assignment ds 3

```
#include <stdio.h>

// Function to check if an element is present in an array (Used in problem 2 and 3)
int isPresent(int arr[], int size, int element)
{
    for (int i = 0; i < size; i++)
    {
        if (arr[i] == element)
            return 1;
    }
    return 0;
}

// function for problem 1
void mergeInPlace(int a[], int m, int b[], int n)
{
    int i = m - 1, j = n - 1, k = m + n - 1;
    while (j >= 0)
    {
        if (i >= 0 && a[i] > b[j])
            a[k--] = a[i--];
        else
            a[k--] = b[j--];
    }
}

// function for problem 2 (union)
void mergeWithoutDuplicates(int a[], int m, int b[], int n, int result[], int *resSize)
{
    int k = 0;
    for (int i = 0; i < m; i++)
        result[k++] = a[i];
    for (int j = 0; j < n; j++)
    {
        if (!isPresent(a, m, b[j]))
            result[k++] = b[j];
    }
    *resSize = k;
}
```

```
// function for problem 3 (intersection)
void findIntersection(int a[], int m, int b[], int n, int result[], int *resSize)
{
    int k = 0;
    for (int i = 0; i < m; i++)
    {
        if (isPresent(b, n, a[i]))
            result[k++] = a[i];
    }
    *resSize = k;
}

// functions for problem 4
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

void letsdutchit(int arr[], int n)
{
    int lo = 0;
    int hi = n - 1;
    int mid = 0;

    while (mid <= hi)
    {
        if (arr[mid] == 0)
            swap(&arr[lo++], &arr[mid++]);
        else if (arr[mid] == 1)
            mid++;
        else
            swap(&arr[mid], &arr[hi--]);
    }
}

// function for problem 5
int countSwaps(int arr[], int n)
{
    int swapCount = 0;
```

```
    for (int i = 1; i < n; ++i)
    {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
            swapCount++;
        }
        arr[j + 1] = key;
    }

    return swapCount;
}

int main()
{
    // Test case arrays for problems 1 to 3
    int a[10] = {1, 3, 5, 7}, b[] = {2, 3, 6, 7}, m = 4, n = 4;
    int mergedResult[10], uniqueResult[10], intersectionResult[10];
    int mergedSize, uniqueSize, intersectionSize;

    // Problem 2 Test
    mergeWithoutDuplicates(a, m, b, n, uniqueResult, &uniqueSize);
    printf("Merged without duplicates: ");
    for (int i = 0; i < uniqueSize; i++)
        printf("%d ", uniqueResult[i]);
    printf("\n");

    // Problem 3 Test
    findIntersection(a, m, b, n, intersectionResult, &intersectionSize);
    printf("Intersection of arrays: ");
    for (int i = 0; i < intersectionSize; i++)
        printf("%d ", intersectionResult[i]);
    printf("\n");

    // Problem 1 Test
    mergeInPlace(a, m, b, n);
    printf("In-place merged array: ");
    for (int i = 0; i < m + n; i++)
        printf("%d ", a[i]);
```

```
    printf("\n");

    // Problem 4 Test
    int arr[] = {0, 0, 2, 0, 1, 0, 0, 2, 2, 1, 0};
    n = sizeof(arr) / sizeof(arr[0]);

    letsdutchit(arr, n);
    for (int i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Problem 5 Test
    int arr2[] = {8, 4, 2, 1};
    n = sizeof(arr2) / sizeof(arr2[0]);

    int swaps = countSwaps(arr2, n);
    printf("Number of swaps required: %d\n", swaps);

    return 0;
}
```

-
- Merged without duplicates: 1 3 5 7 2 6
  Intersection of arrays: 3 7
  In-place merged array: 1 2 3 3 5 6 7 7
  0 0 0 0 0 0 1 1 2 2 2
  Number of swaps required: 6