

AP Experiment 10

Name: Shivangi Gupta

UID: 22BCS15008

Section: 601-A

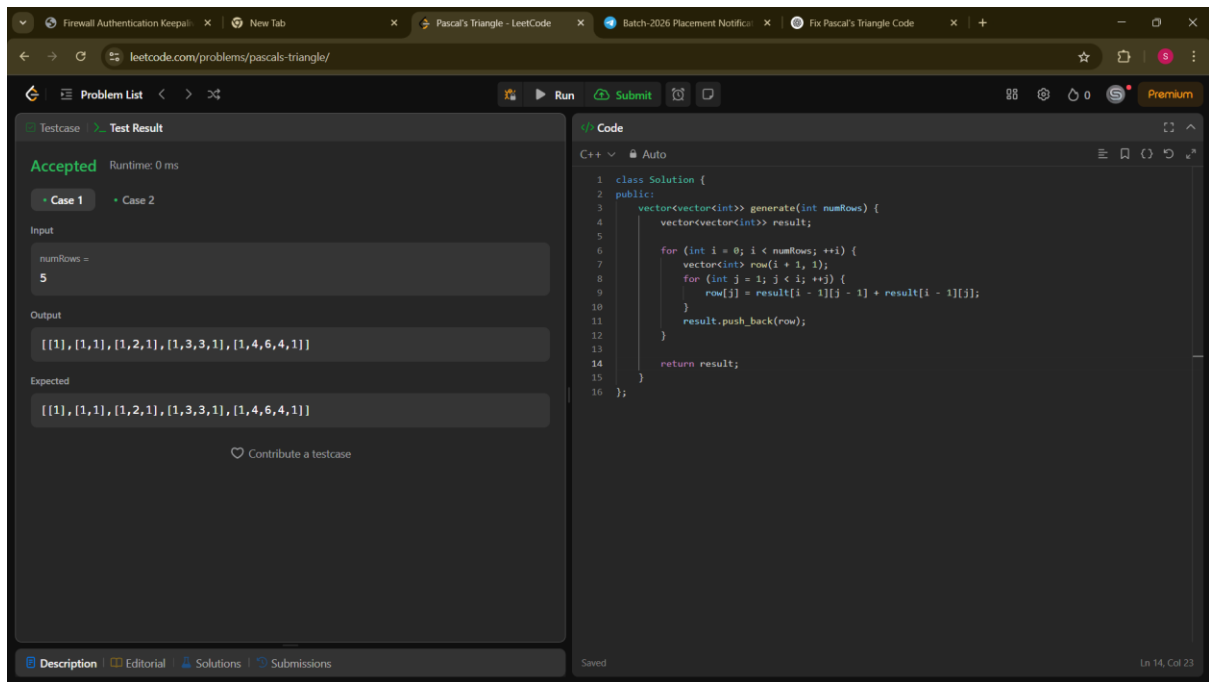
Q1. Pascal's Triangle <https://leetcode.com/problems/pascals-triangle/>

CODE:

```
class Solution {
public:
    vector<vector<int>> generate(int numRows) {
        vector<vector<int>> result;

        for (int i = 0; i < numRows; ++i) {
            vector<int> row(i + 1, 1);
            for (int j = 1; j < i; ++j) {
                row[j] = result[i - 1][j - 1] + result[i - 1][j];
            }
            result.push_back(row);
        }

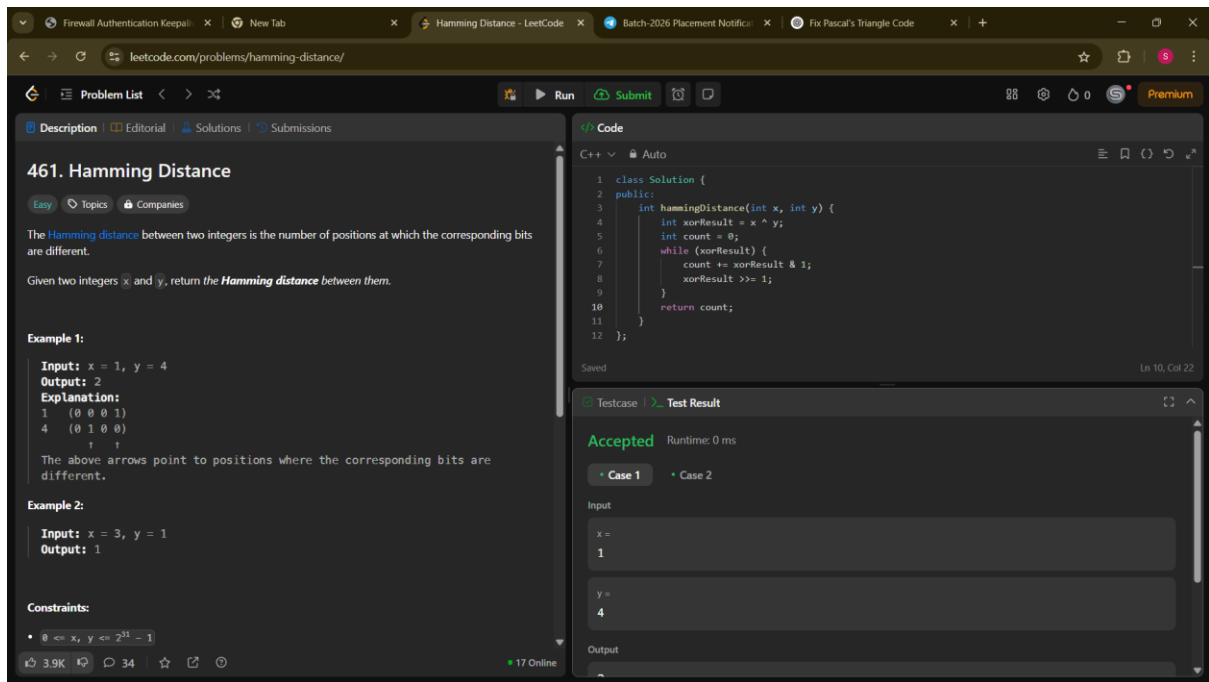
        return result;
    }
};
```



