Name: Mrigaj Shaw
Roll No: 2024CSB041

Assignment 2

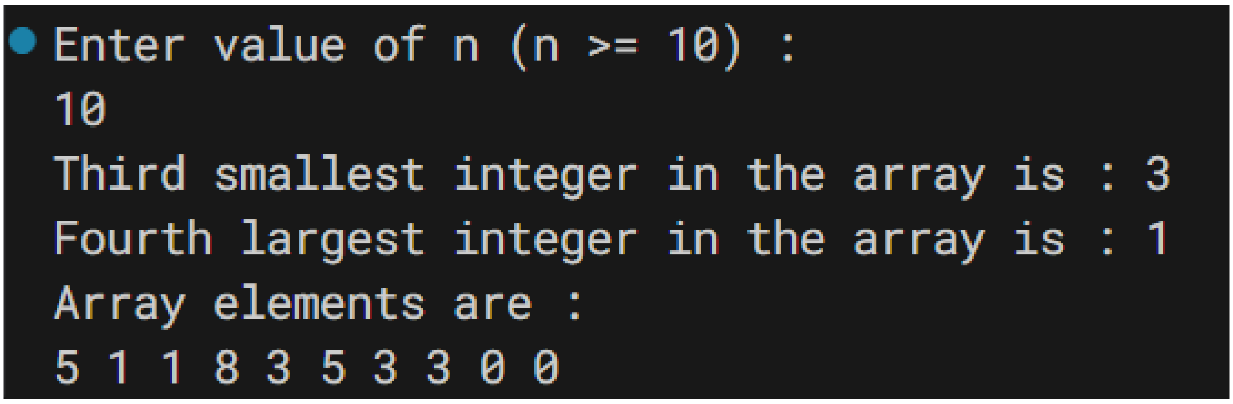
Problem 1

Problem Statement: Write a C program that dynamically creates an array of integers of size n, where n is a positive integer such that $n \geq 10$. The value of n must be read from the user at runtime. Populate the array with randomly generated integer values. After generating the array, design and implement an efficient algorithm to determine the 4th largest and the 3rd smallest elements present in the array. The solution should avoid sorting the entire array and must operate in linear time complexity, O(n). Finally, display the generated array along with the computed 4th largest and 3rd smallest elements as output.

Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>

void ThirdSmallest(int *a, int n) {
    int s1, s2, s3;
    s1 = s2 = s3 = INT_MAX;

    for(int i = 0; i < n; i++) {
        int val = a[i];
        if(val < s1) {
            s3 = s2; s2 = s1; s1 = val;
        }
        else if(val < s2 && val != s1) {
            s3 = s2; s2 = val;
        }
        else if(val < s3 && val != s1 && val != s2){
            s3 = val;
        }
    }

    printf("Third smallest integer in the array is : %d\n", s3);
}

void FourthLargest(int *a, int n) {
    int l1, l2, l3, l4;
    l1 = l2 = l3 = l4 = INT_MIN;

    for(int i = 0; i < n; i++) {
        int val = a[i];

        if(val > l1) {
            l4 = l3, l3 = l2, l2 = l1, l1 = val;
        }
        else if(val > l2 && val != l1) {
            l4 = l3, l3 = l2, l2 = val;
        }
        else if(val > l3 && val != l1 && val != l2) {
            l4 = l3, l3 = val;
        }
        else if(val > l4 && val != l1 && val != l2 && val != l3) {
            l4 = val;
        }
    }

    printf("Fourth largest integer in the array is : %d\n", l4);
}
void printArray(int *a, int n) {
    for(int i = 0; i < n; i++) printf("%d ", a[i]);
    printf("\n");
}

int main() {
    //variable declaration zone
    int n = 0;
    int *a = NULL;
    printf("Enter value of n (n >= 10) : \n");
    scanf("%d", &n);
    if(n < 10) {
        printf("n is lesser than 10, program exiting\n");
        return -1;
    }

    srand(time(NULL));

    a = (int *) malloc(n * sizeof(int));
    if(a == NULL) {
        printf("Memory allocation failed!\n");
        return -1;
    }

    for(int i = 0; i < n; i++) a[i] = rand() % 10;

