Algorithms Recitation 01 Assignment

Omar Muhammad Gaber
20221446137

Date: October 25, 2023

1 Exercise 6: Palindrome Check

Problem Statement:
Write a recursive method to check if a given string is a palindrome.

Sample Input: ”level”
Expected Output: true

Listing 1: Palindrome Check Subprogram

```
bool isPalindromeHelper(const string &str, int left, int right) {
    if (left >= right)
        return true;
    if (str[left] != str[right])
        return false;
    return isPalindromeHelper(str, left + 1, right − 1);
}
bool isPalindrome(const string &str) {
    return isPalindromeHelper(str, 0, str.size() − 1);
}
```

Time Complexity: $O(n)$
Auxiliary Space: $O(n)$, where n is the length of the string.

2 Exercise 7: Array Sum

Problem Statement:
Create a recursive method to find the sum of elements in an integer array.

Sample Input: [2, 4, 6, 8, 10]
Expected Output: 30

Listing 2: Array Sum Subprogram

```
template<typename T>
T ArraySum(vector<T> &arr, int length) {
    if (length == 0)
        return 0;
    return arr[length − 1] + ArraySum(arr, length − 1);
}
```

Time Complexity: $O(n)$
Auxiliary Space: $O(n)$, where n is the size of the array.

3 Exercise 8: Binary Search

Problem Statement:
Implement a recursive method for binary search on a sorted array.

Sample Input: [1, 2, 3, 4, 5, 6, 7, 8, 9], target = 5
Expected Output: 4 (index of the target element)

Listing 3: Binary Search Subprogram

```
template<typename T>
int binarySearch(vector<T> &arr, T &val, int low, int high) {
    if (low > high)
        return −1;

    int mid = low + ((high − low) / 2);
    if (arr[mid] == val)
        return mid;
    if (arr[mid] > val)
        return binarySearch(arr, val, low, mid − 1);
    return binarySearch(arr, val, mid + 1, high);
}
```

Time Complexity: $O(\log n)$
Auxiliary Space: $O(\log n)$, where n is the size of the array.

4 Exercise 9: Reverse String

Problem Statement:
Write a recursive method to reverse a given string.

Sample Input: ”hello”
Expected Output: ”olleh”

Listing 4: Reverse String Procedure

```
void swap(char &a, char &b) {
    char tempChar = a;
    a = b;
    b = tempChar;
}

void reverseString(string &str, int left, int right) {
    if (left >= right || right > str.length())
        return;

    swap(str[left], str[right]);
    reverseString(str, left + 1, right − 1);
}
```

Time Complexity: $O(n)$
Auxiliary Space: $O(n)$, where n is the length of the string.

5 Exercise 10: Tower of Hanoi

Problem Statement:
Implement the Tower of Hanoi problem using recursion.

Sample Input:
Number of disks = 3
Source = A
Auxiliary = B
Destination = C
Expected Output:
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C

Listing 5: Tower of Hanoi Procedure

```
void hanoi(int n, const string &source, const string &auxiliary, const string &destination) {
    if (n == 1)
        cout << "Move␣disk␣" << n << "␣from␣" << source << "␣to␣" << destination << '\n';
    else {
        hanoi(n - 1, source, destination, auxiliary);
        cout << "Move␣disk␣" << n << "␣from␣" << source << "␣to␣" << destination << '\n';
        hanoi(n - 1, auxiliary, source, destination);
    }
}

int main() {
    int n;
    string source, auxiliary, destination;

    cout << "Number␣of␣disks:␣\n";
    cin >> n;

    cout << "Source:␣\n";
    cin >> source;

    cout << "Auxiliary:␣\n";
    cin >> auxiliary;

    cout << "Destination:␣\n";
    cin >> destination;

    hanoi(n, source, auxiliary, destination);

    return 0;
}
```

Time Complexity:

$$T(n) = 2^n - 1$$

Auxiliary Space: $O(n)$, where n is the number of disks.

6 Coin Changing Problem

Problem Statement:
You are given an array representing different coin denominations and a total amount of money. The goal is to find the minimum number of coins needed to make up that amount. Assume an unlimited supply of coins of each denomination.

Input:

- An array `coins` representing the coin denominations, where each coin denomination is a positive integer.
- An integer `amount` representing the total amount of money to make up.

Output:

- An integer representing the minimum number of coins needed to make up the amount.
- If it's not possible to make up the amount using the given coin denominations, return -1.

Listing 6: Coin Changing Problem Dynamic Programming Approach Subprogram

```
int minCoins(vector<int>& coins, int target) {
    vector<int> dp(target + 1, INT_MAX);

    dp[0] = 0;

    for (int i = 1; i <= target; i++) {
        for (int coin : coins) {
            if (i - coin >= 0 && dp[i - coin] != INT_MAX) {
                dp[i] = min(dp[i], dp[i - coin] + 1);
            }
        }
    }

    return dp[target] == INT_MAX ? -1 : dp[target];
}
```

Time Complexity: $O(target * n)$
Auxiliary Space: $O(n)$, where n is the number of coins.