**DS Lab 03: Sorting**

**Name: Shozab Mehdi**

**ID: 22k-4522**

**Task 1:**

#include<iostream>

#include<cstdlib>

using namespace std;

int main()

{

    cout << "For array having a size of 50" << endl << endl;

    int arr[50],random,count=0;

    bool swapped;

    cout << "Unsorted Array, having random values is:" << endl << endl;

    for (int i=0;i<50;i++){

        random=rand() % 100;

        arr[i]=random;

        cout << arr[i] << " ";

    }

    for (int j=0;j<49;j++){

        swapped=false;

        for (int k=0;k<49-j;k++){

            if (arr[k]>arr[k+1]){

                swap(arr[k],arr[k+1]);

                count++;

                swapped=true;

            }

            if (swapped=false){

                    break;

                }

        }

    }

    cout << endl << "Sorted array is:" << endl;

    for (int i=0;i<50;i++){

        cout << arr[i] << " ";

    }

    cout << endl << "Number of comparisons performed by the inner array: " << count << endl;

    int comparisons=0;

    for (int i=0;i<50;i++){

        cout << arr[i] << " ";

        comparisons++;

    }

    cout << "The number of comparisons in completely sorted array of 50 numbers: " << comparisons;

    cout << endl << "For array having a size of 100" << endl << endl;

    int arr1[100],random1,count1=0;

    cout << "Unsorted Array having random values is:" << endl << endl;

    for (int i=0;i<100;i++){

        random1=rand() % 100;

        arr1[i]=random1;

        cout << arr1[i] << " ";

    }

    for (int j=0;j<99;j++){

        for (int k=0;k<99-j;k++){

            if (arr1[k]>arr1[k+1]){

                swap(arr1[k],arr1[k+1]);

                count1++;

            }

        }

    }

    cout << endl << "Updated array is:" << endl;

    for (int i=0;i<100;i++){

        cout << arr1[i] << " ";