    ThirdSmallest(a, n);
    FourthLargest(a, n);

    printf("Array elements are : \n");
    printArray(a, n);
}
```

Output



```
Enter value of n (n >= 10) :
10
Third smallest integer in the array is : 3
Fourth largest integer in the array is : 1
Array elements are :
5 1 1 8 3 5 3 3 0 0
```

Problem 2

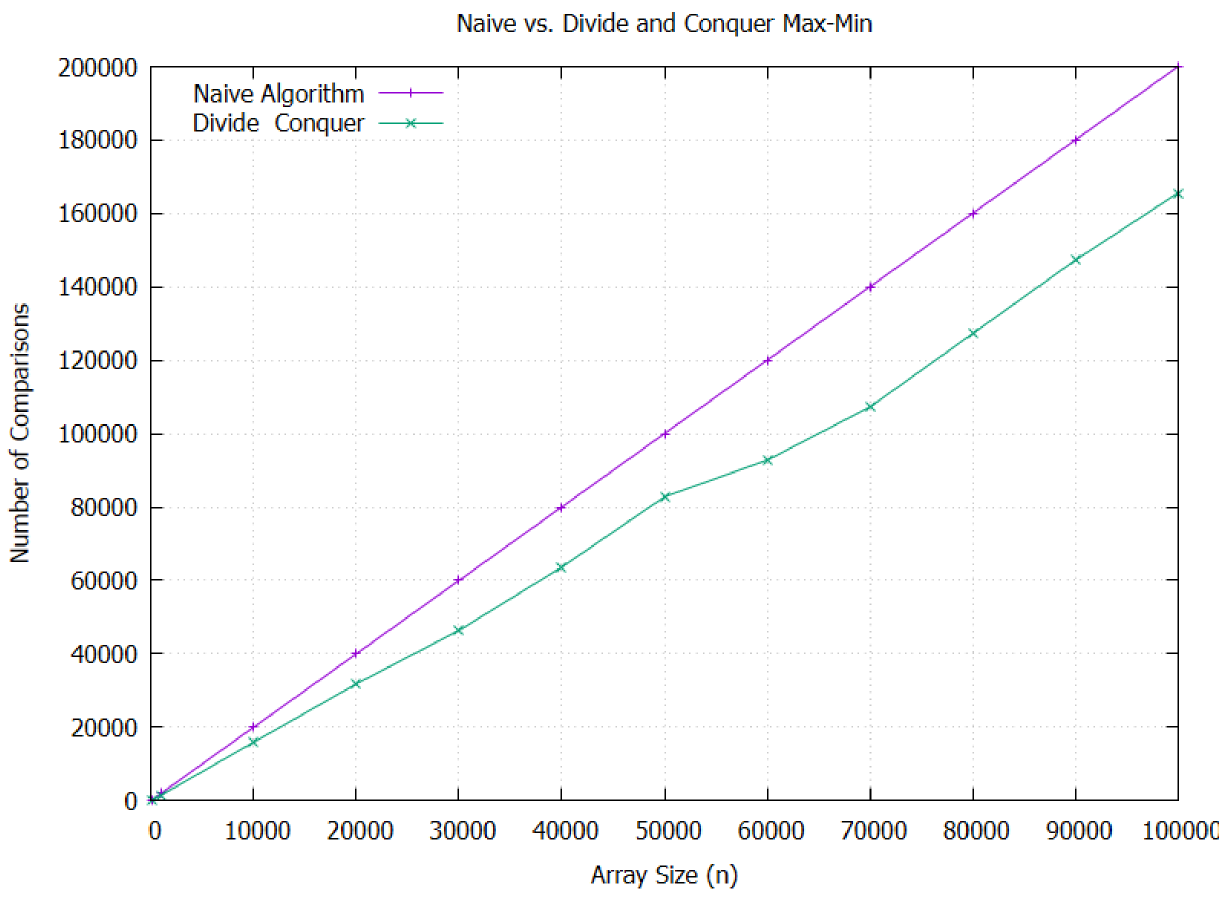
Problem Statement: In this assignment, you are required to develop two separate programs to determine the maximum and minimum elements in an array of n integers. The first program should implement the naïve Max–Min algorithm, which finds the maximum and minimum values by scanning the array sequentially. The second program should implement the Max–Min algorithm using the Divide and Conquer strategy, where the array is recursively divided into smaller subarrays and the maximum and minimum elements are determined efficiently. Execute both programs on randomly generated input arrays of varying sizes, specifically: n=100,1000,10,000,20,000,...,100,000. For each input size, count and record the number of element comparisons performed by each algorithm. Finally, plot a graph with the input size (n) on the X-axis and the number of comparisons on the Y-axis to visually compare the performance of the two approaches.

Code

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

typedef struct {
    int max;
    int min;
    int count;
} Result;

// functions returns number of comparisons
int naiveMaxMin(int *a, int n) {
    int max = a[0];
    int min = a[0];
    int comparisons = 0;

    for (int i = 1; i < n; i++) {
        int val = a[i];

        max = val >= max ? val : max;
        min = val <= min ? val : min;

        comparisons += 2;
    }

    // printf("Max : %d\n", max);
    // printf("Min : %d\n", min);
    return comparisons;
}

Result recursiveMaxMin(int *a, int low, int high) {
    Result result, left, right;
    int mid;

    if (low == high) { // 1 element
        result.max = a[low];
        result.min = a[low];
        result.count = 0;
        return result;
    }

