

Experiment-9

Student Name: Amit Kumar UID:22BCS50073

Branch: BE-CSE Section: NTPP_IOT-602 (A)

Semester: 6th Date of Performance: 17/4/2025

Subject: AP LAB 2 Subject Code: 22CSP-351

1. Aim: solve the various problems.

a. Easy: Hamming Distance, Pascal's Triangle

b. **Medium:** Valid Parenthesis String, Divide Two Integers

c. Hard: Max number of tasks you can assign,

2. Algorithm:

a. EASY1: Pascal's Triangle

- Initialize a 2D vector triangle.
- For each row i from 0 to numRows 1:
- Create a row of size i + 1.
- Set first and last elements to 1.
- Fill middle elements using values from previous row.
- Return the triangle.

b. Easy2: Hamming Distance

- Initialize a counter cnt to 0. This will store the number of differing bits.
- While both x and y are not zero:

Compare the least significant bits of x and y using $(x \& 1) \land (y \& 1)$.

If the bits differ, increment cnt.

Right shift both x and y by 1 bit.

• If x still has remaining bits:

For each set bit in x, increment cnt.

Right shift x until it's 0.

• If y still has remaining bits:

For each set bit in y, increment cnt.

Right shift y until it's 0.

• Return cnt.

c. **Medium1:**Divide Two Integers

- Handle overflow case: if dividend = INT_MIN and divisor = -1, return INT_MAX.
- Record the sign of result based on dividend and divisor.
- Convert both numbers to long long and take their absolute values.
- Repeatedly subtract (or use bit shifts to speed up) the divisor from dividend and count how many
- Apply the sign to the result and return.

d. Medium2: Valid Parenthesis String

• Initialize two counters:

leftMin \rightarrow the **minimum** number of unmatched '(' that might still be open. leftMax \rightarrow the **maximum** number of unmatched '(' that might still be open.

• Loop through each character c in the string:

If c is '(':

Increase both leftMin and leftMax by 1.

If c is ')':

Decrease both leftMin and leftMax by 1.

If c is '*':

'*' can be '(', ')', or empty:

Decrease leftMin by 1 (assume it's ')')

Increase leftMax by 1 (assume it's '(')

• If at any point leftMax becomes negative:

Return false \rightarrow too many closing brackets.

• If leftMin drops below zero:

Reset leftMin to $0 \rightarrow$ we can ignore extra closing by treating '*' as empty.

• After processing the entire string:

If leftMin is 0, return true (all parentheses balanced)

Else, return false.

e. Hard: Max Number of tasks you can assign

- Let n be the size of target.
- Create a map map to store the index of each element in target □ Sort the tasks in ascending order.
- Sort the workers in ascending order.
- Set the search range for binary search: low = 0, high = min(number of workers, number of tasks).
- Initialize ans to store the final answer.

• Binary Search Loop:

While low <= high:

Set mid = $(low + high) / 2 \rightarrow trying to assign mid number of tasks.$

Copy all workers into a multiset st to efficiently remove used workers.

Set a counter count = 0 to track pills used.

Set a flag flag = true to track if assignment is successful.

For each task from hardest to easiest among mid tasks:

Get the strongest available worker.

If the worker can do the task without a pill, assign and remove the worker.

Else, find the **weakest** worker that can do the task **with pill boost**.

If such worker exists, assign task, remove worker, increment count.

If not, set flag = false, break.

If count > p, break and set flag = false.

If assignment was successful (flag == true), store ans = mid and try a higher value (low = mid + 1).

Else, try a smaller value (high = mid - 1).

• Return ans \rightarrow maximum number of tasks that can be assigned.

3. <u>Code:</u>

```
a. Hamming Distance: class Solution
```

```
{ public:
  int hamming Distance(int x, int
    y) \{ int cnt = 0;
    while(x && y)
      { if(x&1 ^ y&1)
         cnt++;
                 x >>= 1; y >>= 1;
    while(x) {
      if(x\&1) \{ cnt++; \}
      x >>= 1;
    while(y) {
      if(y&1) { cnt++; }
      y >>= 1;
    return cnt;
  }
};
```

b. Pascal's Triangle:

```
class Solution {
public:
vector<vector<int>>> generate(int numRows) {
vector<vector<int>>> res(numRows);
for (int i = 0; i < numRows; i++) {
    res[i].resize(i + 1, 1);
    long long ans = 1; // Store intermediate values to avoid overflow
    for (int j = 1; j < i; j++) {
        ans = ans * (i - j + 1) / j; // Using binomial coefficient formula
        res[i][j] = ans;
    }
    return res;
}
</pre>
```

c. Divide Two Integers:

```
int divide(int dividend, int divisor) {
  if (dividend == INT_MIN && divisor == -1)
    return INT_MAX;

long long a = abs((long long)dividend);
long long b = abs((long long)divisor);
int result = 0;

while (a >= b) {
  long long temp = b, multiple = 1;
  while (a >= (temp << 1)) {
    temp <<= 1;
    multiple <<= 1;
  }
  a -= temp;
  result += multiple;
}

return ((dividend > 0) ^ (divisor > 0)) ? -result : result;
```

d. Valid Parenthesis String:

```
class Solution
{ public:
      bool checkValidString(string
      s) { int leftMin = 0, leftMax =
      0; for (char c : s) {
       if (c == '(') \{
          leftMin++;
          leftMax++;
        } else if (c == ')')
          { leftMin--;
          leftMax--;
        } else {
          leftMin--;
          leftMax++;
       if (leftMax < 0) return false;
       if (leftMin < 0) leftMin = 0;
     return leftMin == 0;
  }
};
```

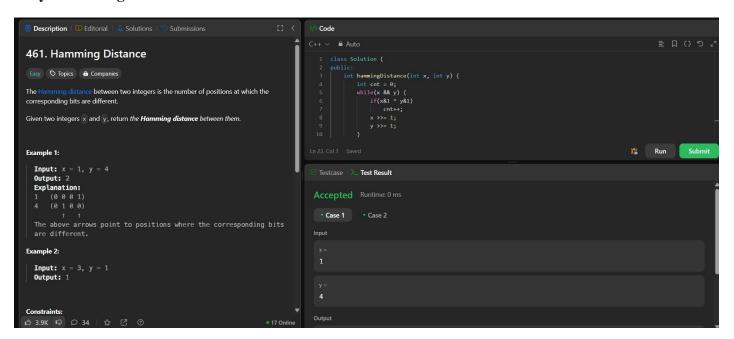
e. Max number of task you can assign

```
class Solution
{ public:
  int maxTaskAssign(vector<int>& tasks, vector<int>& workers, int p, int strength)
     { int n = tasks.size(), m = workers.size();
     sort(tasks.begin(), tasks.end());
     sort(workers.begin(), workers.end());
     int lo = 0, hi = min(m, n);
     int ans:
     while(lo <= hi) {
        int mid = lo + (hi - lo) / 2;
        int count = 0;
        bool flag = true;
        multiset<int> st(workers.begin(), workers.end());
        for(int i = mid - 1; i >= 0; i--) {
          auto it = prev(st.end());
          if(tasks[i] <= *it) {
             st.erase(it);
           } else {
             auto it = st.lower_bound(tasks[i] - strength);
             if(it != st.end()) {
                count++;
                st.erase(it);
             } else {
                flag = false;
                break;
             }
          if(count > p)
             \{ flag =
             false; break;
           }
        if(flag) {
          ans = mid;
          lo = mid + 1;
        } else {
          hi = mid - 1;
        }} return ans; }};
```

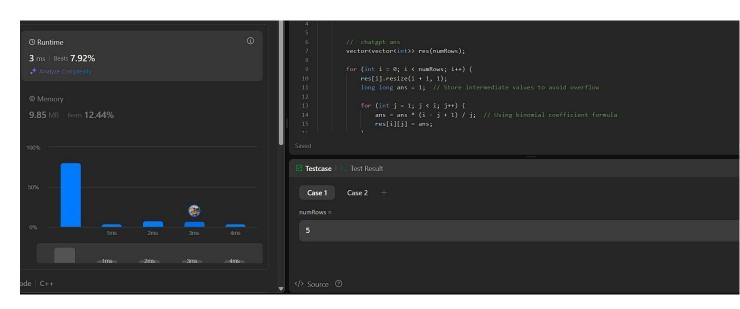


4. Output:

Easy: Hamming Distance

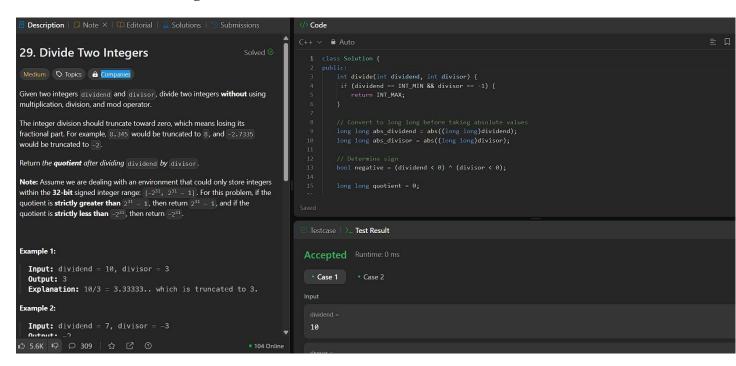


EASY: PASCAL TRIANGLE

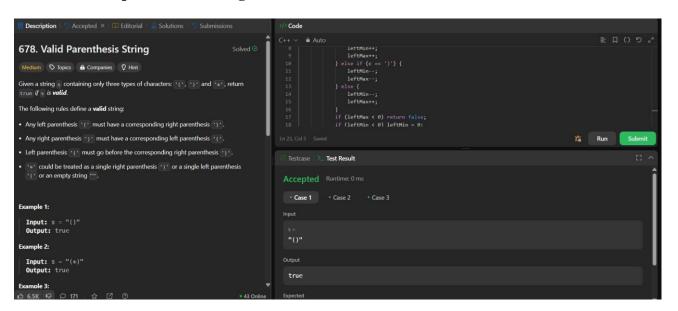




Medium: Divide Two Integers



Medium: Valid parenthesis String





Hard: Max number of tasks you can assign