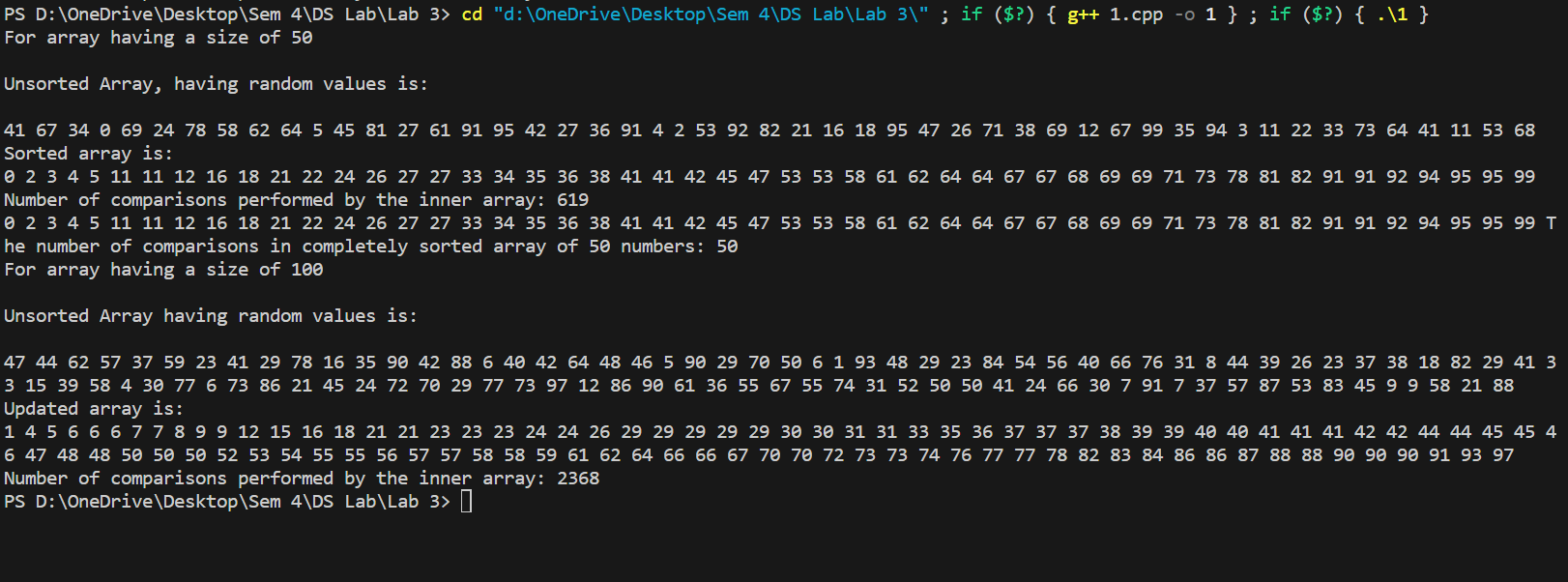
    }

    cout << endl << "Number of comparisons performed by the inner array: " << count1;

return 0;

}

**Output:**

****

**Point 5 Answer:** Yes, we can absolutely reduce the number of comparisons for the sorted array by introducing flags. At first, we will assign the false value to it, and then assign true value if it’s executing for loop. Moreover, also if statement will check if the array has already been sorted.

**Task 2:**

#include<iostream>

#include<cstdlib>

using namespace std;

int main()

{

    cout << endl << "For array having a size of 50" << endl << endl << endl;

    int arr[50],random,count=0;

    cout << "Unsorted Array, having random values is:" << endl << endl;

    for (int i=0;i<50;i++){

        random=rand() % 100;

        arr[i]=random;

        cout << arr[i] << " ";

    }

    for (int j=0;j<49;j++){

        int min\_index=j;

        for (int k=j+1;k<50;k++){

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

                min\_index=k;

                count++;

            }

        }

        swap(arr[min\_index],arr[j]);

    }

    cout << endl << "Sorted array is:" << endl << endl;

    for (int i=0;i<50;i++){

        cout << arr[i] << " ";

    }

    cout << endl << endl << "Number of comparisons performed by the inner array: " << count << endl << endl;

    int comparisons=0;

    for (int i=0;i<50;i++){

        cout << arr[i] << " ";

        comparisons++;

    }

    cout << "The number of comparisons in completely sorted array of 50 numbers: " << comparisons;

    cout << endl << "For array having a size of 100" << endl << endl << endl << endl;

    int arr1[100],random1,count1=0;

    cout << "Unsorted Array having random values is:" << endl << endl;

    for (int i=0;i<100;i++){

        random1=rand() % 100;

        arr1[i]=random1;

        cout << arr1[i] << " ";

    }

    for (int j=0;j<99;j++){

        int min\_index=j;

        for (int k=j+1;k<100;k++){

            if (arr1[k]<arr1[min\_index]){

                min\_index=k;

                count1++;

            }

        }

        swap(arr1[min\_index],arr1[j]);

    }

    cout  << endl << endl << "Updated array is:" << endl << endl;

    for (int i=0;i<100;i++){

        cout << arr1[i] << " ";