    if (high == low + 1) { // 2 elements
```

```c
        if (a[low] > a[high]) {
            result.max = a[low];
            result.min = a[high];
            result.count = 1;
            return result;
        } else {
            result.max = a[high];
            result.min = a[low];
            result.count = 1;
            return result;
        }
    }

    mid = (low + high) >> 1;
    left = recursiveMaxMin(a, low, mid);
    right = recursiveMaxMin(a, mid + 1, high);

    result.max = left.max > right.max ? left.max : right.max;
    result.min = left.min < right.min ? left.min : right.min;
    result.count = left.count + right.count + 2;

    return result;
}
void insertSizes(int *a) {
    int i = 0;
    for (; i < 3; i++)
        a[i] = pow(10, i + 2);
    for (; i < 12; i++)
        a[i] = a[i - 1] + 10000;
}

void populateArray(int *a, int n) {
    for (int i = 0; i < n; i++)
        a[i] = rand();
}

void printArray(int *a, int n) {
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
}

int main() {
    int n = 12;
    int sizes[n];
    Result result;

    insertSizes(sizes); // Sizes are 100, 1000, 10000, 20000, .... , 100,000

    sizes[1] = 1024;
    // printf("| %-10s | %-25s | %-15s\n", "Size", "Naive MaxMin(Comparisons)",
    //        "Recursive MaxMin(Comparisons)");
    // printf
   ("--------------------------------------------------------------------"
    //        "---\n");

    for (int i = 0; i < n; i++) {
        int *a = (int *)malloc(sizes[i] * sizeof(int));
        if (a == NULL) {
            printf("Memory allocation failed!\n");
            return -1;
        }

        populateArray(a, sizes[i]);
        int naive_count = naiveMaxMin(a, sizes[i]);
        result = recursiveMaxMin(a, 0, sizes[i] - 1);
        // printf("| %-10d | %-25d | %-15d\n", sizes[i], naive_count,
        // result.count);
        printf("%d %d %d\n", sizes[i], naive_count, result.count);

        free(a);
    }

    printf("-------------------------------------------------------------------"
           "---\n");
    return 0;
}
```

