DYNAMIC PROGRAMMING

Q1)
https://leetcode.com/problems/fibonacci-number/description/

```
class Solution {
public:
   int solve(int n,vector<int>&v){
      if(n<=1)
      return n;
      if(v[n]!=-1)
      return v[n];

      return v[n]=solve(n-1,v)+solve(n-2,v);
   }

   int fib(int n) {
     if(n<=1){
      return n;
     }
     vector<int>v(n+1,-1);
     return solve(n,v);
   }
};
```

Q2)
https://leetcode.com/problems/climbing-stairs/

```
class Solution {
public:
   int solve(int n,vector<int>&v){
      if(n<0)
       return 0;
       if(n==0)
       return 1;
       if(v[n]!=-1){
         return v[n];
       }
       int first=solve(n-1,v);
       int second=solve(n-2,v);
       return v[n]=first+second;
   }
   int climbStairs(int n) {
      vector<int>v(n+1,-1);
      return solve(n,v);
   }
};
```

Q3) **https://leetcode.com/problems/house-robber-ii/submissions/1338349698/**

```
class Solution {
public:
 int solve(vector<int>& nums, int i, int n,vector<int>&t) {
      if(i >= n)
         return 0;

      if(t[i] != -1)
         return t[i];

      int take = nums[i] + solve(nums, i+2, n,t); //steals ith house and moves to i+2 (because we can't steal adjacent)
      int skip = solve(nums, i+1, n,t); //skips this house, now we can move to adjacent next house

      return t[i]=max(take, skip);
   }
   int rob(vector<int>& nums){
      int n = nums.size();
      if(n == 1)
         return nums[0];
      if(n == 2)
         return max(nums[0], nums[1]);

         vector<int>v1(n,-1);
         vector<int>v2(n,-1);

         int take_0th_index_house = solve(nums, 0, n-1,v1);

         int take_1st_index_house = solve(nums, 1, n,v2);

         return max(take_0th_index_house, take_1st_index_house);

   }
};
```

Q4) **https://leetcode.com/problems/house-robber/**

```
class Solution {
public:
   int solve(vector<int>& nums, int i, int n,vector<int>&t) {
      if(i >= n)
         return 0;

      if(t[i] != -1)
         return t[i];
```

```
        int take = nums[i] + solve(nums, i+2, n,t);
//steals ith house and moves to i+2 (because we
can't steal adjacent)
        int skip = solve(nums, i+1, n,t); //skips this
house, now we can move to adjacent next house

        return t[i]=max(take, skip);
    }

    int rob(vector<int>& nums) {
        int n = nums.size();
        vector<int>v(n,-1);
        return solve(nums, 0, n,v);
    }
};
```

Q5) https://leetcode.com/problems/maximum-alternating-subsequence-sum/description/

```
class Solution {
public:
    long long solve(int idx, vector<int>& nums, bool
iseven, vector<vector<long long>>& t, int n) {
        if (idx >= n) {
            return 0;
        }
        if (t[idx][iseven] != -1)
            return t[idx][iseven];

        long long skip = solve(idx + 1, nums, iseven, t,
n);

        long long val = nums[idx];
        if (!iseven) {
            val = -val;
        }

        long long take = solve(idx + 1, nums, !iseven, t,
n) + val;

        return t[idx][iseven] = max(skip, take);
    }

    long long maxAlternatingSum(vector<int>&
nums) {
        int n = nums.size();
        vector<vector<long long>> t(n, vector<long
long>(2, -1));
        return solve(0, nums, true, t, n);
    }
};
```

//bottom up approach(dry run and understand

```
class Solution {
public:
    long long solve(int idx, vector<int>& nums, bool
iseven, vector<vector<long long>>& t, int n) {
        if (idx >= n) {
            return 0;
        }
        if (t[idx][iseven] != -1)
            return t[idx][iseven];

        long long skip = solve(idx + 1, nums, iseven, t,
n);

        long long val = nums[idx];
        if (!iseven) {
            val = -val;
        }

        long long take = solve(idx + 1, nums, !iseven, t,
n) + val;

        return t[idx][iseven] = max(skip, take);
    }

    long long maxAlternatingSum(vector<int>&
nums) {
        int n = nums.size();
        vector<vector<long long>> t(n, vector<long
long>(2, -1));
        return solve(0, nums, true, t, n);
    }
};
```

