DAY 8

Student Name: Suman Kumar UID: 22BCS15488
Branch: BE-CSE Section/Group: 620 - B
Date of Performance:28/12/24

Problem 1

1. Aim: N-th Tribonacci Number
2. Code:

```
#include <iostream> #include
<vector> using namespace
std; int tribonacci(int n) {     if
(n == 0) return 0;     if (n == 1
|| n == 2) return 1;
vector<int> T(n + 1);
   T[0] = 0;
   T[1] = 1;
T[2] = 1;
   for (int i = 3; i <= n; ++i) {
      T[i] = T[i - 1] + T[i - 2] + T[i - 3];
   }     return
T[n];
} int main()
{     int n;
   cout << "Enter a number n: ";     cin >> n;     int result = tribonacci(n);
cout << "The " << n << "-th Tribonacci number is: " << result << endl;

   return 0;
}
```

3. Output:

```
Enter a number n: 5
The 5-th Tribonacci number is: 7
```

Problem 2

1. Aim: Divisor Game
2. Code:

```
#include <iostream> bool
divisorGame(int n) {
return n % 2 == 0;
} int main() {     int n;     std::cout << "Enter
a number: ";     std::cin >> n;     if
(divisorGame(n)) {         std::cout << "Alice
wins!" << std::endl;
    } else {         std::cout << "Bob wins!"
<< std::endl;
    }
return 0;
}
```

3. Output:

```
Enter a number: 2
Alice wins!
```

Problem 3

1. Aim: Maximum Repeating Substring

2. Code:

```
#include <iostream> #include <string> int maxKRepeating(const
std::string& sequence, const std::string& word) {     int k = 0;     std::string
repeatedWord;     while (true) {         repeatedWord += word; //
Concatenate the word         k++; // Increment k
        // Check if the repeatedWord is a substring of the sequence
if (sequence.find(repeatedWord) == std::string::npos) {
break; // If not found, exit the loop
```

```
        }
}
    return k - 1;
} int main() {     std::string sequence = "ababc";
std::string word = "ab";     int result =
maxKRepeating(sequence, word);
    std::cout << "Maximum k-repeating value: " << result << std::endl; //
Output: 2

    return 0;
}
```

3. Output:

```
Maximum k-repeating value: 2
```

Problem 4

1. Aim: Pascal's Triangle II
2. Code:

```
#include <iostream> #include <vector> std::vector<int> getRow(int
rowIndex) {     std::vector<int> row(rowIndex + 1, 1); // Initialize
the row with 1s     for (int i = 1; i <= rowIndex; ++i) {         for (int j
= i - 1; j > 0; --j) {             row[j] = row[j] + row[j - 1];
        }
    }     return
row;
} int main() {     int rowIndex;     std::cout << "Enter the row
index: ";     std::cin >> rowIndex;     std::vector<int> result =
getRow(rowIndex);     std::cout << "Row " << rowIndex << " of
Pascal's Triangle: ";     for (int num : result) {         std::cout <<
num << " ";
```

```
    }     std::cout <<
std::endl;     return 0;
}
```

3. Output:

```
Enter the row index: 3
Row 3 of Pascal's Triangle: 1 3 3 1
```

Problem 5

1. Aim: Maximum Repeating Substring
2. Code:

```
#include <iostream> #include <string> int maxKRepeating(const
std::string& sequence, const std::string& word) {     if
(sequence.find(word) == std::string::npos) {         return 0; // Word is not a
substring
    }     int k = 0;     std::string repeatedWord;     while (true)
{         repeatedWord += word; // Concatenate the word
k++; // Increment k         if (sequence.find(repeatedWord)
== std::string::npos) {             break; // If not found, break
the loop
        }
    }
    return k - 1; // Return the maximum k found
} int main() {     std::string sequence = "ababc";     std::string word =
"ab";     int result = maxKRepeating(sequence, word);     std::cout <<
"Maximum k-repeating value: " << result << std::endl;     return 0;
}
```

3. Output:

```
Maximum k-repeating value: 2
```

Problem 6

1. Aim: Climbing Stairs
2. Code:

```
#include <iostream> #include <vector> int
climbStairs(int n) {     if (n <= 1) {         return 1; //
There is 1 way to climb 0 or 1 step
    }
    std::vector<int> dp(n + 1);     dp[0] = 1;
// 1 way to stay at the ground     dp[1] = 1;
// 1 way to reach the first step     for (int i
= 2; i <= n; ++i) {         dp[i] = dp[i - 1] +
dp[i - 2];
    }
    return dp[n]; // The number of ways to reach the nth step
} int main() {     int n;     std::cout << "Enter the number of steps: ";     std::cin
>> n;     int result = climbStairs(n);     std::cout << "Number of distinct ways
to climb to the top: " << result << std::endl;     return 0;
}
```

