

# Assignment 10 (Advance Programming)

Name – Ashish Kumar Singh

UID – 22BCS16892

## 118. 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 as shown:

Example 1:

Input: numRows = 5

Output: [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]

Example 2:

Input: numRows = 1

Output: [[1]]

### **Solution:**

```
class Solution {
```

```
public:
```

```
    vector<vector<int>> generate(int numRows) {
```

```
        vector<vector<int>> triangle;
```

```
        for (int i = 0; i < numRows; ++i) {
```

```
            vector<int> row(i + 1, 1);
```

```

        for (int j = 1; j < i; ++j) {
            row[j] = triangle[i - 1][j - 1] + triangle[i - 1][j];
        }

        triangle.push_back(row);
    }

    return triangle;
}
};

```

**118. Pascal's Triangle** Solved

Easy Topics Companies

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 as shown:

```

      1
     1 1
    1 2 1
   1 3 3 1
  1 6 6 3 1
 1 10 10 6 3 1

```

**Example 1:**

Input: `numRows = 5`

Output: `[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 6, 6, 3, 1]]`

**Code**

```

1 class Solution {
2 public:
3     vector<vector<int>> generate(int numRows) {
4         vector<vector<int>> triangle;
5
6         for (int i = 0; i < numRows; ++i) {
7             vector<int> row(i + 1, 1);
8
9             for (int j = 1; j < i; ++j) {
10                row[j] = triangle[i - 1][j - 1] + triangle[i - 1][j];
11            }
12
13            triangle.push_back(row);
14        }
15
16        return triangle;
17    }
18 }

```

Ln 8, Col 13 Saved Upgrade to Cloud Saving Run Submit

**Testcase** | **Test Result**

**Accepted** Runtime: 2 ms

Case 1 Case 2

Input

## 461. 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.

Example 1:

Input: `x = 1, y = 4`

Output: 2

Explanation:

1 (0 0 0 1)

4 (0 1 0 0)

↑ ↑

The above arrows point to positions where the corresponding bits are different.

### Solution:

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

The screenshot displays a coding platform interface for the problem "461. Hamming Distance". The left sidebar contains the problem description, which states that the Hamming distance is the number of positions where corresponding bits of two integers differ. It includes an example with input x=1 and y=4, resulting in an output of 2. The right pane shows the C++ code for the solution, which uses XOR to find differing bits and a loop to count them. Below the code, the "Test Result" section shows "Accepted" with a runtime of 0 ms. The bottom of the interface shows engagement metrics like 3.9K votes and 22 online users.

**461. Hamming Distance** Solved ✓

Easy Topics Companies

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.

**Example 1:**

Input:  $x = 1, y = 4$   
Output: 2  
Explanation:  
1 (0 0 0 1)  
4 (0 1 0 0)  
↑ ↑  
The above arrows point to positions where the corresponding bits are different.

**Example 2:**

3.9K 34 22 Online

**Code**

```
1 class Solution {  
2 public:  
3     int hammingDistance(int x, int y) {  
4         int xorResult = x ^ y;  
5         int distance = 0;  
6  
7         while (xorResult) {  
8             distance += xorResult & 1;  
9             xorResult >>= 1;  
10        }  
11  
12        return distance;  
13    }  
14 }  
15
```

Ln 8, Col 40 Saved Run Submit

**Testcase** | **Test Result**

**Accepted** Runtime: 0 ms

Case 1 Case 2

Input

## 621. 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.

Example 1:

Input: tasks = ["A","A","A","B","B","B"], n = 2

Output: 8

Explanation: A possible sequence is: A -> B -> idle -> A -> B -> idle -> A -> B.

After completing task A, you must wait two intervals before doing A again. The same applies to task B. In the 3rd interval, neither A nor B can be done, so you idle. By the 4th interval, you can do A again as 2 intervals have passed.

### Solution:

```
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 countMax = 0;

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

int partCount = maxFreq - 1;
int partLength = n + 1;
int emptySlots = partCount * partLength + countMax;

return max((int)tasks.size(), emptySlots);
}
};

```

## 621. Task Scheduler

Solved

Medium Topics Companies Hint

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.

**Example 1:**

Input: `tasks = ["A","A","A","B","B","B"], n = 2`

Output: 8

Explanation: A possible sequence is: A -> B -> idle -> A -> B -> idle -> A -> B.

