DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
Discover. Learn. Empower.

EXPERIMENT - 10

Student Name: Sameer **UID:** 22BCS15631
Branch: Computer Science & Engineering **Section/Group:** IOT-614/B
Semester: 6th **Date of Performance:** 17/04/2025
Subject Name: Advanced Programming Lab-2 **Subject Code:** 22CSP-351

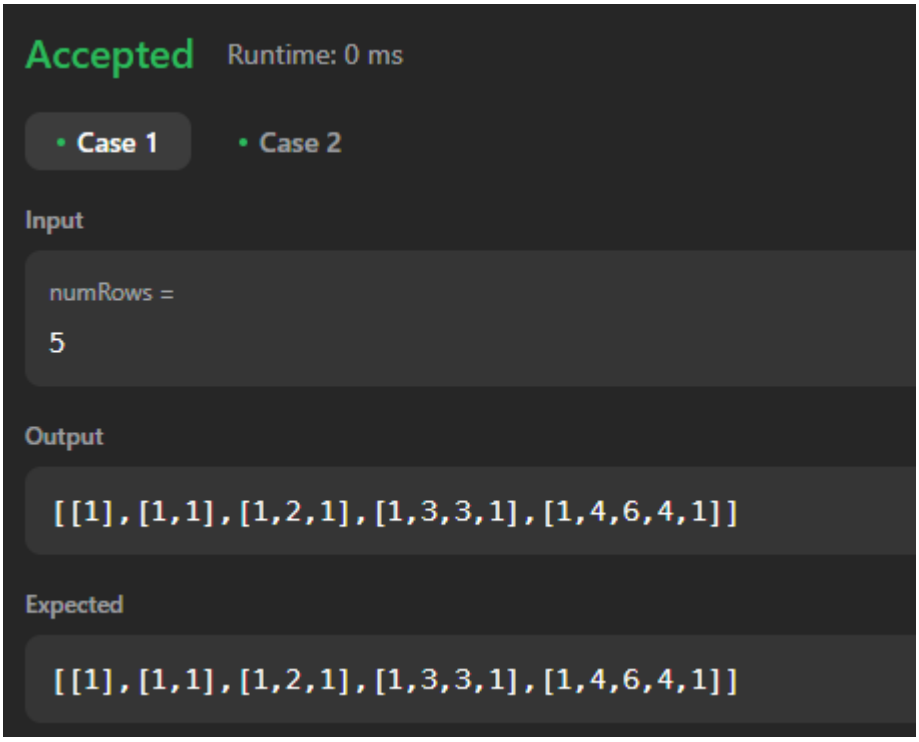
Q.1. Pascal's Triangle

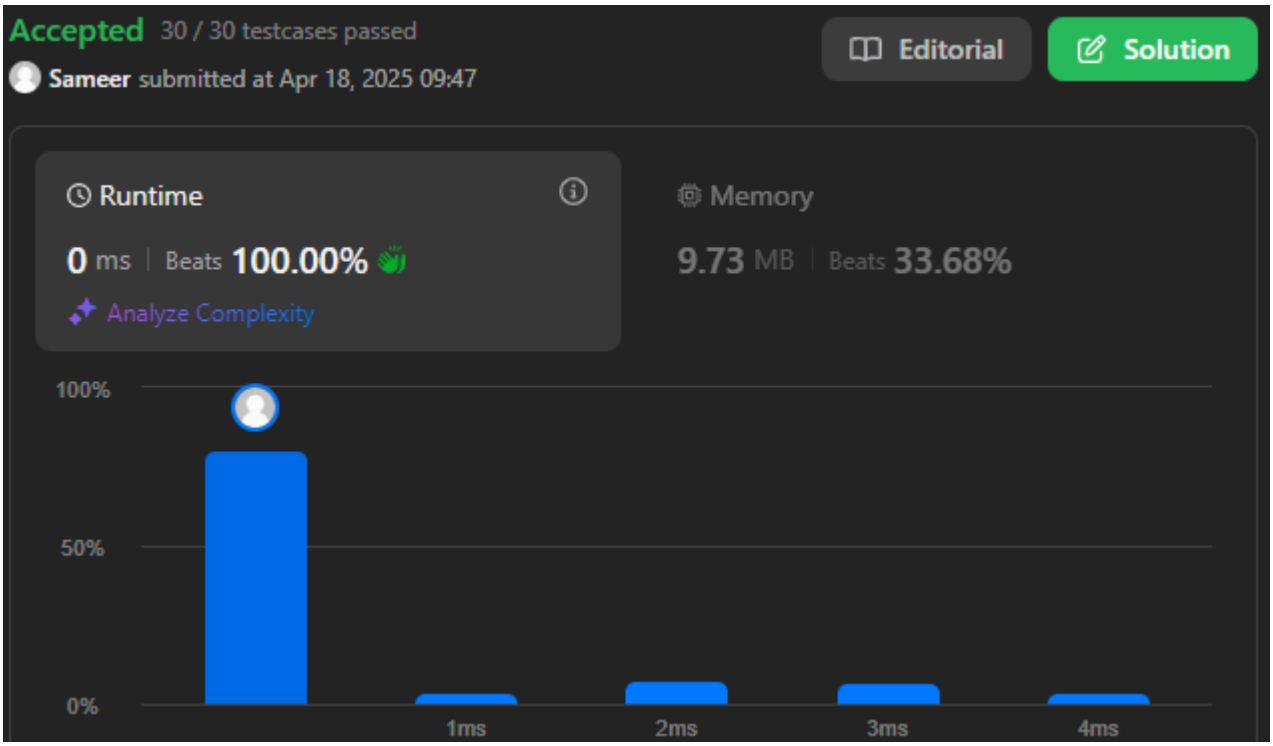
Given an integer numRows, return the first numRows of Pascal's triangle. In Pascal's triangle, each number is the sum of the two numbers directly above it.

Code:

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

      for (int i = 1 ;  i <= numRows ;  i++)
      {
         vector<int> row ;
         int coeff = 1 ;

         for (int j = 1 ;  j <= i ;  j++)
         {
            row.push_back(coeff) ;
            coeff = coeff * (i-j) / j ;
         }

