Data Structure and Algorithms

Submitted by

NAME OF THE CANDIDATE: SARANSH SALUJA

ROLL NO.: 2023UIC3661

BACHELOR OF TECHNOLOGY

IN

INSTRUMENTATION AND CONTROL ENGINEERING

DEPARTMENT OF INSTRUMENTATION AND CONTROL ENGINEERING
NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY
NEW DELHI – 110078, INDIA

November 2024

TABLE OF CONTENTS

S. No.	Topic	Page No.
1	Write a program to sort an array. Next, the program should take two arrays as input from the user, sort each of them individually, merge the two sorted arrays, and then sort the merged array as well.	1
2	Write a program to implement linear search and binary search algorithms. Analyze and compare their computational time complexities.	3
3	Write a program to implement the Bubble Sort algorithm.	6
4	Write a program to implement the Insertion Sort algorithm.	8
5	Write a program to implement the Selection Sort algorithm.	10
6	Write a program to implement the Merge Sort algorithm.	12
7	Write a program to implement the Quick Sort algorithm.	14

I. Write a program to sort an array. Next, the program should take two arrays as input from the user, sort each of them individually, merge the two sorted arrays, and then sort the merged array as well.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// Function to sort and display an array
void sortAndDisplay(vector<int>& arr, const string& message) {
    sort(arr.begin(), arr.end());
    cout << message << ": ";
    for (int num : arr) {
        cout << num << " ";
    }
    cout << endl;
}

// Function to merge two sorted arrays and sort the merged
    array
vector<int> mergeAndSortArrays(const vector<int>& arr1, const
    vector<int>& arr2) {
    vector<int> merged(arr1.begin(), arr1.end());
    merged.insert(merged.end(), arr2.begin(), arr2.end());
    sort(merged.begin(), merged.end());
    return merged;
}

int main() {
    int n1, n2;
    cout << "Enter the number of elements in the first array: "
        ;
    cin >> n1;

    vector<int> arr1(n1);
    cout << "Enter the elements of the first array: ";
    for (int i = 0; i < n1; i++) {
        cin >> arr1[i];
    }

    cout << "Enter the number of elements in the second array:
        ";
    cin >> n2;

    vector<int> arr2(n2);
    cout << "Enter the elements of the second array: ";
    for (int i = 0; i < n2; i++) {
        cin >> arr2[i];
    }

    // Sort and display individual arrays
    sortAndDisplay(arr1, "Sorted first array");
    sortAndDisplay(arr2, "Sorted second array");

    // Merge and sort the arrays, then display the result
    vector<int> mergedArray = mergeAndSortArrays(arr1, arr2);
    sortAndDisplay(mergedArray, "Merged and sorted array");

    return 0;
}
```

```
Enter the number of elements in the first array: 3
Enter the elements of the first array: 9 6 7
Enter the number of elements in the second array: 4
Enter the elements of the second array: 1 9 8 10
Sorted first array: 6 7 9
Sorted second array: 1 8 9 10
Merged and sorted array: 1 6 7 8 9 9 10


=== Code Execution Successful ===
```

II. Write a program to implement linear search and binary search algorithms. Analyse and compare their computational time complexities.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>

using namespace std;
using namespace chrono;

// Function for linear search
int linearSearch(const vector<int>& arr, int target) {
    for (size_t i = 0; i < arr.size(); i++) {
        if (arr[i] == target) {
            return i; // Return index if found
        }
    }
    return -1; // Return -1 if not found
}

// Function for binary search
int binarySearch(const vector<int>& arr, int target) {
    int left = 0, right = arr.size() - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2; // To avoid overflow
        if (arr[mid] == target) {
            return mid; // Return index if found
        } else if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return -1; // Return -1 if not found
}
```

```cpp
int main() {
    int n, target;

    cout << "Enter the number of elements: ";
    cin >> n;

    vector<int> arr(n);
    cout << "Enter the elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    cout << "Enter the target element to search: ";
    cin >> target;

    // Measure time for Linear Search
    auto startLinear = high_resolution_clock::now();
    int resultLinear = linearSearch(arr, target);
    auto endLinear = high_resolution_clock::now();
    auto durationLinear = duration_cast<nanoseconds>(endLinear - startLinear
        );

    if (resultLinear != -1) {
        cout << "Linear Search: Element found at index " << resultLinear <<
            endl;
    } else {
        cout << "Linear Search: Element not found" << endl;
    }
    cout << "Time taken by Linear Search: " << durationLinear.count() << "
        ns" << endl;

    // Sort the array for Binary Search
    sort(arr.begin(), arr.end());

    // Measure time for Binary Search
    auto startBinary = high_resolution_clock::now();
    int resultBinary = binarySearch(arr, target);
    auto endBinary = high_resolution_clock::now();
    auto durationBinary = duration_cast<nanoseconds>(endBinary - startBinary
        );

    if (resultBinary != -1) {
        cout << "Binary Search: Element found at index " << resultBinary <<
            " (in sorted array)" << endl;
    } else {
        cout << "Binary Search: Element not found" << endl;
    }
    cout << "Time taken by Binary Search: " << durationBinary.count() << "
        ns" << endl;

    // Complexity Analysis
    cout << "\nTime Complexity Analysis:" << endl;
    cout << "Linear Search: O(n)" << endl;
    cout << "Binary Search: O(log n)" << endl;

    return 0;
}
```

```
Enter the number of elements: 5
Enter the elements: 1 3 5 7 9
Enter the target element to search: 7
Linear Search: Element found at index 3
Time taken by Linear Search: 690 ns
Binary Search: Element found at index 3 (in sorted array)
Time taken by Binary Search: 110 ns

Time Complexity Analysis:
Linear Search: O(n)
Binary Search: O(log n)
```