11K 163 185 Online

C++ Auto

```

1 class Solution {
2 public:
3     int leastInterval(vector<char>& tasks, int n) {
4         vector<int> freq(26, 0);
5
6         for (char task : tasks) {
7             freq[task - 'A']++;
8         }
9
10        sort(freq.begin(), freq.end());
11
12        int maxFreq = freq[25];
13        int countMax = 0;
14
15        for (int i = 25; i >= 0 && freq[i] == maxFreq; --i) {
16            ...
17        }
18    }
19 };

```

Ln 5, Col 9 | Saved Run Submit

Testcase Test Result

Accepted Runtime: 0 ms

Case 1

Case 2

Case 3

Input

## 191. Number of 1 Bits

Given a positive integer  $n$ , write a function that returns the number of set bits in its binary representation (also known as the Hamming weight).

Example 1:

Input:  $n = 11$

Output: 3

Explanation:

The input binary string 1011 has a total of three set bits.

Example 2:

Input:  $n = 128$

Output: 1

Explanation:

The input binary string 10000000 has a total of one set bit.

### **Solution:**

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

The screenshot shows the LeetCode interface for the problem "191. Number of 1 Bits". The left sidebar contains the problem description, which asks for the number of set bits in the binary representation of a positive integer  $n$ . It includes two examples: Example 1 with input  $n = 11$  and output 3, and Example 2 with input  $n = 128$  and output 1. The right pane shows a C++ code editor with a solution for the Hamming weight problem. The code defines a class `Solution` with a public method `hammingWeight` that uses a while loop to count the number of 1s in  $n$  by repeatedly applying  $n = n \& (n - 1)$ . Below the code editor, the "Test Result" section shows "Accepted" with a runtime of 0 ms for Case 1.

**191. Number of 1 Bits** Solved ✓

Easy Topics Companies

Given a positive integer  $n$ , write a function that returns the number of set bits in its binary representation (also known as the Hamming weight).

**Example 1:**

Input:  $n = 11$

Output: 3

**Explanation:**

The input binary string 1011 has a total of three set bits.

**Example 2:**

Input:  $n = 128$

Output: 1

```
1 class Solution {
2 public:
3     int hammingWeight(int n) {
4         int count = 0;
5         while (n != 0) {
6             n = n & (n - 1);
7             count++;
8         }
9         return count;
10    }
11 }
12
```

Ln 10, Col 5 Saved Run Submit

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

## 42. 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.

Example 1:

Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]

Output: 6

Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

### Solution:

```
class Solution {
public:
    int trap(vector<int>& height) {
        int n = height.size();
        if (n == 0) return 0;
```

```
vector<int> left_max(n, 0);
vector<int> right_max(n, 0);

left_max[0] = height[0];
for (int i = 1; i < n; ++i) {
    left_max[i] = max(left_max[i-1], height[i]);
}

right_max[n-1] = height[n-1];
for (int i = n-2; i >= 0; --i) {
    right_max[i] = max(right_max[i+1], height[i]);
}

int total_water = 0;
for (int i = 0; i < n; ++i) {

    total_water += min(left_max[i], right_max[i]) - height[i];
}

return total_water;
};
```




DescriptionAccepted ×EditorialSolutions

## 42. Trapping Rain Water

HardTopicsCompanies

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.

**Example 1:**



**Input:** height = [0,1,0,2,1,0,1,3,2,1,2,1]  
**Output:** 6  
**Explanation:** The above elevation map (black bars) has several valleys (blue areas) which can be trapped after raining.

33.8K377473 Online

Code

C++Auto

```
1 class Solution {
2 public:
3     int trap(vector<int>& height) {
4         int n = height.size();
5         if (n == 0) return 0;
6
7         vector<int> left_max(n, 0);
8         vector<int> right_max(n, 0);
9
10        left_max[0] = height[0];
11        for (int i = 1; i < n; ++i) {
12            left_max[i] = max(left_max[i-1], height[i]);
13        }
14
15        right_max[n-1] = height[n-1];
16        for (int i = n-2; i >= 0; --i) {
17            right_max[i] = max(right_max[i+1], height[i]);
18        }
19
20        int water = 0;
21        for (int i = 1; i < n-1; ++i) {
22            water += min(left_max[i], right_max[i]) - height[i];
23        }
24        return water;
25    }
26 }
```

Ln 29, Col 1 Saved

RunSubmit

TestcaseTest Result

AcceptedRuntime: 0 ms

Case 1Case 2

Input