         v.push_back(row) ;
      }
      return v ;
   }
};
```

Output:


Accepted
Runtime: 0 ms
Case 1
Case 2
Input
numRows =
5
Output
[[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]
Expected
[[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]


Accepted
30 / 30 testcases passed
Sameer submitted at Apr 18, 2025 09:47
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
9.73 MB | Beats 33.68%
100%
50%
0%
1ms
2ms
3ms
4ms

Q.2. Task Scheduler

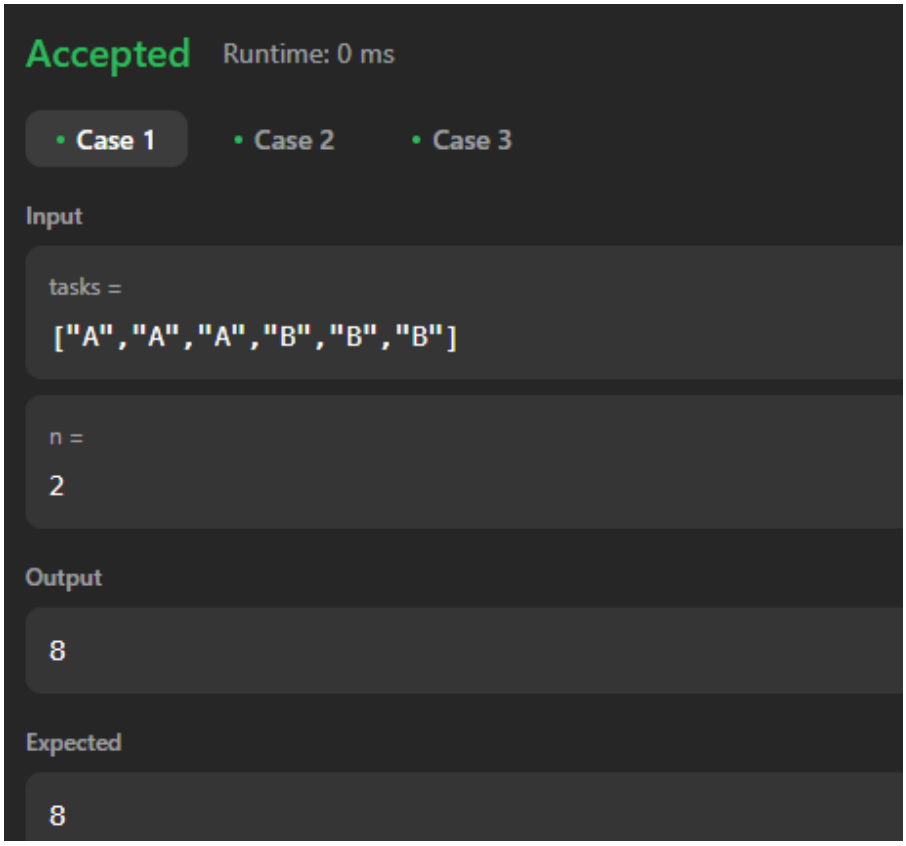
You are given an array of CPU tasks, each labeled with a letter from A to Z, and a number n. Each CPU interval can be idle or allow the completion of one task. Tasks can be completed in any order, but there's a constraint: there has to be a gap of at least n intervals between two tasks with the same label.

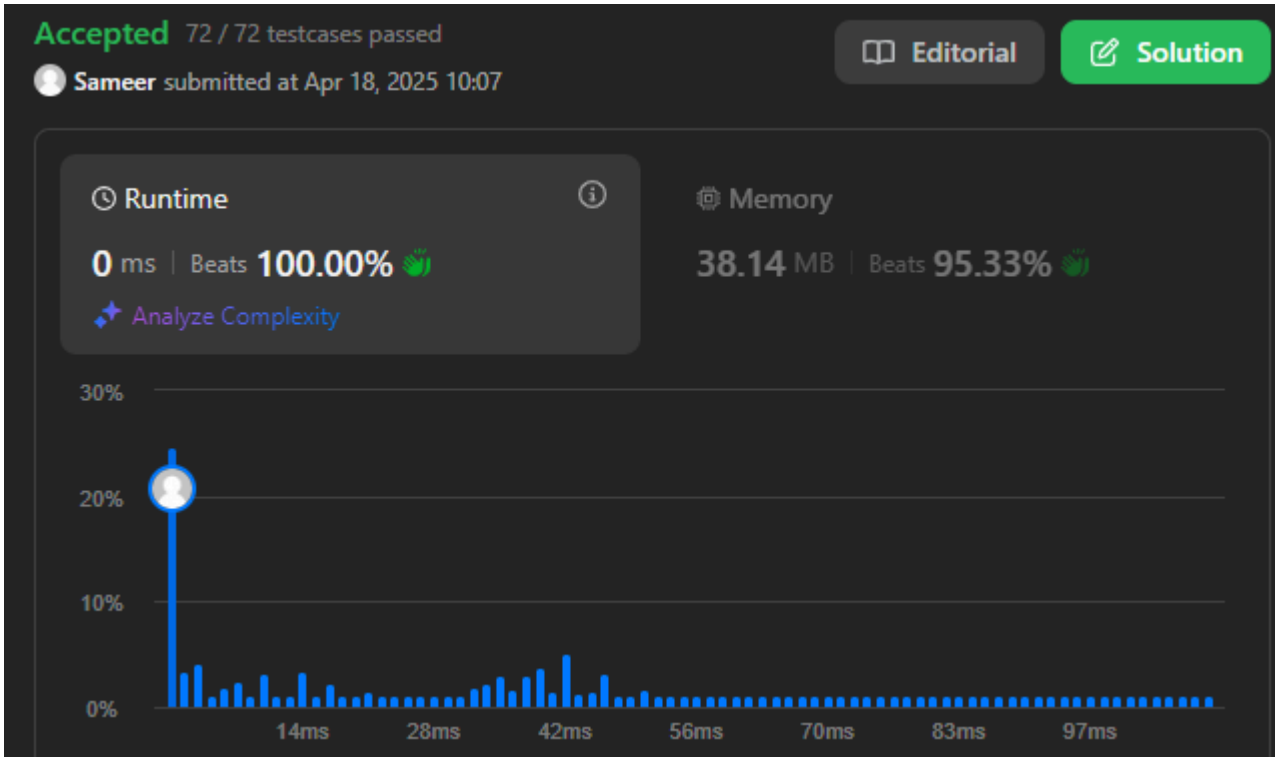
Return the minimum number of CPU intervals required to complete all tasks.

Code:

```
class Solution {
public:
    int leastInterval(vector<char>& tasks, int n) {
        vector<int> freq(26, 0);

        for (char task : tasks) {
            freq[task - 'A']++;
        }

        sort(freq.begin(), freq.end());

        int maxFreq = freq[25];
        int maxFreqCount = 0;

        for (int i = 25; i >= 0 && freq[i] == maxFreq; --i) {
            maxFreqCount++;
        }

        int partCount = maxFreq - 1;
        int partLength = n - (maxFreqCount - 1);
        int emptySlots = partCount * partLength;
        int availableTasks = tasks.size() - maxFreq * maxFreqCount;
        int idles = max(0, emptySlots - availableTasks);

        return tasks.size() + idles;
    }
};
```

Output:


Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
tasks =
["A","A","A","B","B","B"]
n =
2
Output
8
Expected
8


Accepted
72 / 72 testcases passed
Sameer submitted at Apr 18, 2025 10:07
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
38.14 MB | Beats 95.33%
30%
20%
10%
0%
14ms
28ms
42ms
56ms
70ms
83ms
97ms

Q.3. Valid Parentheses

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

- Open brackets must be closed by the same type of brackets.
- Open brackets must be closed in the correct order.
- Every close bracket has a corresponding open bracket of the same type.

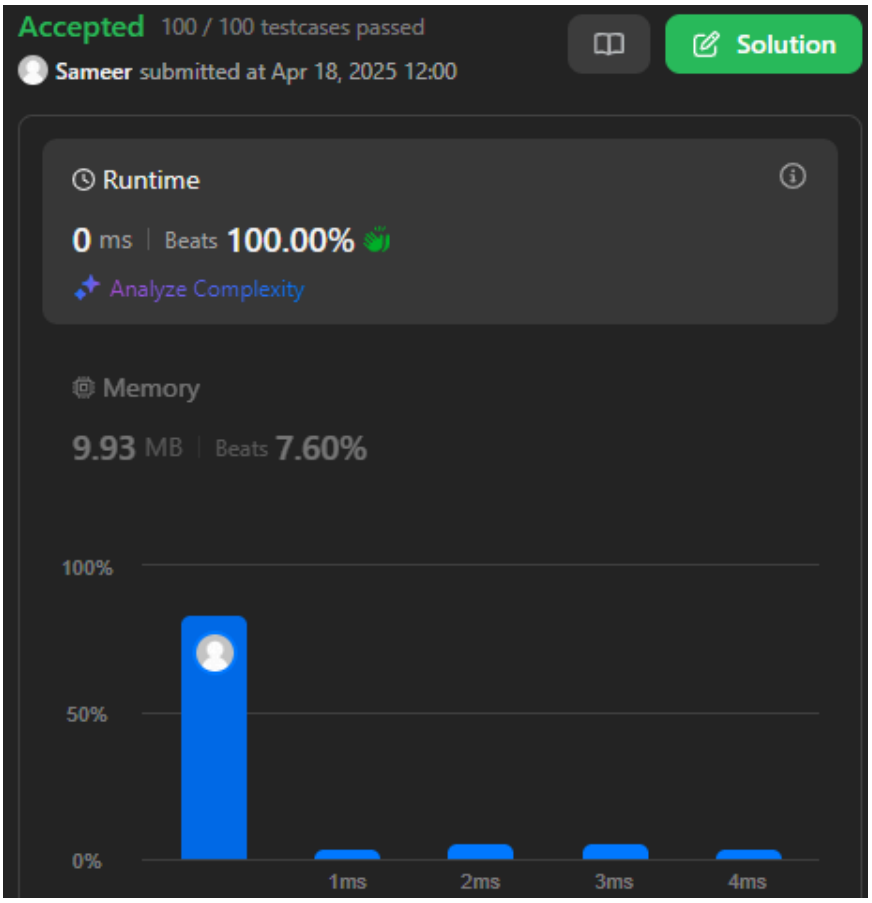
Code:

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

        for (int i = 0; i < s.size(); i++) {
            if (s[i] == '(' || s[i] == '{' || s[i] == '[') {
                st.push(s[i]);
            }

            else {
                if (st.empty()) {
                    return false;
                }

                else {
                    if (s[i] == ')' && st.top() == '(' ||
                        s[i] == ']' && st.top() == '[' ||
                        s[i] == '}' && st.top() == '{') {
                        st.pop();
                    }
```

```
                else
                {
                    return false;

                }
            }
        }
    }

    if (st.empty())
    {
        return true;
    }

    else
    {
        return false;
    }
  }
};
```