Output



Problem 3

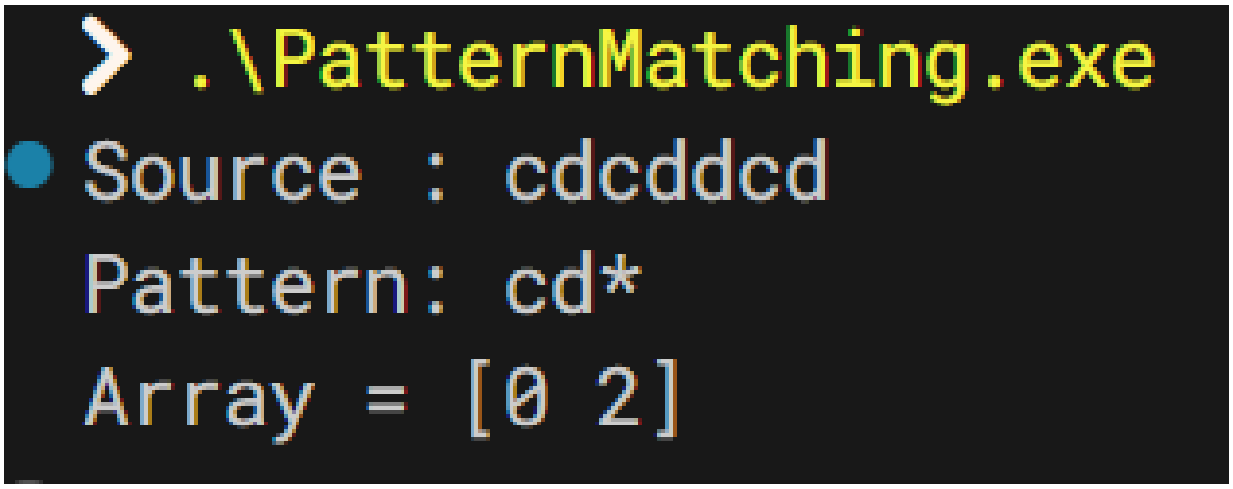
Problem Statement: Consider a source string S[0...n-1] of length n, composed exclusively of the symbols 'c' and 'd'. You are also given a pattern string P[0...m-1] of length m, where m ¡¡ n. The pattern string consists of the symbols 'c', 'd', and the special character '*'. The symbol '*' acts as a wildcard and can match any single character, either 'c' or 'd', in the source string. All other symbols in the pattern must match the corresponding characters in the source string exactly. The objective is to identify all valid match positions in the source string where the pattern occurs. A position j is considered a valid match if the pattern P matches the substring S[j...j+m-1]. The output should be a sorted list M containing all such valid starting indices j in the source string.

Example: If the source string is S = c d c d d c d and the pattern is P = c d *, then the pattern matches the source string starting at positions 0 and 2. Therefore, the output list M should be [0, 2].

Code

```c
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int *findMatches(char *source, char *pattern, int *match_count) {
    int n = strlen(source);
    int m = strlen(pattern);

    *match_count = 0;

    if (m > n || m == 0)
        return NULL;

    int *results = (int *)malloc(n * sizeof(int));
    // maximum number of matches is n (source = ...., pattern = *, all
    // characters match)

    if (results == NULL) {
        printf("Memory allocation failed");
        return NULL;
    }

    // Sliding window approach
    // iterating upto index n - m

    for (int i = 0; i <= n - m; i++) {
        bool is_match = true;

        for (int j = 0; j < m; j++) {
            char source_Character = source[i + j];
            char pattern_Character = pattern[j];

            if (pattern_Character != '*' &&
                pattern_Character != source_Character) {
                is_match = false;
                break;
            }
        }

        if (is_match) {
            results[*match_count] = i;
            (*match_count)++;
        }
    }
    return results;
}

void printArray(int *a, int n) {
    printf("Array = [");

    for (int i = 0; i < n; i++) {
        printf("%d", a[i]);
        if (i < n - 1) {
            printf(" ");
        }
    }
    printf("]");
    printf("\n");
}

int main() {
    char *source = "cdcddcd";
    char *pattern = "cd*";

    printf("Source : %s\n", source);
    printf("Pattern: %s\n", pattern);

    int count = 0;

    int *matches = findMatches(source, pattern, &count);

    if (matches != NULL) {
        printArray(matches, count);
    } else {
        printf("No matches found or invalid input\n");
    }

    free(matches);
    return 0;
}
```

Output



```
> .\PatternMatching.exe
Source : cdcddcd
Pattern: cd*
Array = [0 2]
```