3. Output:

```
Enter the number of steps: 2
Number of distinct ways to climb to the top: 2
```

Problem 7

1. Aim: Best Time to Buy and Sell Stock
2. Code:

```
#include <iostream>
```

```
#include <vector> #include <algorithm> int
maxProfit(std::vector<int>& prices) {     if (prices.empty()) return
0; // If the prices array is empty, return 0     int minPrice = prices[0];
// Initialize minPrice to the first price.
    int maxProfit = 0; // Initialize maxProfit to 0.
    for (int i = 1; i < prices.size(); ++i) {
if (prices[i] < minPrice) {
minPrice = prices[i];
        } else {                int profit = prices[i] -
minPrice;                                maxProfit =
std::max(maxProfit, profit);
        }
    }
    return maxProfit; // Return the maximum profit found.
} int main() {     std::vector<int> prices =
{7, 1, 5, 3, 6, 4};     int result =
maxProfit(prices);
    std::cout << "Maximum profit: " << result << std::endl; // Output: 5
return 0;
}
```

3. Output:

```
Maximum profit: 5
```

Problem 8

1. Aim: Counting Bits
2. Code:

```
#include <iostream> #include <vector> std::vector<int> countBits(int n) {
std::vector<int> ans(n + 1, 0); // Initialize a vector of size n + 1 with all
elements set to 0     for (int i = 1; i <= n; ++i) {         ans[i] = ans[i >> 1] +
(i & 1);
```

```
    }
    return ans;
} int main() {     int n;     std::cout <<
"Enter an integer n: ";     std::cin >> n;
std::vector<int> result = countBits(n);
std::cout << "Output: [";     for (size_t i =
0; i < result.size(); ++i) {         std::cout
<< result[i];         if (i < result.size() - 1) {
std::cout << ", ";
        }
    }
    std::cout << "]" << std::endl;
return 0;
}
```

3. Output:

```
Enter an integer n: 2
Output: [0, 1, 1]
```

Problem 9

1. Aim: Is Subsequence

2. Code:

```
#include <iostream> #include <string> bool
isSubsequence(const std::string& s, const std::string& t) {
int s_len = s.length();     int t_len = t.length();     int s_index =
0; // Pointer for string s     int t_index = 0; // Pointer for string t
while (s_index < s_len && t_index < t_len) {         if
(s[s_index] == t[t_index]) {             s_index++;
        }
        t_index++;
```

```
    }
    return s_index == s_len;
} int main() {     std::string s1 = "abc";     std::string t1 = "ahbgdc";
std::cout << std::boolalpha << isSubsequence(s1, t1) << std::endl;
return 0;
}
```

3. Output:

```
true
```

Problem 10

1. Aim: Longest Palindromic Substring
2. Code:

```
#include <iostream>
#include <string>
using namespace std;
class Solution { public:
    string longestPalindrome(string s) {         if
(s.empty()) return "";         int start = 0, end = 0;
for (int i = 0; i < s.size(); i++) {             int len1 =
expandAroundCenter(s, i, i);             int len2 =
expandAroundCenter(s, i, i + 1);             int len =
max(len1, len2);             if (len > end - start) {
start = i - (len - 1) / 2;                end = i + len / 2;
            }
        }         return s.substr(start, end -
start + 1);
    }
private:
    int expandAroundCenter(const string& s, int left, int right) {
while (left >= 0 && right < s.size() && s[left] == s[right]) {
left--;             right++;
```

```
        }
        return right - left - 1; // Length of the palindrome
    } }; int main() {     Solution solution;     string s1 = "babad";     string s2
= "cbbd";     cout << "Longest palindromic substring of \"" << s1 << "\": "
<<
solution.longestPalindrome(s1) << endl;
    cout << "Longest palindromic substring of \"" << s2 << "\": " <<
solution.longestPalindrome(s2) << endl;     return 0;
}
```

3. Output:

```
Longest palindromic substring of "babad": aba
Longest palindromic substring of "cbbd": bb
```