Output:


Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Case 4
Input
s =
"()"
Output
true
Expected
true


Accepted
100 / 100 testcases passed
Sameer submitted at Apr 18, 2025 12:00
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
9.93 MB | Beats 7.60%
100%
50%
0%
1ms
2ms
3ms
4ms

Q.4. Hamming Distance

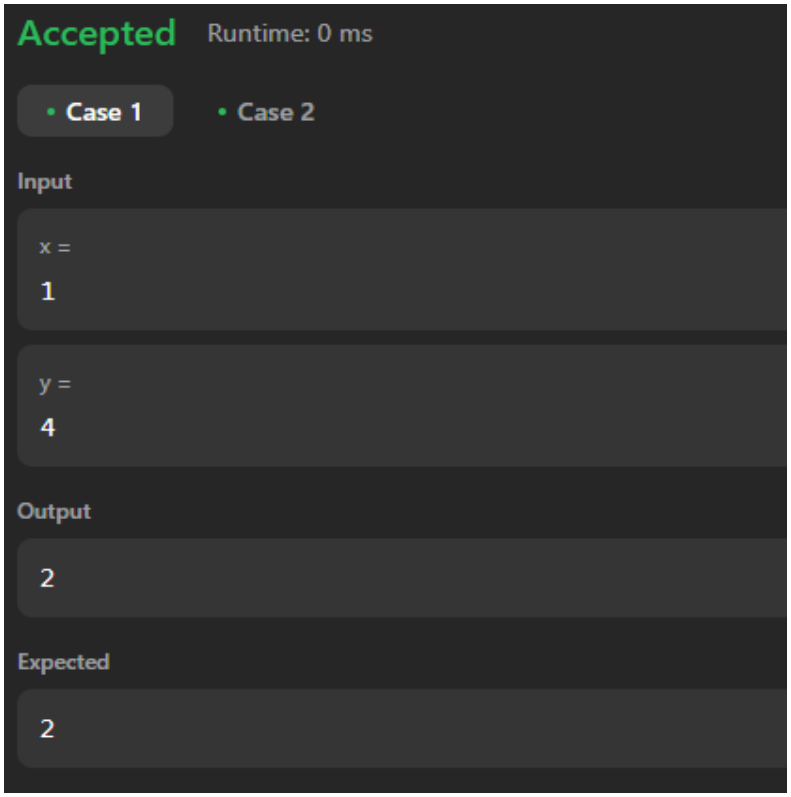
The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