Q2. Hamming Distance <https://leetcode.com/problems/hamming-distance/>

CODE:

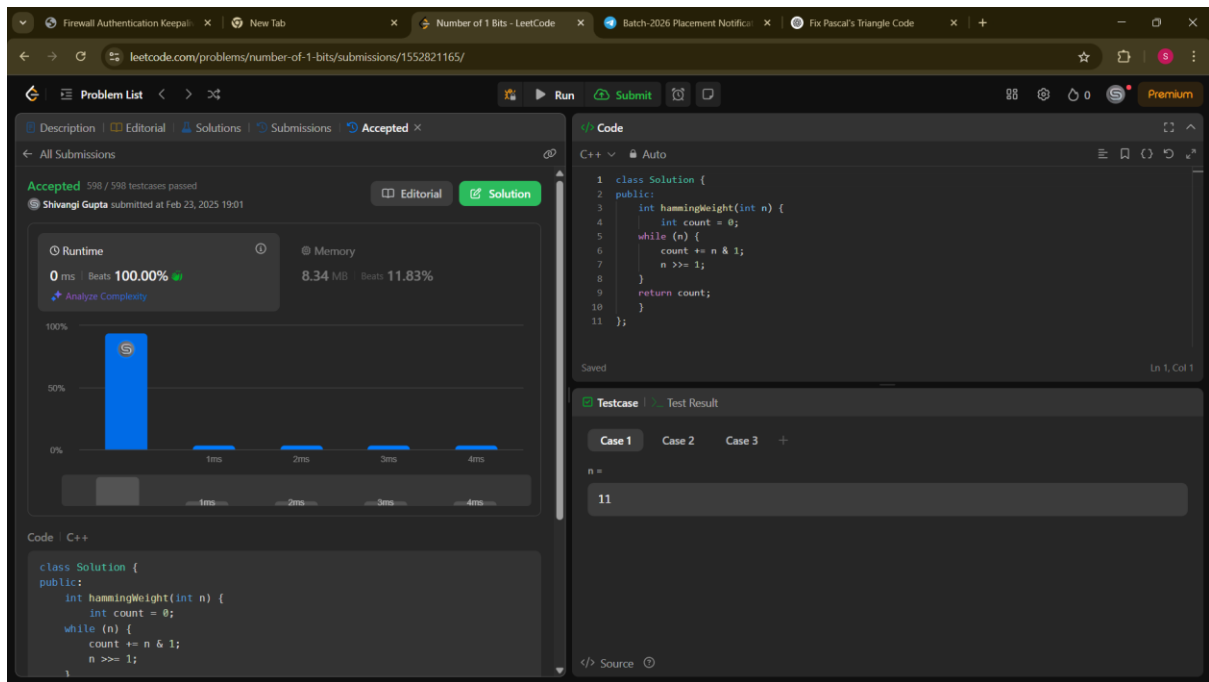
```
class Solution {
public:
    int hammingDistance(int x, int y) {
        int xorResult = x ^ y;
        int count = 0;
        while (xorResult) {
            count += xorResult & 1;
            xorResult >>= 1;
        }
        return count;
    }
};
```



Q3. Number of 1 Bits <https://leetcode.com/problems/number-of-1-bits/description/>

CODE:

```
class Solution {
public:
    int hammingWeight(int n) {
        int count = 0;
        while (n) {
            count += n & 1;
            n >>= 1;
        }
        return count;
    }
};
```



Q4. Divide two integers <https://leetcode.com/problems/divide-two-integers/description/>