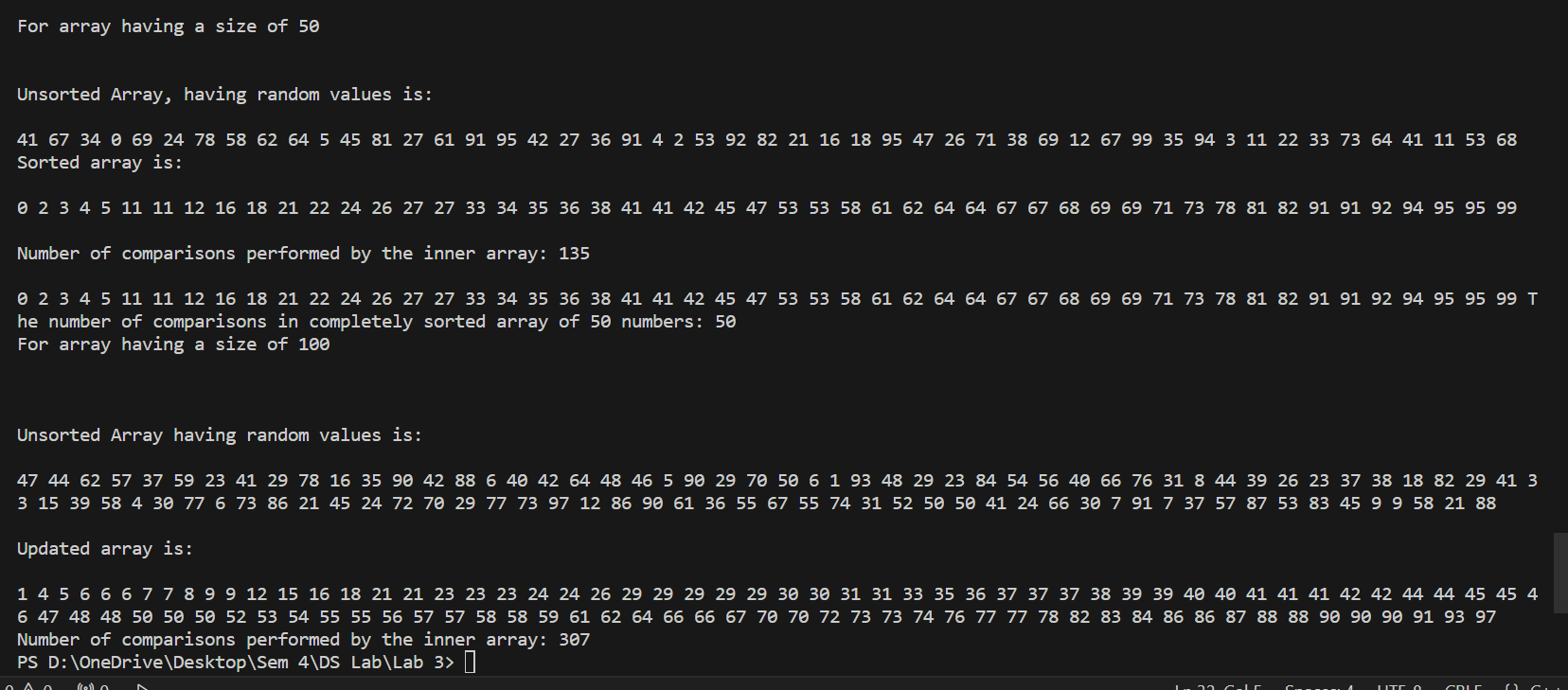
    }

    cout << endl << "Number of comparisons performed by the inner array: " << count1;

return 0;

}

**Output:**

****

**Q.4 Answer:** Comparing between bubble sort and selection sort, the selection sort works better. As you can see, it requires less comparisons/ iterations than bubble sort.

**Q.6 Answer:** No, I don’t think so that it is possible to reduce the number of comparisons for the sorted array, as it is already checking each and every element which is necessary to conclude the results. As far as we have studied, if we do it from any other method, the time and space complexity will still remain the same.

**Task 3:**

#include<iostream>

#include<cstdlib>

using namespace std;

int main()

{

    cout << endl << "For array having a size of 50" << endl << endl << endl;

    int key,arr[50],random,count=0;

    cout << "Unsorted Array, having random values is:" << endl << endl;

    for (int i=0;i<50;i++){

        random=rand() % 100;

        arr[i]=random;

        cout << arr[i] << " ";

    }

    for(int i=1;i<50;i++){

        key=arr[i];

        int j=i-1;

        while(j>=0 && arr[j]>key){

            arr[j+1]=arr[j];

            j=j-1;

            count++;

        }

        arr[j+1]=key;

    }

    cout << endl << "Sorted array is:" << endl << endl;

    for (int i=0;i<50;i++){

        cout << arr[i] << " ";

    }

    cout << endl << endl << "Number of comparisons performed by the inner array: " << count << endl << endl;

    int comparisons=0;

    for (int i=0;i<50;i++){

        cout << arr[i] << " ";

        comparisons++;