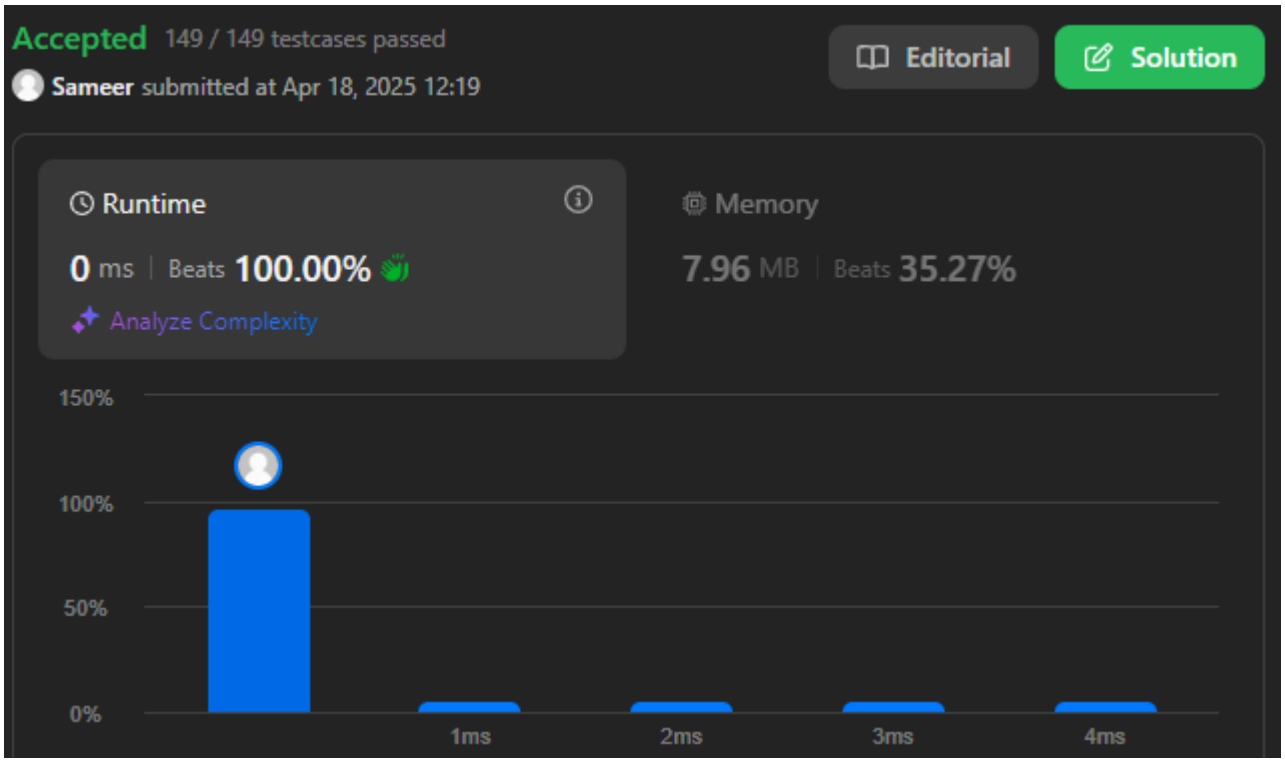
Given two integers x and y, return the Hamming distance between them.

Code:

```
int hammingDistance(int x, int y)
{
    int xorResult = x ^ y;
    int distance = 0;

    while (xorResult > 0)
    {
        if ((xorResult & 1) == 1)
        {
            distance++;
        }

        xorResult = xorResult >> 1;
    }

    return distance;
}
```

Output:


Accepted
Runtime: 0 ms
Case 1
Case 2
Input
x =
1
y =
4
Output
2
Expected
2


Accepted
149 / 149 testcases passed
Sameer submitted at Apr 18, 2025 12:19
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
7.96 MB | Beats 35.27%
150%
100%
50%
0%
1ms
2ms
3ms
4ms

Q.5. Trapping Rain Water

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Code:

```
class Solution {
public:
    int trap(vector<int>& height) {
        int n = height.size();
        int left = 0, right = n - 1;
        int leftMax = 0, rightMax = 0;
        int water = 0;

        while (left < right) {
            if (height[left] < height[right]) {
                if (height[left] >= leftMax) {
                    leftMax = height[left];
                }

                else {
                    water += leftMax - height[left];
                }

                left++;
            }

            else {
                if (height[right] >= rightMax) {
                    rightMax = height[right];
                }

                else {
                    water += rightMax - height[right];
                }

                right--;
            }
        }
```

```
        return water;
    }
};
```

Output: