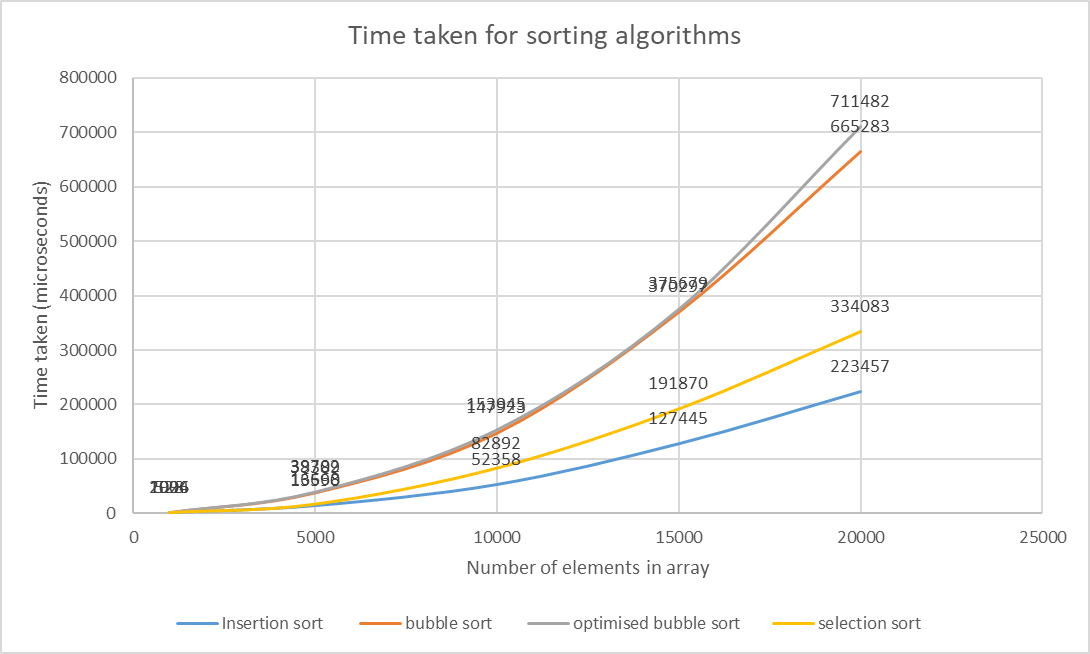
**CS2023 – Data structures and algorithms**

**Lab 03 report**

**Sajeev Kugarajah (210554M)**

|  |  |  |
| --- | --- | --- |
| **Sorting Type** | **Elements in array** | **Time taken (microseconds)** |
| Insertion Sort | 1000 | 528 |
| 5000 | 13590 |
| 10000 | 52358 |
| 150000 | 127445 |
| 200000 | 223457 |
| Bubble Sort | 1000 | 1994 |
| 5000 | 38389 |
| 10000 | 147923 |
| 150000 | 370297 |
| 200000 | 665283 |
| Optimised Bubble Sort | 1000 | 2026 |
| 5000 | 39702 |
| 10000 | 153945 |
| 150000 | 375679 |
| 200000 | 711482 |
| Selection Sort | 1000 | 1006 |
| 5000 | 16668 |
| 10000 | 82892 |
| 150000 | 191870 |
| 200000 | 334083 |

**1. Output**

**2. Plot**

**3. Conclusion**

Bubble sort and optimized bubble sort algorithms are simple and easy to understand but the time complexities are higher than other sorting algorithms.

The optimized bubble sort algorithm is a small upgrade to the bubble sort algorithm, where it can find whether the array is already sorted or not and execute or stop execution according to the state of an array.

But in the worst-case scenarios, both of these are the same.

Selection sort also has the same time complexity as bubble sort algorithms, but it can deal with larger arrays better than bubble sort algorithms, as you can see in the plot when the array size is getting larger, the time taken for selection sort is becoming lower than bubble sort.

above all algorithms, insertion sort is far better. but it can perform well in small arrays and best-case scenarios. it also has the same time complexity as other algorithms

**3. Appendix (source codes)**

1. **insertion sort**

#include <iostream>

#include <random>

#include <chrono>

using namespace std;

void insertionSort(int arr[], int size)

{

    for (int i = 1; i < size; i++)

    {

        int key = arr[i];

        int j = i - 1;

        while (j >= 0 && arr[j] > key)

        {

            arr[j + 1] = arr[j];

            j--;

        }

        arr[j + 1] = key;

    }

}

int main()

{

    // initializing sizes for random arrays

    int sizes[5] = {1000, 5000, 10000, 15000, 20000};

    // create a random number generator

    random\_device rd;

    mt19937 gen(rd());

    uniform\_int\_distribution<> dis(1, 1000); // create random number between 1,1000

    for (int i = 0; i < 5; i++)

    {

        int size = sizes[i];

        int arr[size];

        for (int i = 0; i < size; i++)

        {

            arr[i] = dis(gen); // assigning random number to array

        }

        // printing initial array

        cout << "Initial array with size " << size << " is : ";

        for (int i = 0; i < size; i++)

        {

            cout << arr[i] << " ";

        }

        cout << endl;

        // calculating the time taken for sorting in microseconds

        auto start\_time = chrono::steady\_clock::now();

        insertionSort(arr, size);

        auto end\_time = chrono::steady\_clock::now();

        auto time\_taken = chrono::duration\_cast<chrono::microseconds>(end\_time - start\_time).count();

        // printing the final sorted array

        cout << "Sorted array after insertion sort : ";

        for (int i = 0; i < size; i++)