III. Write a program to implement the Bubble Sort algorithm.

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

// Function to implement Bubble Sort
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap arr[j] and arr[j + 1]
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

// Function to print an array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array: ";
    printArray(arr, n);

    bubbleSort(arr, n);

    cout << "Sorted array: ";
    printArray(arr, n);

    return 0;
}
```

Output

```
Original array: 64 34 25 12 22 11 90
Sorted array: 11 12 22 25 34 64 90


=== Code Execution Successful ===
```

IV. Write a program to implement the Insertion Sort algorithm.

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

// Function to implement Insertion Sort
void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        // Move elements of arr[0..i-1], that are greater than key, to one position ahead of their
            current position
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

// Function to print an array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array: ";
    printArray(arr, n);

    insertionSort(arr, n);

    cout << "Sorted array: ";
    printArray(arr, n);

    return 0;
}
```

Output

```
Original array: 12 11 13 5 6
Sorted array: 5 6 11 12 13


=== Code Execution Successful ===
```

V. Write a program to implement the Selection Sort algorithm.

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

// Function to implement Selection Sort
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i; // Find the minimum element in the unsorted part
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        // Swap the found minimum element with the first element of the unsorted part
        swap(arr[minIndex], arr[i]);
    }
}

// Function to print the array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int arr[] = {29, 10, 14, 37, 13};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array: ";
    printArray(arr, n);

    selectionSort(arr, n);

    cout << "Sorted array: ";
    printArray(arr, n);

    return 0;
}
```

Output

```
Original array: 29 10 14 37 13
Sorted array: 10 13 14 29 37


=== Code Execution Successful ===
```

VI. Write a program to implement the Merge Sort algorithm.

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

// Function to merge two subarrays
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1], R[n2]; // Temporary arrays for left and right subarrays

    // Copy data to temp arrays
    for (int i = 0; i < n1; i++) {
        L[i] = arr[left + i];
    }
    for (int j = 0; j < n2; j++) {
        R[j] = arr[mid + 1 + j];
    }

    // Merge the temp arrays back into arr[left...right]
    int i = 0, j = 0, k = left;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Copy any remaining elements of L[]
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Copy any remaining elements of R[]
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

// Function to implement Merge Sort
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        // Sort first and second halves
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        // Merge the sorted halves
        merge(arr, left, mid, right);
    }
}

// Function to print the array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int arr[] = {38, 27, 43, 3, 9, 82, 10};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array: ";
    printArray(arr, n);

    mergeSort(arr, 0, n - 1);

    cout << "Sorted array: ";
    printArray(arr, n);

    return 0;
}
```

Output

```
Original array: 38 27 43 3 9 82 10
Sorted array: 3 9 10 27 38 43 82


=== Code Execution Successful ===
```

VII. Write a program to implement the Quick Sort algorithm.

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

// Function to partition the array into two subarrays
int partition(int arr[], int low, int high) {
    int pivot = arr[high];  // Taking the last element as the pivot
    int i = low - 1;  // Index of the smaller element

    // Traverse through all elements
    for (int j = low; j < high; j++) {
        // If current element is smaller than or equal to pivot
        if (arr[j] <= pivot) {
            i++;  // Increment index of smaller element
            swap(arr[i], arr[j]);  // Swap the elements
        }
    }

    // Place the pivot in the correct position
    swap(arr[i + 1], arr[high]);
    return i + 1;  // Return the partitioning index
}

// Function to implement Quick Sort
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        // Partition the array and get the pivot index
        int pi = partition(arr, low, high);

        // Recursively sort the subarrays
        quickSort(arr, low, pi - 1);  // Left part
        quickSort(arr, pi + 1, high); // Right part
    }
}

// Function to print the array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int arr[] = {38, 27, 43, 3, 9, 82, 10};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array: ";
    printArray(arr, n);

    quickSort(arr, 0, n - 1);

    cout << "Sorted array: ";
    printArray(arr, n);

    return 0;
```

Output

```
Original array: 38 27 43 3 9 82 10
Sorted array: 3 9 10 27 38 43 82


=== Code Execution Successful ===
```