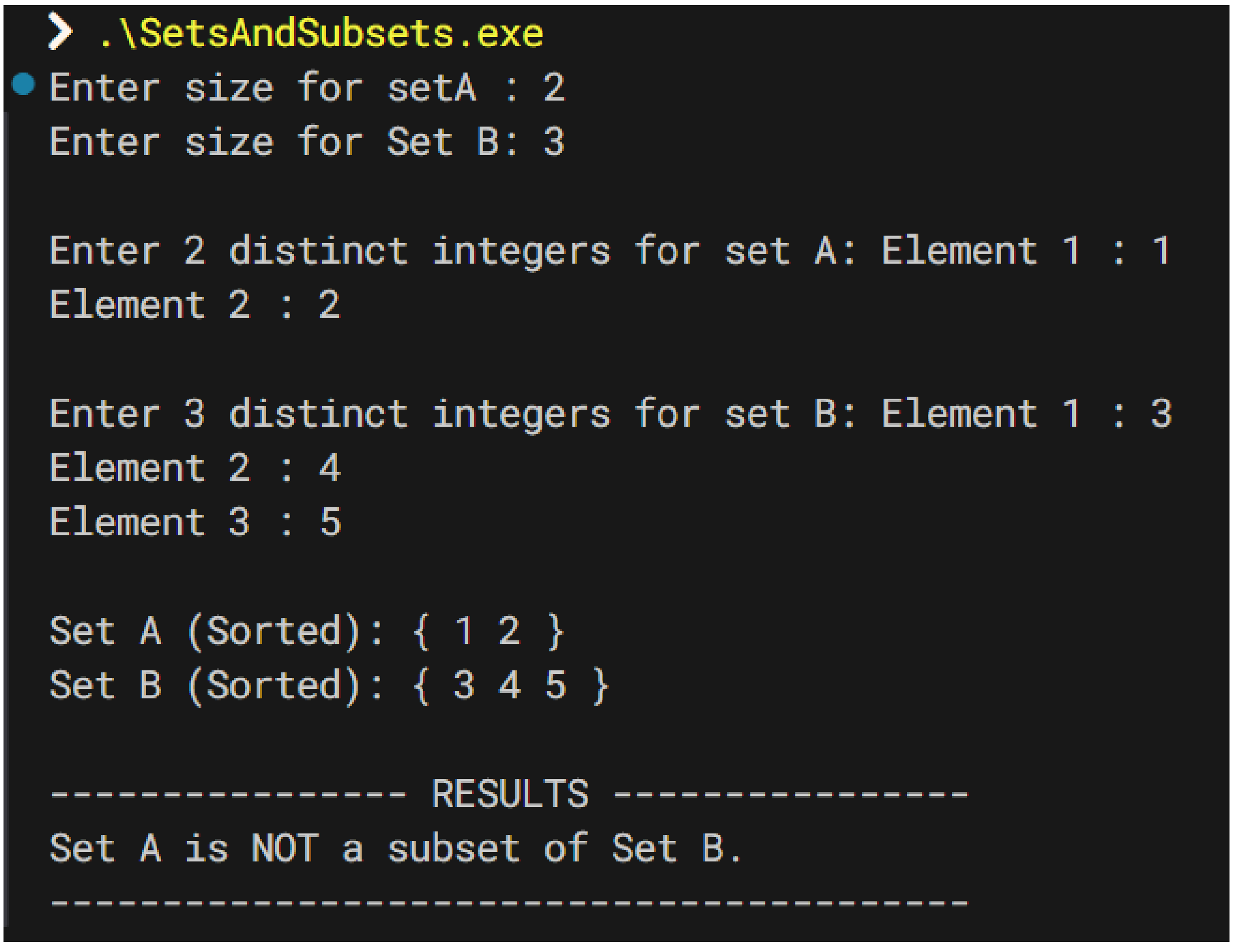
Problem 4

Problem Statement: Design an algorithm to create two sets of integers, namely A and B. The number of elements in each set must be entered by the user at runtime. While constructing the sets, ensure that each set contains only distinct elements, thereby preventing the insertion of duplicate values. After successfully creating the two sets, develop a procedure to determine whether set A is a subset of set B. Additionally, verify whether A is a proper subset of B.