CODE:

```
class Solution {
public:
```

```
    int divide(int dividend, int divisor) {
```

```
        // Handle overflow case for (INT_MIN / -1) which cannot be represented
```

```
        if (dividend <= INT_MIN && divisor == -1) {
```

```
            return INT_MAX; // This is the only case where overflow happens
```

```
        }
```

```
        // Convert both dividend and divisor to long long to avoid overflow
```

```
        long long absDividend = abs((long long)dividend);
```

```
        long long absDivisor = abs((long long)divisor);
```

```
        long long result = 0;
```

```

// Use bit manipulation to speed up the division
while (absDividend >= absDivisor) {
    long long tempDivisor = absDivisor;
    int multiple = 1;

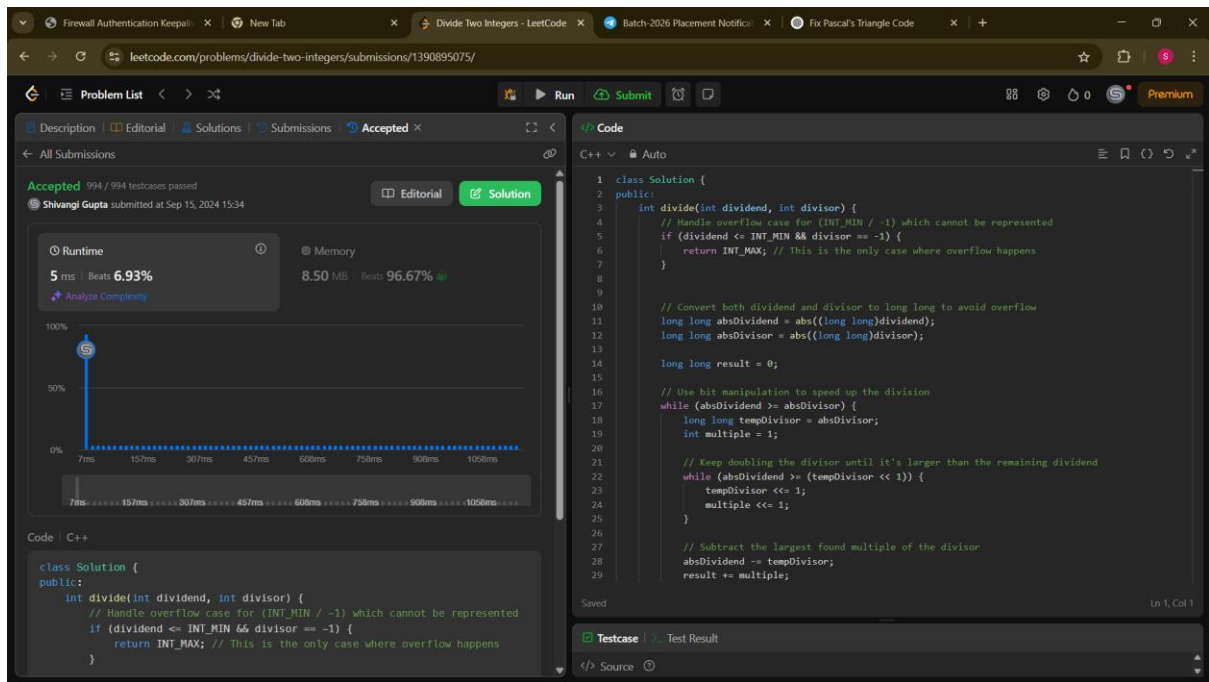
    // Keep doubling the divisor until it's larger than the remaining dividend
    while (absDividend >= (tempDivisor << 1)) {
        tempDivisor <<= 1;
        multiple <<= 1;
    }

    // Subtract the largest found multiple of the divisor
    absDividend -= tempDivisor;
    result += multiple;
}

// Determine the sign of the result
if ((dividend < 0 && divisor > 0) || (dividend > 0 && divisor < 0)) {
    result = -result;
}

return result;
}
};

```



Q5. Valid parentheses <https://leetcode.com/problems/valid-parentheses/description/>

Code:

```

class Solution {
public:
    bool isValid(string s) {
        stack<char> st;

        for (char c : s) {
            if (c == '(' || c == '{' || c == '[') {
                st.push(c);
            } else {
                if (st.empty()) return false;

                char top = st.top();

                if ((c == ')' && top != '(') ||
                    (c == '}' && top != '{') ||
                    (c == ']' && top != '[')) {
                    return false;
                }

                st.pop();
            }
        }

        return st.empty();
    }
};

```

```

    }

    }

    return st.empty();

}

};

```

The screenshot shows the LeetCode interface for the 'Valid Parentheses' problem. The left sidebar indicates the solution is 'Accepted' with a runtime of 0 ms. The input is 's = \"()\"' and the output is 'true'. The right pane displays the following C++ code:

```

1  class Solution {
2  public:
3      bool isValid(string s) {
4          stack<char> st;
5          for (char c : s) {
6              if (c == '(' || c == '[' || c == '{') {
7                  st.push(c);
8              } else {
9                  if (st.empty()) return false;
10                 char top = st.top();
11                 if ((c == ')' && top != '(') ||
12                     (c == ']' && top != '[') ||
13                     (c == '}' && top != '{')) {
14                     return false;
15                 }
16                 st.pop();
17             }
18         }
19         return st.empty();
20     }
21 };

```