    }

    cout << "The number of comparisons in completely sorted array of 50 numbers: " << comparisons;

    cout << endl << "For array having a size of 100" << endl << endl << endl;

    int arr1[100],random1,count1=0;

    cout << "Unsorted Array having random values is:" << endl << endl;

    for (int i=0;i<100;i++){

        random1=rand() % 100;

        arr1[i]=random1;

        cout << arr1[i] << " ";

    }

        for(int i=1;i<100;i++){

        key=arr1[i];

        int j=i-1;

        while(j>=0 && arr1[j]>key){

            arr1[j+1]=arr1[j];

            j=j-1;

            count1++;

        }

        arr1[j+1]=key;

    }

    cout  << endl << endl << "Updated array is:" << endl << endl;

    for (int i=0;i<100;i++){

        cout << arr1[i] << " ";

    }

    cout << endl << "Number of comparisons performed by the inner array: " << count1;

    cout << endl << "Unsorted array having half sorted values and half unsorted: " << endl << endl;

    for (int i=0;i<25;i++){

        random1=rand() % 100;

        arr[i]=random1;

        cout << arr[i] << " ";

    }

    int count2=0;

    // cout << "Sorted Array" << endl;

    for(int i=1;i<50;i++){

        key=arr[i];

        int j=i-1;

        while(j>=0 && arr[j]>key){

            arr[j+1]=arr[j];

            j=j-1;

            count2++;

        }

        arr[j+1]=key;

    }

    cout << endl << "Sorted array is:" << endl << endl;

    for (int i=0;i<50;i++){

        cout << arr[i] << " ";

    }

    cout << endl << endl << "Number of comparisons performed by the inner array: " << count2 << endl << endl;

    return 0;

}

**Output:**

****

**Q.4 Answer:** In terms of performance, insertion sort tends to perform better than selection sort and bubble sort for small to medium-sized lists or when the array is nearly sorted.

**Q.7 Answer:** Yes, it does take advantage of already sorted data, as observed in the array in which we took half sorted array and half unsorted

**Task 4:**

#include<iostream>

#include<cstdlib>

using namespace std;

int main()

{

    int N,K,sum=0,count=0;

    cout << "Enter N" << endl;

    cin >> N;

    cout << "Enter K" << endl;

    cin >> K;

    int arr[100];

    for (int i=0;i<N;i++) {

        cout << "Input the value " << i+1 << " of the array" << endl;

        cin >> arr[i];

    }

    for (int j = 0; j < N - 1; j++) {

        bool swapped = false;

        for (int k=0;k<N-j-1;k++) {

            if (arr[k]>arr[k+1]) {

                swap(arr[k], arr[k+1]);

                swapped=true;

            }

        }

        if (!swapped)

            break;

    }

    for (int i=0;i<N;i++) {

        if (sum+arr[i]<=K) {

            sum+=arr[i];

            count++;

        } else {

            break;

