CS 358 - COMPILER TECHNIQUES LAB

WEEKLY ASSIGNMENT REPORT

Problem 1:

```C/C++
#include <stdio.h>

#include <stdlib.h>

#include <limits.h>


int max(int a, int b) {

    return (a > b) ? a : b;

}


int upper_bound(int* arr, int n, int x) {

    int left = 0;

    int right = n;


    while (left < right) {

        int mid = left + (right - left) / 2;

        if (x >= arr[mid]) {

            left = mid + 1;

        } else {

            right = mid;

        }
```

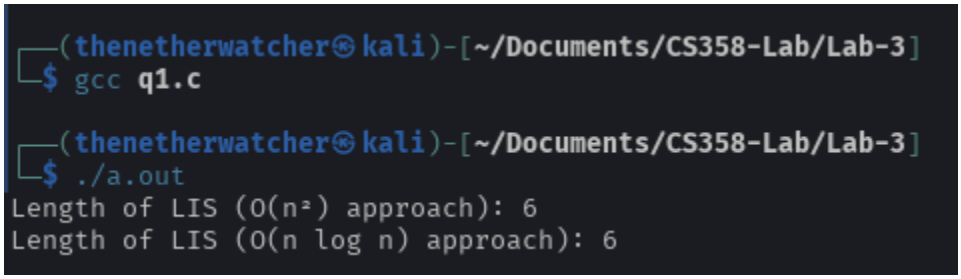
```
    }
    return left;
}

// O(n*n) approach
int find_lis_quadratic(int* a, int n) {
    int* d = (int*)malloc(n * sizeof(int));
    if (d == NULL) {
        return -1;
    }

    for (int i = 0; i < n; i++) {
        d[i] = 1;
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++) {
            if (a[j] < a[i]) {
                d[i] = max(d[i], d[j] + 1);
            }
        }
    }
```

```
    int ans = d[0];
    for (int i = 1; i < n; i++) {
        ans = max(ans, d[i]);
    }

    free(d);
    return ans;
}

// O(n log n) approach
int find_lis_optimized(int* a, int n) {
    const int INF = INT_MAX;
    int* d = (int*)malloc((n + 1) * sizeof(int));
    if (d == NULL) {
        return -1;
    }

    for (int i = 0; i <= n; i++) {
        d[i] = INF;
    }
    d[0] = INT_MIN;

    for (int i = 0; i < n; i++) {
```

```
        int pos = upper_bound(d, n + 1, a[i]);
        if (d[pos - 1] < a[i] && a[i] < d[pos]) {
            d[pos] = a[i];
        }
    }

    int ans = 0;
    for (int l = 0; l <= n; l++) {
        if (d[l] < INF) {
            ans = l;
        }
    }

    free(d);
    return ans;
}

int main() {
    int numbers[] = {10, 22, 9, 33, 21, 50, 41, 60, 80};
    int size = sizeof(numbers) / sizeof(numbers[0]);

    printf("Length of LIS (O(n²) approach): %d\n",
           find_lis_quadratic(numbers, size));
```

```
    printf("Length of LIS (O(n log n) approach): %d\n",
           find_lis_optimized(numbers, size));

    return 0;
}
```

Output:



Problem 2:

```
C/C++

#include <stdio.h>

#include <stdlib.h>


// O(n log n) approach using sorting

int find_missing_sorted(int arr[], int n) {

    for (int i = 0; i < n - 1; i++) {

        for (int j = 0; j < n - i - 1; j++) {

            if (arr[j] > arr[j + 1]) {

                int temp = arr[j];
```

```
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }

    int smallest_positive = 1;

    for (int i = 0; i < n; i++) {
        if (arr[i] <= 0) {
            continue;
        }

        if (arr[i] == smallest_positive) {
            smallest_positive++;
        } else if (arr[i] > smallest_positive) {
            return smallest_positive;
        }
    }

    return smallest_positive;
}
```

```
// O(n) approach using array marking
int find_missing_optimized(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        if (arr[i] <= 0 || arr[i] > n) {
            arr[i] = n + 1;
        }
    }

    for (int i = 0; i < n; i++) {
        int num = abs(arr[i]);
        if (num <= n) {
            arr[num - 1] = arr[num - 1] > 0 ? -arr[num - 1] : arr[num - 1];
        }
    }

    for (int i = 0; i < n; i++) {
        if (arr[i] > 0) {
            return i + 1;
        }
    }

    return n + 1;
}
```

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

int main() {
    // Test cases
    int arr1[] = {2, 3, -7, 6, 8, 1, -10, 15};
    int n1 = sizeof(arr1) / sizeof(arr1[0]);

    int arr2[] = {2, 3, -7, 6, 8, 1, -10, 15};
    int n2 = sizeof(arr2) / sizeof(arr2[0]);

    printf("Original array: ");
    printArray(arr1, n1);

    printf("Smallest missing positive (Sorting approach): %d\n",
           find_missing_sorted(arr1, n1));

    printf("Smallest missing positive (Optimized approach): %d\n",
```

```
            find_missing_optimized(arr2, n2));

