Experiment 7

Student Name: Manjot Kaur **UID: 22BCS16792**
Branch: CSE **Section: 22KPIT-901/B**
Semester: 6th **Date of Performance:20/03/2025**
Subject: AP Lab **Subject Code: 22CSH-352**

1. **Aim:** Greedy algorithms are a class of algorithms that make locally optimal choices at each step with the hope of finding a global optimum. They aim to solve optimization problems by selecting the best available option at each stage, without reconsidering previous choices. While greedy algorithms are often efficient and easy to implement, they do not always guarantee the optimal solution. They are typically used in problems like coin change, Huffman coding, and minimum spanning trees.

2. **Objective 1:** There are n children standing in a line. Each child is assigned a rating value given in the integer array ratings. You are giving candies to these children subject to the following requirements:
   - Each child must have at least one candy.
   - Children with a higher rating get more candies than their neighbors.

Return the minimum number of candies you need to distribute to the children.

3. **Code/Implementation:**

```
#include <iostream>
#include <vector>

using namespace std;

int candy(vector<int>& ratings) {
    int n = ratings.size();
    vector<int> candies(n, 1);  // Each child gets at least 1 candy

    // Left-to-right: Ensure right child gets more candy if they have a higher
rating
    for (int i = 1; i < n; i++) {
        if (ratings[i] > ratings[i - 1]) {
            candies[i] = candies[i - 1] + 1;
        }
    }

    // Right-to-left: Ensure left child gets more candy if they have a higher
rating
    for (int i = n - 2; i >= 0; i--) {
        if (ratings[i] > ratings[i + 1]) {
            candies[i] = max(candies[i], candies[i + 1] + 1);
        }
```

```
    }

    // Sum up the total candies needed
    int totalCandies = 0;
    for (int c : candies) {
        totalCandies += c;
    }

    return totalCandies;
}

int main() {
    vector<int> ratings = {1, 0, 2};
    cout << "Minimum candies required: " << candy(ratings) << endl;
    return 0;
}
```

4. **Objective 2:** You are given an integer array prices where prices[i] is the price of a given stock on the ith day. On each day, you may decide to buy and/or sell the stock. You can only hold at most one share of the stock at any time. However, you can buy it and then immediately sell it on the same day.Find and return the maximum profit you can achieve.

5. **Code/Implementation:**

```
#include <iostream>

#include <vector>

using namespace std;

int maxProfit(vector<int>& prices) {

    int profit = 0;

 for (int i = 1; i < prices.size(); i++) {

        if (prices[i] > prices[i - 1]) {

            profit += prices[i] - prices[i - 1]; // Buy at i-1, sell at i

}}

return profit;

}

int main() {

    vector<int> prices = {7, 1, 5, 3, 6, 4};

    cout << "Maximum Profit: " << maxProfit(prices) << endl;

    return 0;

      }
```

6. **Objective 3 :** Given a string s, remove duplicate letters so that every letter appears once and only once. You must make sure your result is the smallest in lexicographical order among all possible results.

7. Code/Implementation:

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <unordered_set>
#include <stack>

using namespace std;

string removeDuplicateLetters(string s) {
    vector<int> lastIndex(26, 0);  // Store last occurrence index of each character
    vector<bool> visited(26, false); // To check if a character is already in stack
    stack<char> stk;  // Monotonic stack for result

    // Store the last occurrence index of each character
    for (int i = 0; i < s.size(); i++) {
        lastIndex[s[i] - 'a'] = i;
    }

    for (int i = 0; i < s.size(); i++) {
        char ch = s[i];

        // If the character is already in the result, skip it
        if (visited[ch - 'a']) continue;

        // Maintain lexicographical order by popping larger characters if they appear later
        while (!stk.empty() && stk.top() > ch && lastIndex[stk.top() - 'a'] > i) {
            visited[stk.top() - 'a'] = false;
            stk.pop();
        }

        // Push current character to stack
        stk.push(ch);
        visited[ch - 'a'] = true;
    }

    // Build the result string from the stack
    string result;
    while (!stk.empty()) {
        result = stk.top() + result;
        stk.pop();
    }

    return result;
```

```
}

int main() {
    string s = "bcabc";
    cout << "Smallest lexicographical string: " << removeDuplicateLetters(s) << endl;
    return 0;
}
```

8. Learning Outcomes:

- Understand and apply fundamental programming concepts in C++.
- Develop efficient algorithms to solve computational problems.
- Implement and optimize data structures for various applications.
- Design and debug software using best coding practices.
- Analyze and improve the performance of implemented solutions.