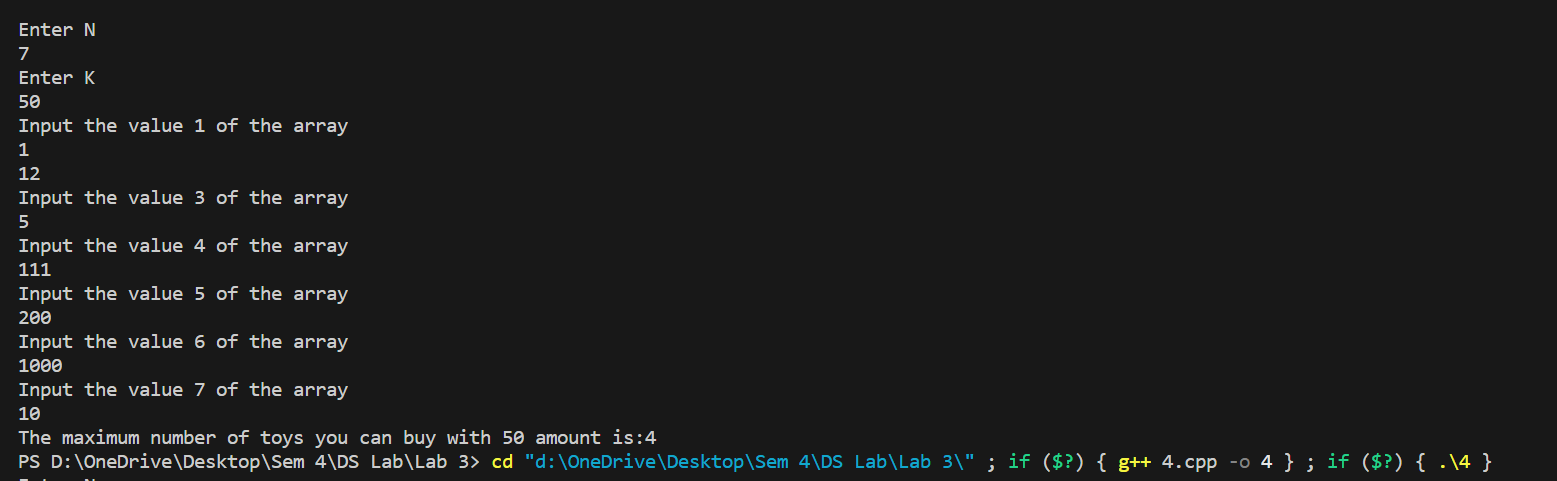
        }

    }

    cout << "The maximum number of toys you can buy with " << K << " amount is:" << count;

}

**Output:**

****

**Task 5:**

#include <iostream>

#include <algorithm>

using namespace std;

string frequencySort(string s) {

    int freq[256]={0};

    for (int i=0;i<s.size();++i) {

        freq[s[i]]++;

    }

    sort(s.begin(), s.end(), [&](char a, char b) {

        return freq[a] > freq[b] || (freq[a] == freq[b] && a < b);

    });

    return s;

}

int main() {

    string s;

    cout << "Enter the string: ";

    cin >> s;

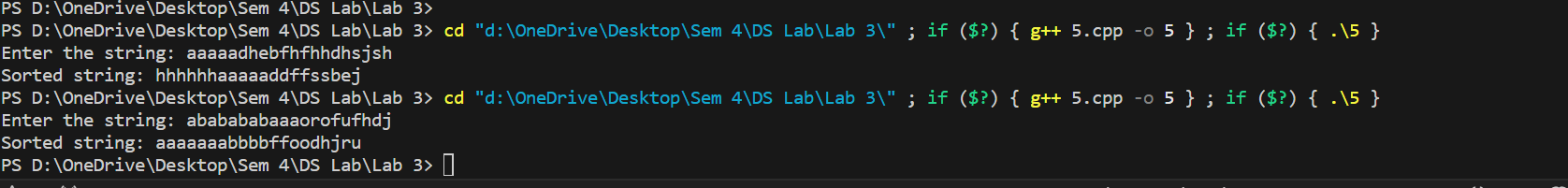
    string sortedString = frequencySort(s);

    cout << "Sorted string: " << sortedString << endl;

    return 0;

}

**Output:**

****

**Task 6:**

We will use insertion sort for this purpose, because insertion sort has better average-case performance and adaptability to partially sorted lists, as may be some of the books have already been sorted.

#include <iostream>

using namespace std;

struct Book {

    string title;

    int dueDate;

};

void insertionSort(Book books[], int n) {

    for (int i=1;i<n;++i) {

        Book key=books[i];

        int j=i-1;

        while (j>=0 && books[j].dueDate>key.dueDate) {

            books[j+1]=books[j];

            j=j-1;

        }

        books[j+1]=key;

    }

}

int main() {

    const int num = 5;

    Book books[num] = {

        {"Book 1", 5},

        {"Book 2", 3},

        {"Book 3", 7},

        {"Book 4", 1},

        {"Book 5", 4}

    };

    cout << "Books before sorting:" << endl;

    for (int i=0;i<num;++i) {

        cout << books[i].title << " Due Date:" << books[i].dueDate << endl;