    // Additional test cases
    int arr3[] = {1, 2, 3, 4, 5};
    int n3 = sizeof(arr3) / sizeof(arr3[0]);
    printf("\nTest case 2 (consecutive numbers): ");
    printArray(arr3, n3);
    printf("Smallest missing positive (Optimized approach): %d\n",
            find_missing_optimized(arr3, n3));

    int arr4[] = {-5, -3, -1, -2};
    int n4 = sizeof(arr4) / sizeof(arr4[0]);
    printf("\nTest case 3 (all negative): ");
    printArray(arr4, n4);
    printf("Smallest missing positive (Optimized approach): %d\n",
            find_missing_optimized(arr4, n4));

    return 0;
}
```

Output:

```
┌──(thenetherwatcher㉿kali)-[~/Documents/CS358-Lab/Lab-3]
└─$ gcc q2.c

┌──(thenetherwatcher㉿kali)-[~/Documents/CS358-Lab/Lab-3]
└─$ ./a.out
Original array: 2 3 -7 6 8 1 -10 15
Smallest missing positive (Sorting approach): 4
Smallest missing positive (Optimized approach): 4

Test case 2 (consecutive numbers): 1 2 3 4 5
Smallest missing positive (Optimized approach): 6

Test case 3 (all negative): -5 -3 -1 -2
Smallest missing positive (Optimized approach): 1
```

Problem 3:

C/C++

```
#include <stdio.h>

#include <string.h>

#include <ctype.h>

#include <stdlib.h>

#include <stdbool.h>


bool isAlphanumeric(char c) {

    return (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <=
'9');

}


void cleanString(const char* input, char* output) {

    int j = 0;
```

```
    for (int i = 0; input[i]; i++) {
        if (isAlphanumeric(input[i])) {
            output[j++] = tolower(input[i]);
        }
    }
    output[j] = '\0';
}

bool isPalindrome(const char* str) {
    int left = 0;
    int right = strlen(str) - 1;

    while (left < right) {
        if (str[left] != str[right]) {
            return false;
        }
        left++;
        right--;
    }
    return true;
}

void rotateLeft(char* str) {
```

```
    int len = strlen(str);
    if (len <= 1) return;

    char firstChar = str[0];
    for (int i = 0; i < len - 1; i++) {
        str[i] = str[i + 1];
    }
    str[len - 1] = firstChar;
}

bool isRotatedPalindrome(const char* input) {
    int len = strlen(input);
    char* cleanedStr = (char*)malloc((len + 1) * sizeof(char));
    if (!cleanedStr) return false;

    cleanString(input, cleanedStr);
    int cleanedLen = strlen(cleanedStr);

    if (cleanedLen == 0) {
        free(cleanedStr);
        return true;
    }
```

```
    char* rotatedStr = (char*)malloc((cleanedLen + 1) * sizeof(char));
    if (!rotatedStr) {
        free(cleanedStr);
        return false;
    }
    strcpy(rotatedStr, cleanedStr);

    for (int i = 0; i < cleanedLen; i++) {
        if (isPalindrome(rotatedStr)) {
            free(cleanedStr);
            free(rotatedStr);
            return true;
        }
        rotateLeft(rotatedStr);
    }

    free(cleanedStr);
    free(rotatedStr);
    return false;
}

int main() {
    const char* testStrings[] = {
```

```
        "ABAB",           // false
        "aaaa",           // true
        "Race a car",    // false
        "A man, a plan, a canal: Panama",  // true
        "a",             // true (single character)
        "Ab@Ba",         // true (after cleaning)
        "aab",           // true
        NULL
    };

    for (int i = 0; testStrings[i] != NULL; i++) {
        printf("String: \"%s\"\n", testStrings[i]);
        if (isRotatedPalindrome(testStrings[i])) {
            printf("IS a rotated palindrome\n\n");
        } else {
            printf("is NOT a rotated palindrome\n\n");
        }
    }

    return 0;
}
```

Output:

```
┌──(thenetherwatcher㉿kali)-[~/Documents/CS358-Lab/Lab-3]
└─$ gcc q3.c

┌──(thenetherwatcher㉿kali)-[~/Documents/CS358-Lab/Lab-3]
└─$ ./a.out
String: "ABAB"
is NOT a rotated palindrome

String: "aaaa"
IS a rotated palindrome

String: "Race a car"
is NOT a rotated palindrome

String: "A man, a plan, a canal: Panama"
IS a rotated palindrome

String: "a"
IS a rotated palindrome

String: "Ab@Ba"
IS a rotated palindrome

String: "aab"
IS a rotated palindrome
```