Q6) https://leetcode.com/problems/longest-increasing-subsequence/description/

```
class Solution {
public:
    int n;
    int lis(vector<int>& nums, int prev_idx, int
curr_idx,vector<vector<int>>&t) {
        if(curr_idx == n)
            return 0;

        if(prev_idx != -1 && t[prev_idx][curr_idx] != -
1)
            return t[prev_idx][curr_idx];

        int taken = 0;
        if(prev_idx == -1 || nums[curr_idx] >
nums[prev_idx])
            taken = 1 + lis(nums, curr_idx, curr_idx+1,t);

        int not_taken = lis(nums, prev_idx,
curr_idx+1,t);
```

```
        if(prev_idx != -1)
            t[prev_idx][curr_idx] =  max(taken,
not_taken);

        return max(taken, not_taken);

    }
    int lengthOfLIS(vector<int>& nums) {
        n = nums.size();
        vector<vector<int>> t(n, vector<int>(n, -1));
        return lis(nums, -1, 0,t);
    }
};
```

Q7)

https://leetcode.com/problems/maximum-length-of-pair-chain/

```
class Solution {
public:
    int n;
    int lis(vector<vector<int>>& pairs, int prev_idx,
int curr_idx, vector<vector<int>>& t) {
        if (curr_idx == n)
            return 0;

        if (prev_idx != -1 && t[prev_idx][curr_idx] != -
1)
            return t[prev_idx][curr_idx];

        int taken = 0;
        if (prev_idx == -1 || pairs[curr_idx][0] >
pairs[prev_idx][1])
            taken = 1 + lis(pairs, curr_idx, curr_idx + 1,
t);

        int not_taken = lis(pairs, prev_idx, curr_idx +
1, t);

        if (prev_idx != -1)
            t[prev_idx][curr_idx] = max(taken,
not_taken);

        return max(taken, not_taken);
    }

    int findLongestChain(vector<vector<int>>&
pairs) {
        n = pairs.size();
        vector<vector<int>> t(n, vector<int>(n, -1));
        sort(begin(pairs), end(pairs)); // Sort pairs
based on the first element by default
        return lis(pairs, -1, 0, t);
    }
};
```

Q8)

https://leetcode.com/problems/longest-string-chain/description/

```
class Solution {
public:
    int n;
    bool predecessor(string& prev, string& curr) {
        int M = prev.length();
        int N = curr.length();

        if(M >= N || N-M != 1)
            return false;

        int i = 0, j = 0;
        while(i < M && j < N) {
            if(prev[i] == curr[j]) {
                i++;
            }
            j++;
        }
        return i==M;
    }

    int lis(vector<string>& words, int prev_idx, int
curr_idx,vector<vector<int>>&t) {
        if(curr_idx == n)
            return 0;

        if(prev_idx != -1 && t[prev_idx][curr_idx] != -
1)
            return t[prev_idx][curr_idx];

        int taken = 0;
        if(prev_idx == -1 ||
predecessor(words[prev_idx], words[curr_idx]))
            taken = 1 + lis(words, curr_idx,
curr_idx+1,t);

        int not_taken = lis(words, prev_idx,
curr_idx+1,t);

        if(prev_idx != -1)
            t[prev_idx][curr_idx] =  max(taken,
not_taken);

        return max(taken, not_taken);

    }

    static bool myFunction(string& s1, string& s2) {
        return s1.length() < s2.length();
    }
```

```
    int longestStrChain(vector<string>& words) {
        n = words.size();
        vector<vector<int>>t(n,vector<int>(n,-1));
        sort(begin(words), end(words), myFunction);
//You can select pairs in any order.
        return lis(words, -1, 0,t);
    }
};
```

Q9)

https://leetcode.com/problems/build-array-where-you-can-find-the-maximum-exactly-k-comparisons/description/

```
class Solution {
public:
    int N, M, K;
    int MOD = 1e9+7;

    int solve(int idx, int searchCost, int maxSoFar,
vector<vector<vector<int>>>& t) {
        if (idx == N) {
            return searchCost == K ? 1 : 0;
        }

        if (searchCost > K || maxSoFar > M) {
            return 0; // Out of bounds case, should
return 0 as it's invalid
        }

        if (t[idx][searchCost][maxSoFar] != -1) {
            return t[idx][searchCost][maxSoFar];
        }

        int result = 0;

        for (int i = 1; i <= M; i++) {
            if (i > maxSoFar) {
                result = (result + solve(idx + 1, searchCost
+ 1, i, t)) % MOD;
            } else {
                result = (result + solve(idx + 1, searchCost,
maxSoFar, t)) % MOD;
            }
        }

        return t[idx][searchCost][maxSoFar] = result %
MOD;
    }

    int numOfArrays(int n, int m, int k) {
        N = n;
        M = m;
        K = k;
        vector<vector<vector<int>>> t(N + 1,
vector<vector<int>>(K + 1, vector<int>(M + 1, -1)));
        return solve(0, 0, 0, t);
    }
};
```

