1. Dr. B.R. Ambedkar National Institute of Technology, Jalandhar

Data Structure Lab

ITDC0233

Submitted To:

Dr Nisha Chaurasia

Assistant Professor

Department of IT

Dr B.R. Ambedkar NIT Jalandhar, Punjab

Submitted By:

Navdeep Singh

24234073 Group 3

Branch-IT

INDEX

LAB ASSIGNMENT 1
SORTING

1 Given an array nums with n objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue. We will use the integers: 0 to represent red 1 to represent white 2 to represent blue

Approach 1 Using Bubble Sort

How it Works : Bubble Sort repeatedly compares adjacent elements and swaps them if they are in the wrong order.For an array of 0s, 1s, and 2s, it keeps swapping 2s to the right and 0s to the left in each pass.After multiple passes, all 0s come first, then 1s, and 2s move to the end, sorting the array.

```
#include <bits/stdc++.h>
using namespace std;

void bubbleSort(vector<int>&arr){
int n = arr.size();
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], arr[j+1]);
                  }
            }
      }
}
```

Approach 2 Selection Sort

How it Works : Selection Sort repeatedly finds the **minimum element** from the unsorted part and places it at the beginning. For an array of 0s, 1s, and 2s, it picks the smallest (0 first, then 1) and moves it to the correct position in each pass. After all passes, the array is sorted with 0s first, then 1s, and 2s at the end.

```
void selectionSort(vector<int>&arr){
    int n = arr.size();
    for(int i=0; i<n-1; i++){
        int minIndex = i;
        for(int j=i+1; j<n; j++){
            if(arr[j] < arr[minIndex]) minIndex = j;
        }
        swap(arr[i], arr[minIndex]);
    }
}
```

Approach 3 Insertion Sort

How it Works:
Insertion Sort builds the sorted array one element at a time by taking each element and placing it in its correct position among the already sorted elements.For an array of 0s, 1s, and 2s, it moves each 0 or 1 to the left of larger elements in each pass.
After all passes, the array is sorted with 0s first, then 1s, and 2s at the end.

```
void insertionSort(vector<int>&arr){
    int n = arr.size();
    for(int i=1; i<n; i++){
        int min = arr[i];
        int prev = i - 1;
        while(prev >= 0 && arr[prev] > min){
            arr[prev + 1] = arr[prev];
            prev--;
        }
        arr[prev + 1] = min;
    }
}
```

Output Using any od Bubble Sort Selection Sort And Insertion Sort

```
Enter the number of objects: 3
Enter the objects color (Red, White, Blue) in any case combination:
Enter color of object 1 out of 3 : Red
Enter color of object 2 out of 3 : White
Enter color of object 3 out of 3 : Blue
Your sorted array of object is :
Red White Blue
```

Approach 4 Using Merge Sort

How it Works : Merge Sort is a **divide and conquer** algorithm that splits the array into two halves, recursively sorts each half, and then merges them into a sorted array. For an array of 0s, 1s, and 2s, it keeps dividing and merging, ensuring that 0s come first, then 1s, and 2s at the end.

Better Complexity:

- Merge Sort has a time complexity of **O(n log n)**, which is much faster than Bubble, Selection, and Insertion Sort (**O(n^2)**) for larger arrays.

- Space complexity is **O(n)** due to temporary arrays used during merging.

```
vector<int>mergeTwoSortedArray(vector<int>&arr1,vector<int>&arr2){
    int i = 0;
    int j = 0;
    int k = 0;
    int size1 = arr1.size();
    int size2 = arr2.size();
    vector<int>arr3(size1+size2);
    while(i<size1 && j<size2){
        if(arr1[i] < arr2[j]) arr3[k++] = arr1[i++];
        else arr3[k++] = arr2[j++];
    }
    while(i<size1) arr3[k++] = arr1[i++];
    while(j<size2) arr3[k++] = arr2[j++];
    return arr3;
}

vector<int> mergeSort(vector<int>&arr,int start,int end){
    int mid = (start+end)/2;
    if(start<=end) return {arr[start]};
    vector<int>arr1 = mergeSort(arr,start,mid);
    vector<int>arr2 = mergeSort(arr,mid+1,end);
    return mergeTwoSortedArray(arr1,arr2);
}
```

Approach 5 : Counting Sort

How it Works : Count the number of 0s, 1s, and 2s in the array first. Then, overwrite the array by putting all 0s first, followed by 1s, and then 2s.

