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

Data Structure Lab 1

ITDC0233

Submitted To: **Submitted By:**

Dr Nisha Chaurasia   Navdeep Singh

Assistant Professor   24234073 Group 3

Department of IT   Branch-IT

Dr B.R. Ambedkar NIT Jalandhar, Punjab

INDEX

S.No	Date	Assignment Title	Page No	Remarks
1.	07/08/2025	Lab 1 Sorting Algorithms (Sort array containing 0,1 and 2)	3 to 8	
2.				
3.				
4.				
5.				
6.				
7.				
8.				
9.				
10.				

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.

```cpp
#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.

```cpp
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(n²)**) 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.

```cpp
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

```cpp
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 ";
    }
}
```