while (find(usedRollNumbers, usedRollNumbers + usedRollCount, rollNo) != usedRollNumbers + usedRollCount); // Ensure uniqueness

    usedRollNumbers[usedRollCount++] = rollNo;

    return rollNo;

}

// Function to generate random student data

Student generateRandomStudent(int usedRollNumbers[], int& usedRollCount)

{

    Student student;

    student.rollNo = generateUniqueRollNo(usedRollNumbers, usedRollCount);

    student.id = "I" + to\_string(student.rollNo);

    student.subject1 = rand() % 101; // Random score between 0 and 100

    student.subject2 = rand() % 101;

    student.subject3 = rand() % 101;

    student.subject4 = rand() % 101;

    student.subject5 = rand() % 101;

    return student;

}

int main()

{

    srand(static\_cast<unsigned>(time(nullptr)));

    const int numStudents = 10000;

    const string filename = "student\_data.csv";

    ofstream outputFile(filename);

    if (!outputFile) {

        cerr << "Error: Unable to open the file " << filename << endl;

        return 1;

    }

    // Write CSV header

    outputFile << "Id,Subject1,Subject2,Subject3,Subject4,Subject5" << endl;

    int usedRollNumbers[numStudents] = { 0 }; // Initialize with 0

    int usedRollCount = 0;

    // Generate and write student data

    for (int i = 1; i <= numStudents; ++i) {

        Student student = generateRandomStudent(usedRollNumbers, usedRollCount);

        outputFile << student.id<<","

            << student.subject1 << "," << student.subject2 << ","

            << student.subject3 << "," << student.subject4 << ","

            << student.subject5 << endl;

    }

    outputFile.close();

    cout << "Data has been generated and written to " << filename << endl;

    return 0;

}

# Selection-Sort Analysis

**Machine Specifications:**

* Processor: Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
* Installed RAM: 16 GB
* Operating System: 64 Bit OS, x64 based processor, Windows 11

**Selection Sort Results:** I conducted a series of experiments to analyse the behaviour of the Selection Sort algorithm. The data consisted of 10,000 students, sorted based on their roll numbers. The execution times (in milliseconds) were measured for various input sizes, ranging from 200 to 10,000 students, with increments of 200.

A graph with a line

Description automatically generated

**Selection Sort Algorithm Description:** The Selection Sort algorithm is a simple sorting algorithm that repeatedly selects the minimum element from the unsorted portion of the list and moves it to the beginning. This process continues until the entire list is sorted. Selection Sort has a worst-case time complexity of O(n^2), making it less efficient for larger input sizes.

**Behaviour and Justifications:** The provided results clearly illustrate the behaviour of the Selection Sort algorithm. As the input size increases, the execution time also increases, demonstrating a quadratic growth pattern. This aligns with the expected behaviour, considering the algorithm's O(n^2) worst-case time complexity.

The algorithm efficiently handles smaller input sizes, as reflected in the minimal execution times. However, as the input size grows, the algorithm's performance degrades noticeably. This behaviour is due to the inherent nature of Selection Sort, which involves comparing and swapping elements extensively.

**Relationship with Machine Specifications:** The machine specifications, including the Intel Core i7 processor and 16 GB of RAM, indicate a capable computing environment. However, for larger input sizes, the algorithm's performance is noticeably affected. This behaviour can be attributed to the machine's processing power and memory capacity.

As the input size increases, the algorithm's execution time grows substantially, suggesting that the machine's processor may be challenged when handling the sorting process. Additionally, the memory may also play a role in the algorithm's performance, as sorting larger datasets may require more memory resources.

## Worst-Case Time Complexity of Selection Sort (O(n^2))

The graph illustrates the worst-case time complexity of the Selection Sort algorithm as a function of the input size (n). This analysis is a key aspect of understanding how the algorithm behaves when facing its most challenging scenario, where elements are arranged in a way that requires a maximum number of comparisons and swaps.

A graph with a line

Description automatically generated

The x-axis represents the input size (n), which increases in increments of 200, while the y-axis represents the worst-case execution time, obtained by squaring the input size (n^2). The results show a quadratic growth pattern, which aligns with the expected worst-case time complexity of O(n^2) for Selection Sort.

* For smaller input sizes (e.g., 200 to 2,000), the execution time is relatively low, consistent with the efficient nature of Selection Sort for small datasets. It indicates that the algorithm can quickly sort such inputs.
* However, as the input size increases beyond 2,000, the execution time experiences a substantial increase. This rapid growth signifies that Selection Sort becomes significantly less efficient when dealing with larger datasets.

The graph reinforces the fact that Selection Sort is less suitable for sorting large datasets, where its quadratic worst-case time complexity leads to a high number of comparisons and swaps, resulting in longer execution times.

**Justifications:** The behaviour depicted in the graph aligns with the theoretical analysis of Selection Sort's worst-case time complexity. The O(n^2) worst-case scenario occurs when the algorithm requires the maximum number of comparisons and swaps to sort an input, as observed in the graph.

This behaviour is justified by the nature of Selection Sort, which involves selecting the minimum element through multiple comparisons and then placing it in its correct position. In the worst-case scenario, the algorithm must perform these comparisons and swaps for each element in the input, leading to a quadratic growth in execution time.