    }

    insertionSort(books, num);

    cout << "\nBooks after sorting:" << endl;

    for (int i=0;i<num;++i) {

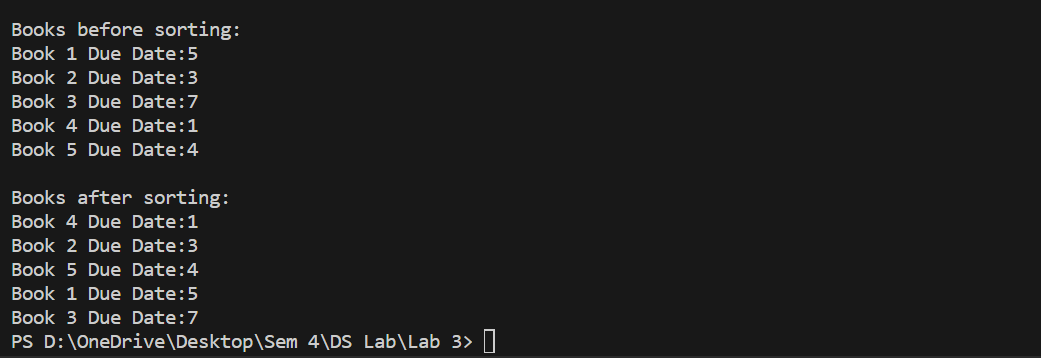
        cout << books[i].title << " Due Date:" << books[i].dueDate << endl;

    }

    return 0;

}

**Output:**

****

**Task 7:**

Insertion sort will be used in this kind of situation, as we’re adding books to a shelf one by one, and ensuring after adding each book from the very first to the latest one that are in order based on their due dates.

#include <iostream>

using namespace std;

struct Book {

    string title;

    int dueDate;

};

void insertionSort(Book books[], int n) {

    for (int i=1;i<n;++i) {

        Book key=books[i];

        int j=i-1;

        while (j>=0 && books[j].dueDate>key.dueDate) {

            books[j+1]=books[j];

            j=j-1;

        }

        books[j+1]=key;

    }

}

int main() {

    const int num = 5;

    Book books[num] = {

        {"Book 1", 5},

        {"Book 2", 3},

        {"Book 3", 7},

        {"Book 4", 1},

        {"Book 5", 4}

    };

    cout << "Books before sorting:" << endl;

    for (int i=0;i<num;++i) {

        cout << books[i].title << " Due Date:" << books[i].dueDate << endl;

    }

    insertionSort(books, num);

    cout << "\nBooks after sorting:" << endl;

    for (int i=0;i<num;++i) {

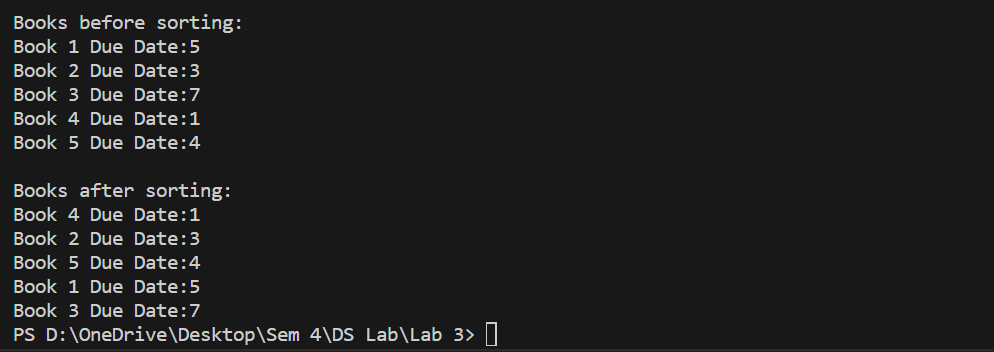
        cout << books[i].title << " Due Date:" << books[i].dueDate << endl;

    }

    return 0;

}

**Output:**

****