Code

```c
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

bool binarySearch(int *a, int n, int target) {
    int left = 0, right = n - 1;

    while (left <= right) {
        int mid = (left + right) >> 1;

        if (a[mid] == target)
            return true;
        if (a[mid] < target)
            left = mid + 1;
        else
            right = mid - 1;
    }

    return false;
}

void insertSorted(int *a, int *currentSize, int value) {
    int i = *currentSize - 1;

    while (i >= 0 && a[i] > value) {
        a[i + 1] = a[i];
        i--;
    }

    a[i + 1] = value;
    (*currentSize)++;
}

void fillSet(int *a, int n, char name) {
    int input;
    int count = 0;

    printf("Enter %d distinct integers for set %c: ", n, name);

    while (count < n) {
        printf("Element %d : ", count + 1);
        scanf("%d", &input);

        if (count > 0 && binarySearch(a, count, input)) {
            printf("[ERROR] Duplicate! Try again.\n");
        } else
            insertSorted(a, &count, input);
    }

    printf("\n");
}

bool checkSubset(int *setA, int nA, int *setB, int nB) {
    if (nA > nB)
        return false;

    int i = 0, j = 0;

    while (i < nA && j < nB) {
        if (setA[i] < setB[j]) {
            return false;
        } else if (setA[i] == setB[j]) {
            i++;
            j++;
        } else
            j++;
    }

    return (i == nA);
}

int main() {
    int nA, nB;

    printf("Enter size for setA : ");
    scanf("%d", &nA);
    printf("Enter size for Set B: ");
    scanf("%d", &nB);
    printf("\n");

    int *setA = (int *)malloc(nA * sizeof(int));
    int *setB = (int *)malloc(nB * sizeof(int));

    fillSet(setA, nA, 'A');
    fillSet(setB, nB, 'B');

    printf("Set A (Sorted): { ");
    for (int i = 0; i < nA; i++)
        printf("%d ", setA[i]);
    printf("}\n");

    printf("Set B (Sorted): { ");
    for (int i = 0; i < nB; i++)
        printf("%d ", setB[i]);
    printf("}\n\n");

    bool isSubset = checkSubset(setA, nA, setB, nB);
    bool isProperSubset = isSubset && (nA < nB);

    printf("---------------- RESULTS ----------------\n");
    if (isSubset) {
        printf("Set A is a SUBSET of Set B.\n");
        if (isProperSubset)
            printf("Set A is a PROPER SUBSET.\n");
        else
            printf("Set A is NOT a proper subset (Identical sets).\n");
    } else {
        printf("Set A is NOT a subset of Set B.\n");
    }
    printf("----------------------------------------\n");

    free(setA);
    free(setB);
    return 0;
}
```

Output



```
> .\SetsAndSubsets.exe
Enter size for setA : 2
Enter size for Set B: 3

Enter 2 distinct integers for set A: Element 1 : 1
Element 2 : 2

Enter 3 distinct integers for set B: Element 1 : 3
Element 2 : 4
Element 3 : 5

Set A (Sorted): { 1 2 }
Set B (Sorted): { 3 4 5 }

---------------- RESULTS ----------------
Set A is NOT a subset of Set B.
----------------------------------------
```