Better Complexity:

- Time Complexity: **O(n)** (single pass to count + single pass to overwrite)
- Space Complexity: **O(1)** if only three counters are used, much better than Bubble, Selection, or Insertion Sort.

```
// sort array of 0s 1s 2s
// basic approach count no of 0s 1s 2s basically and add in arr
// good approach 0n is DNF 3 pointer

void countAndArrangeInArray(vector<int>& arr) {
    int cntZero = 0, cntOne = 0, cntTwo = 0;
    for (int i : arr) {
        if (i == 0) cntZero++;
        else if (i == 1) cntOne++;
        else if (i == 2) cntTwo++;
    }
    int i = 0;
    while (cntZero--) arr[i++] = 0;
    while (cntOne--) arr[i++] = 1;
    while (cntTwo--) arr[i++] = 2;
}
```

Approach 6 : Dutch National Flag (DNF) or Three Pointers Approach

How it Works:
DNF Algorithm sorts 0s, 1s, and 2s in a **single pass** using three pointers: `low` for 0s, `mid` for current element, and `high` for 2s.

- If `arr[mid]` is 0 → swap with `arr[low]` and move `low` and `mid` forward.
- If `arr[mid]` is 1 → just move `mid` forward.
- If `arr[mid]` is 2 → swap with `arr[high]` and move `high` backward.
  After the pass, all 0s, 1s, and 2s are in order.

Complexity:

- Time Complexity: **O(n)** (single pass)
- Space Complexity: **O(1)** (no extra space) — the most efficient approach for this problem.

```
void printArr(vector<int>arr){
    for(auto i : arr) cout<<i <<" ";
    cout<<endl;
}
// 3 pointers
void dnfAlgo(vector<int>&arr){
    int start = 0;
    int mid = 0;
    int end = arr.size()-1;
    while(mid<=end){
        if(arr[mid] == 0){
            swap(arr[start],arr[mid]);
            printArr(arr);
            start++;
            mid++;
        }
        else if(arr[mid] == 1){
            printArr(arr);
            mid++;
        }
        else{
            swap(arr[mid],arr[end]);
            printArr(arr);
            end--;
        }
    }
}
```

Output Window for DNF appoach after every iteration we print arr for better understanding

```
Enter the number of objects: 6
Enter the objects color (Red, White, Blue) in any case combination:
Enter color of object 1 out of 6 : Blue
Enter color of object 2 out of 6 : White
Enter color of object 3 out of 6 : Red
Enter color of object 4 out of 6 : White
Enter color of object 5 out of 6 : Red
Enter color of object 6 out of 6 : Blue
2 1 0 1 0 2
0 1 0 1 2 2
0 1 0 1 2 2
0 1 0 1 2 2
0 0 1 1 2 2
0 0 1 1 2 2
Your sorted array of object is :
Red Red White White Blue Blue
```

Main Functions to call all other functions

```
int colorToNumber(const string &color) {
    string temp = color;
    // Convert to lowercase for uniformity
    for(auto &c : temp) c = tolower(c);
    if(temp == "red") return 0;
    else if(temp == "white") return 1;
    else if(temp == "blue") return 2;
    else return -1; // invalid color
}


int main(){
    int n;
    cout << "Enter the number of objects: ";
    cin >> n;
    vector<int> arr(n);
    cin.ignore(); // flush newline after reading n

    cout << "Enter the objects color (Red, White, Blue) in any case 
combination:\n";

    for(int i = 0; i < n; i++){
        string color;
        cout << "Enter color of object " << i+1 << " out of " << n << " : ";
        getline(cin, color);

        int val = colorToNumber(color);
        if(val == -1){
            cout << "Invalid input! Please enter Red, White, or Blue 
only.\n";
            return 1;
        }
        arr[i] = val;
    }
    bubbleSort(arr);
    // selectionSort(arr);
    // insertionSort(arr);
    // countAndArrangeInArray(arr);
    // dnfAlgo(arr);
    // vector<int>mergeSortAns = mergeSort(arr,0,arr.size()-1);
    cout<<"Your sorted array of object is : \n";
    for(auto i : arr){
        if(i == 0) cout << "Red ";
        else if(i == 1) cout << "White ";
        else if(i == 2) cout << "Blue ";
    }
}
```

