Experiment-9

Student Name: Anupreet Kaur UID: 22BCS50071

Branch: BE-CSE Section/Group: NTPP_IOT-602-A

Semester: 6th Date of Performance: 17/04/2025

Subject Name: AP Lab Subject Code: 22CSP-351

1. Aim: Graphs.

❖ Problem 1.2.1: Hamming Distance

❖ Problem 1.2.2: Task Scheduler

❖ Problem 1.2.2: Divide two Integers

2. Objective:

To calculate the Hamming Distance between two integers by counting the number of differing bits.

To calculate the minimum CPU intervals needed to complete all tasks with at least n intervals between the same type.

To compute the quotient of dividing two integers dividend and divisor without using multiplication, division, or mod operators. The result should truncate toward zero.

3. Code:

Hamming Distance:

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

Task Scheduler:

```
import java.util.*;
class Solution {
```

```
public int leastInterval(char[] tasks, int n) {
     int[] freq = new int[26];
    for (char task : tasks) {
       freq[task - 'A']++;
     }
    Arrays.sort(freq);
    int maxFreq = freq[25];
    int maxCount = 1;
     for (int i = 24; i >= 0; i--) {
       if (freq[i] != maxFreq) break;
       maxCount++;
     }
    int partCount = maxFreq - 1;
    int partLength = n - (maxCount - 1);
    int emptySlots = partCount * partLength;
    int availableTasks = tasks.length - maxFreq * maxCount;
    int idles = Math.max(0, emptySlots - availableTasks);
    return tasks.length + idles;
  }
Divide Two Integers:
class Solution {
  public int divide(int dividend, int divisor) {
    // Handle overflow
    if (dividend == Integer.MIN_VALUE && divisor == -1) {
       return Integer.MAX_VALUE;
     }
    // Get the sign of the result
    boolean negative = (dividend < 0) ^ (divisor < 0);
     // Convert to positive for simplicity
     long dividendAbs = Math.abs((long) dividend);
    long divisorAbs = Math.abs((long) divisor);
    int quotient = 0;
    // Perform division using bit shifts
     while (dividendAbs >= divisorAbs) {
```

}

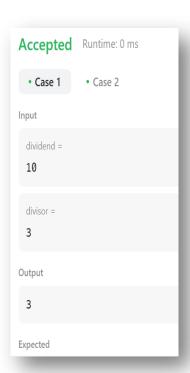
```
long tempDivisor = divisorAbs, multiple = 1;
    // Increase divisor by powers of 2 (left shift) to speed up the process
    while (dividendAbs >= (tempDivisor << 1)) {
        tempDivisor <<= 1;
        multiple <<= 1;
        }
        dividendAbs -= tempDivisor;
        quotient += multiple;
    }

    // Apply sign and return the result
    return negative ? -quotient : quotient;
}</pre>
```

6. Output:







7. Learning Outcomes:

- ➤ Understand bitwise XOR operation and Learn how to count set bits.
- ➤ Efficient use of HashMaps and Arrays and Understanding greedy scheduling strategies.
- > Understand how to prevent overflow in arithmetic operations and Truncating floating-point results in integer division.