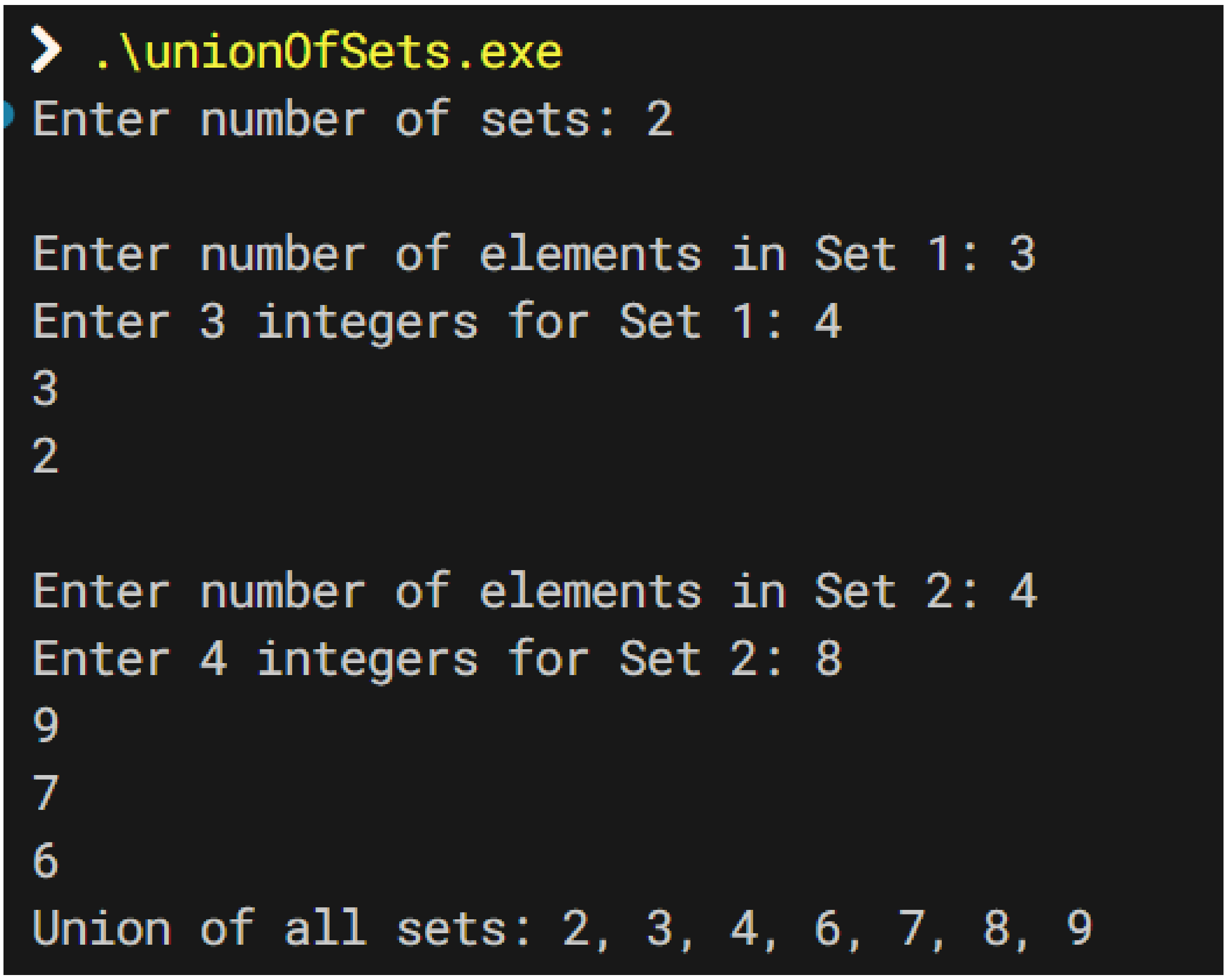
Problem 5

Problem Statement: You are given k disjoint or overlapping sets S1, S2, . . . , Sk each containing up to n integers. Design an algorithm to compute the union of all k sets.

Code

```c
#include <stdio.h>
#include <stdlib.h>

int compare(const void *a, const void *b) { return (*(int *)a - *(int *)b); }

int main() {
    int k;
    int *masterArray = NULL; // holding all elements from all arrays
    int totalElements = 0;

    printf("Enter number of sets: ");
    scanf("%d", &k);

    for (int i = 0; i < k; i++) {
        int n;
        printf("\nEnter number of elements in Set %d: ", i + 1);
        scanf("%d", &n);

        int *temp = realloc(masterArray, (totalElements + n) * sizeof(int));

        if (temp == NULL) {
            printf("Memory allocation failed!\n");
            return -1;
        }

        masterArray = temp;

        printf("Enter %d integers for Set %d: ", n, i + 1);

        for (int j = 0; j < n; j++) {
            scanf("%d", &masterArray[totalElements + j]);
        }

        totalElements += n;
    }

    qsort(masterArray, totalElements, sizeof(int), compare);

    printf("Union of all sets: ");

    if (totalElements > 0) {
        printf("%d", masterArray[0]);

        for (int i = 1; i < totalElements; i++) {
            if (masterArray[i] != masterArray[i - 1]) {
                printf(", %d", masterArray[i]);
            }
        }
    }

    free(masterArray);
}
```

Output



```
> .\unionOfSets.exe
Enter number of sets: 2

Enter number of elements in Set 1: 3
Enter 3 integers for Set 1: 4
3
2

Enter number of elements in Set 2: 4
Enter 4 integers for Set 2: 8
9
7
6
Union of all sets: 2, 3, 4, 6, 7, 8, 9
```