        {

            cout << arr[i] << " ";

        }

        cout << endl;

        cout << "Time taken : " << time\_taken << " microseconds" << endl;

    }

}

1. **bubble sort**

#include <iostream>

#include <random>

#include <chrono>

using namespace std;

void bubbleSort(int arr[], int size)

{

    for (int i = 0; i < size - 1; i++)

    {

        for (int j = 0; j < size - i - 1; j++)

        {

            if (arr[j] > arr[j + 1])

            {

                // swap the elements

                int temp = arr[j];

                arr[j] = arr[j + 1];

                arr[j + 1] = temp;

            }

        }

    }

}

int main()

{

    // initializing sizes for random arrays

    int sizes[5] = {1000, 5000, 10000, 15000, 20000};

    // create a random number generator

    random\_device rd;

    mt19937 gen(rd());

    uniform\_int\_distribution<> dis(1, 1000); // create random number between 1,1000

    for (int i = 0; i < 5; i++)

    {

        int size = sizes[i];

        int arr[size];

        for (int i = 0; i < size; i++)

        {

            arr[i] = dis(gen); // assigning random number to array

        }

        // printing initial array

        cout << "Initial array with size " << size << " is : ";

        for (int i = 0; i < size; i++)

        {

            cout << arr[i] << " ";

        }

        cout << endl;

        // calculating the time taken for sorting in microseconds

        auto start\_time = chrono::steady\_clock::now();

        bubbleSort(arr, size);

        auto end\_time = chrono::steady\_clock::now();

        auto time\_taken = chrono::duration\_cast<chrono::microseconds>(end\_time - start\_time).count();

        // printing the final sorted array

        cout << "Sorted array after bubble sort : ";

        for (int i = 0; i < size; i++)

        {

            cout << arr[i] << " ";

        }

        cout << endl;

        cout << "Time taken : " << time\_taken << " microseconds" << endl;

    }

}

1. **optimized bubble sort**

#include <iostream>

#include <random>

#include <chrono>

using namespace std;

void optimisedBubbleSort(int arr[], int size)

{

    bool swapped; // flag to check if any swaps were made in the previous iteration

    for (int i = 0; i < size - 1; i++)

    {

        swapped = false;

        for (int j = 0; j < size - i - 1; j++)

        {

            if (arr[j] > arr[j + 1])

            {

                // swap the elements

                int temp = arr[j];

                arr[j] = arr[j + 1];

                arr[j + 1] = temp;

                swapped = true;

            }

        }

        // if no swaps were made in this iteration, the array is already sorted

        if (!swapped)

        {

            break;

        }

    }

}

int main()

{

    // initializing sizes for random arrays

    int sizes[5] = {1000, 5000, 10000, 15000, 20000};

    // create a random number generator

    random\_device rd;

    mt19937 gen(rd());

    uniform\_int\_distribution<> dis(1, 1000); // create random number between 1,1000

    for (int i = 0; i < 5; i++)

    {

        int size = sizes[i];

        int arr[size];

        for (int i = 0; i < size; i++)

        {

            arr[i] = dis(gen); // assigning random number to array

        }

        // printing initial array

        cout << "Initial array with size " << size << " is : ";

        for (int i = 0; i < size; i++)

        {

            cout << arr[i] << " ";

        }

        cout << endl;

        // calculating the time taken for sorting in microseconds

        auto start\_time = chrono::steady\_clock::now();

        optimisedBubbleSort(arr, size);

        auto end\_time = chrono::steady\_clock::now();

        auto time\_taken = chrono::duration\_cast<chrono::microseconds>(end\_time - start\_time).count();

        // printing the final sorted array

        cout << "Sorted array after bubble sort : ";

        for (int i = 0; i < size; i++)

        {

            cout << arr[i] << " ";

        }

        cout << endl;

        cout << "Time taken : " << time\_taken << " microseconds" << endl;

    }

}

1. **selection sort**

#include <iostream>

#include <random>

#include <chrono>

using namespace std;

void selectionSort(int arr[], int size) {

    for (int i = 0; i < size - 1; i++) {

        int min\_index = i;

        for (int j = i + 1; j < size; j++) {

            if (arr[j] < arr[min\_index]) {

                min\_index = j;

            }

        }

        // swap the elements

        int temp = arr[i];

        arr[i] = arr[min\_index];

        arr[min\_index] = temp;

    }

}

int main()

{

    // initializing sizes for random arrays

    int sizes[5] = {1000, 5000, 10000, 15000, 20000};

    // create a random number generator

    random\_device rd;

    mt19937 gen(rd());

    uniform\_int\_distribution<> dis(1, 1000); // create random number between 1,1000

    for (int i = 0; i < 5; i++)

    {

        int size = sizes[i];

        int arr[size];

        for (int i = 0; i < size; i++)

        {

            arr[i] = dis(gen); // assigning random number to array

        }

        // printing initial array

        cout << "Initial array with size " << size << " is : ";

        for (int i = 0; i < size; i++)

        {

            cout << arr[i] << " ";

        }

        cout << endl;

        // calculating the time taken for sorting in microseconds

        auto start\_time = chrono::steady\_clock::now();

        selectionSort(arr, size);

        auto end\_time = chrono::steady\_clock::now();

        auto time\_taken = chrono::duration\_cast<chrono::microseconds>(end\_time - start\_time).count();

        // printing the final sorted array

        cout << "Sorted array after insertion sort : ";

        for (int i = 0; i < size; i++)

        {

            cout << arr[i] << " ";

        }

        cout << endl;

        cout << "Time taken : " << time\_taken << " microseconds" << endl;

    }

}