LAB ASSIGNMENT 2
ARRAYS

Q1. Take an input from the user in form of 1D array. Convert the given array in nxn form. If the items are lesser than nxn then insert '0' in the last left elements. Now based on the user input element and base address, determine the row major and column major address.

```
#include<bits/stdc++.h>
using namespace std;



// row major
int addUsingRowMajor(int i, int j, int noOfCols, int baseAddress, int rowLB
= 0, int colLB = 0) {
    int sizeOfElement = 4;
    return baseAddress + ((i - rowLB) * noOfCols + (j - colLB)) *
sizeOfElement;
}

int addUsingColMajor(int i, int j, int noOfRows, int baseAddress, int rowLB
= 0, int colLB = 0) {
    int sizeOfElement = 4;
    return baseAddress + ((j - colLB) * noOfRows + (i - rowLB)) *
sizeOfElement;
}


void printMatrix(vector<int>& arr, int n) {
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            cout<<arr[i * n + j]<<" ";
        }
        cout<<endl;
    }
}

pair<int,int>findElement(vector<int>arr,int elem){
    int n = sqrt(arr.size());
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            if(arr[i * n + j] == elem) {
                return {i, j};
            }
        }
    }
    return {-1, -1};
}

int main1(){
    int size,baseAddress;
    cout<<"Enter Number of Elements in the array : ";
    cin>>size;
    cout<<"Enter the Base Address : ";
    cin>>baseAddress;
    vector<int>arr(size);
    cout<<"Enter the Elements of the array : ";
    for(int i=0; i<size; i++) cin>>arr[i];
    // your n*n array means needed to aapend zero check current size of the
arry and nearest n*n
    int n = ceil(sqrt(size));
    arr.resize(n*n, 0);

    printMatrix(arr, n);
    // if user also enters number to get that particular address (but but
this fails when he enters zeros !!!)
    // then simple using n^2 find element and then store its i and j

    int elemI = 0, elemJ = 0;
    int elem;
    cout<<"Enter Element to find its Address : ";
    cin>>elem;
    pair<int,int> pos = findElement(arr, elem);
    elemI = pos.first;
    elemJ = pos.second;

    // add logic to take lower bound  ask user if wants to enter then yes
else by defaulty steed as 0
    int rowLB = 0, colLB = 0;
    cout<<"Enter Lower Bound of Row (default 0) : ";
    cin>>rowLB;
    cout<<"Enter Lower Bound of Column (default 0) : ";
    cin>>colLB;

    // to print address to complete matrix
    // for(int i = 0; i < n; i++){
    //     for(int j = 0; j < n; j++){
    //         int addressRowMajor = addUsingRowMajor(i, j, n, baseAddress,
rowLB, colLB);
    //         int addressColMajor = addUsingColMajor(i, j, n, baseAddress,
rowLB, colLB);
    //         cout<<"Address of element at ("<<i<<","<<j<<") in Row Major
Order : "<<addressRowMajor<<endl;
    //         cout<<"Address of element at ("<<i<<","<<j<<") in Column
Major Order : "<<addressColMajor<<endl;
    //         cout<<endl;
    //     }
    // }


    // to print address to particular matrix
    if(elemI != -1 && elemJ != -1) {
        int addressRowMajor = addUsingRowMajor(elemI, elemJ, n, baseAddress,
rowLB, colLB);
        int addressColMajor = addUsingColMajor(elemI, elemJ, n, baseAddress,
rowLB, colLB);
        cout<<"Address of element at ("<<elemI<<","<<elemJ<<") in Row Major
Order : "<<addressRowMajor<<endl;
        cout<<"Address of element at ("<<elemI<<","<<elemJ<<") in Column
Major Order : "<<addressColMajor<<endl;
    } else {
        cout<<"Element not found in the array.";
    }
}
```

Output

```
Enter Number of Elements in the array : 5
Enter the Base Address : 1000
Enter the Elements of the array : 1
2
3
4
5
1 2 3
4 5 0
0 0 0
Enter Element to find its Address : 3
Enter Lower Bound of Row (default 0) : 0
Enter Lower Bound of Column (default 0) : 0
Address of element at (0,2) in Row Major Order : 1008
Address of element at (0,2) in Column Major Order : 1024
```

Question 3 :
A company has some interns under a project. In his coding class, who are practicing problems. Given the difficulty of the problems that the students have solved in order, help the Chef identify if they are solving them in non-decreasing order of difficulty. Non-decreasing means that the values in an array are either increasing or remaining the same, but not decreasing. That is, the students should not solve a problem with difficulty d1, and then later a problem with difficulty d2, where d1 > d2. Output "Yes" if the problems are attempted in non-decreasing order of difficulty rating and "No" if not.

```
bool isOrderCorrect(vector<int>order){
    int i = 0;
    while(i < order.size() - 1){
        if(order[i] > order[i + 1]) return false;
        i++;
    }
    return true;
}

int main2(){
    // basically take input of solving levels by student order by which
problems in solved
    // ausmme as numbers from 1 to INT_MAX;
    int questions;
    cout<<"Enter Number of Questions solved by Student : ";
    cin>>questions;
    vector<int>order(questions);
    cout<<"Enter Difficulty level of questions from 1 to INT_MAX that you
solved : \n";
    for(int i=0; i<questions; i++){
        cout<<"Enter difficulty level of "<<i+1 <<" question : ";
        cin>>order[i];
    }
    if(isOrderCorrect(order)) cout<<"Student Solved question in correct
order.";
```

```
    else cout<<"Student Solved Question in Wrong order.";
}
```

Output :

```
Enter Number of Questions solved by Student : 6
Enter Difficulty level of questions from 1 to INT_MAX that you solved :
Enter difficulty level of 1 question : 34
Enter difficulty level of 2 question : 23
Enter difficulty level of 3 question : 12
Enter difficulty level of 4 question : 56
Enter difficulty level of 5 question : 67
Enter difficulty level of 6 question : 1001
Student Solved Question in Wrong order.
```