Q10)

https://leetcode.com/problems/maximum-balanced-subsequence-sum/description/

note- LIS gives TLE

```
class Solution {
public:
    unordered_map<string, long long> mp; //For
memoization
    long long solve(int i, int prev, vector<int>&
nums) {
        if(i >= nums.size()) {
            return 0;
        }

        string key = to_string(i) + "_" + to_string(prev);
        if(mp.find(key) != mp.end()) {
            return mp[key];
        }

        long long taken = INT_MIN;

        if(prev == -1 || nums[i] - i >= nums[prev] -
prev) {
            taken = nums[i] + solve(i+1, i, nums);
        }

        long long not_taken = solve(i+1, prev, nums);
        return mp[key] = max<long long>(taken,
not_taken);
    }

    long long
maxBalancedSubsequenceSum(vector<int>&
nums) {
        int maxEl = *max_element(begin(nums),
end(nums));
        if(maxEl <= 0) {
            return maxEl;
        }
        return solve(0, -1, nums);
    }
};
```

Q11)

https://leetcode.com/problems/longest-increasing-subsequence/

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {
        int n = nums.size();
        vector<int> sorted;

        for(int i = 0; i<n; i++) {
            /*
                                    Why lower bound ?
                                    We want increasing subsequence and  hence
                                    we want to eliminate the duplicates as well.
                                    lower_bound returns iterator to "next greater or equal to"
                                */
            auto it = lower_bound(begin(sorted), end(sorted), nums[i]);

            if(it == end(sorted))
                sorted.push_back(nums[i]); //greatest : so insert it
            else
                *it = nums[i]; //replace
        }

        return (int)sorted.size();
    }
};
```

Q12)

https://leetcode.com/problems/largest-divisible-subset/

shows tle

```
class Solution {
public:

    void generate(int idx, vector<int>& nums, vector<int>& result, vector<int>& temp, int prev) {
        if(idx >= nums.size()) {
            if(temp.size() > result.size()) {
                result = temp;
            }
            return;
        }

        if(prev == -1 || nums[idx] % prev == 0) {
            temp.push_back(nums[idx]);
            generate(idx+1, nums, result, temp, nums[idx]);
            temp.pop_back();
        }

        generate(idx+1, nums, result, temp, prev);
    }

    vector<int> largestDivisibleSubset(vector<int>& nums) {
        sort(begin(nums), end(nums));

        vector<int> result;
```

```
        vector<int> temp;
        generate(0, nums, result, temp, -1);

        return result;
    }
};
```

Q13)

https://leetcode.com/problems/longest-common-subsequence/submissions/1340673819/

```
class Solution {
public:
    int m,n;
    int LCS(string& s1, string& s2, int i, int j,vector<vector<int>>&t) {
        if(i == m || j == n)
            return t[i][j] = 0;

        if(t[i][j] != -1)
            return t[i][j];//memo

        if(s1[i] == s2[j])
            return t[i][j] = 1 + LCS(s1, s2, i+1, j+1,t);

        return t[i][j] = max(LCS(s1, s2, i, j+1,t), LCS(s1, s2, i+1, j,t));
    }
    int longestCommonSubsequence(string text1, string text2) {
        m = text1.length();
        n = text2.length();
        vector<vector<int>>v(m+1,vector<int>(n+1,-1));//note m+1

        return LCS(text1, text2, 0, 0,v);
    }
};
```

Q14)

https://leetcode.com/problems/shortest-common-supersequence/solutions/3347772/c-lcs-type-short-sweet-easy-to-understand-c/

do by tabulation,try by

recursive closest:

```
class Solution {
public:
    int m, n;

    int solve(string& s1, string& s2, int i, int j, vector<vector<int>>& t) {
        if (i == m) {
            return n - j; // If s1 is exhausted, add the remaining characters of s2
        }
        if (j == n) {
            return m - i; // If s2 is exhausted, add the remaining characters of s1
        }

        if (t[i][j] != -1) {
            return t[i][j];
        }

        if (s1[i] == s2[j]) {
```

```
            return t[i][j] = 1 + solve(s1, s2, i + 1, j + 1, t);
        } else {
            return t[i][j] = 1 + min(solve(s1, s2, i + 1, j, t), solve(s1, s2, i, j + 1, t));
        }
    }

    string shortestCommonSupersequence(string str1, string str2) {
        m = str1.length();
        n = str2.length();

        vector<vector<int>> t(m + 1, vector<int>(n + 1, -1));
        solve(str1, str2, 0, 0, t);

        // Reconstruct the shortest common supersequence
        string scs;
        int i = 0, j = 0;

        while (i < m && j < n) {
            if (str1[i] == str2[j]) {
                scs += str1[i];
                i++;
                j++;
            } else {
                if (t[i + 1][j] < t[i][j + 1]) {
                    scs += str1[i];
                    i++;
                } else {
                    scs += str2[j];
                    j++;
                }
            }
        }

        // Add remaining characters from str1
        while (i < m) {
            scs += str1[i++];
        }

        // Add remaining characters from str2
        while (j < n) {
            scs += str2[j++];
        }

        return scs;
    }
};
```

Code story with mik:

```
class Solution {
public:
    int m, n;

    int solve(string& s1, string& s2, int i, int j, vector<vector<int>>& t) {
        // Base case: if either string is exhausted
        if (i == m || j == n) {
            return (m - i) + (n - j);
        }
```

```
    // Return memoized result if it exists
    if (t[i][j] != -1) {
        return t[i][j];
    }

    // If characters match
    if (s1[i] == s2[j]) {
        return t[i][j] = 1 + solve(s1, s2, i + 1, j + 1, t);
    } else {
        // If characters don't match, consider both possibilities and take the minimum
        return t[i][j] = 1 + min(solve(s1, s2, i + 1, j, t), solve(s1, s2, i, j + 1, t));
    }
  }

  // Function to find length of shortest common supersequence of two strings
  int shortestCommonSupersequence(string s1, string s2) {
    m = s1.length();
    n = s2.length();

    vector<vector<int>> t(m + 1, vector<int>(n + 1, -1));
    return solve(s1, s2, 0, 0, t);
  }
};
```

Q15)

https://leetcode.com/problems/edit-distance/

```
class Solution {
public:
  int m, n;

  int solve(string& s1, string& s2, int i, int j, vector<vector<int>>& t) {
    // If one of the strings is fully traversed, return the number of remaining characters in the other string
    if (i == m) return n - j;
    if (j == n) return m - i;

    // If result is already computed, return it
    if (t[i][j] != -1) return t[i][j];

    // If characters match, move to the next pair of characters
    if (s1[i] == s2[j]) {
      return t[i][j] = solve(s1, s2, i + 1, j + 1, t);
    }else{
    // Calculate costs of insert, delete, and replace operations
    int insertC = 1 + solve(s1, s2, i, j + 1, t);
    int deleteC = 1 + solve(s1, s2, i + 1, j, t);
    int replaceC = 1 + solve(s1, s2, i + 1, j + 1, t);

    // Return the minimum cost of the three operations
    return t[i][j] = min({insertC, deleteC, replaceC});
    }
  }
```

```
    int minDistance(string s1, string s2) {
        m = s1.length();
        n = s2.length();
        // Initialize memoization table with -1
        vector<vector<int>> t(m, vector<int>(n, -1));
        return solve(s1, s2, 0, 0, t);
    }
};
```

Q16)

https://leetcode.com/problems/palindromic-substrings/

```
class Solution {
public:
    int n;
    bool check(string &s, int i, int j,vector<vector<int>>&t) {
        if(i >= j) {
            return true;
        }

        if(t[i][j] != -1) {
            return t[i][j]; //1 : True, 0 : False
        }

        if(s[i] == s[j]) {
            return t[i][j] = check(s, i+1, j-1,t);
        }

        return t[i][j] = false;
    }

    int countSubstrings(string s) {
        n = s.length();
        vector<vector<int>>v(n+1,vector<int>(n+1,-1));
        int count = 0;
        for(int i = 0; i < n; i++) {
            for(int j = i; j < n; j++) { //check all possible substrings
                if(check(s, i, j,v)) {
                    count++;
                }
            }
        }

        return count;
    }
};
```

Q17)

https://leetcode.com/problems/longest-palindromic-substring/description/

```
class Solution {
public:
    vector<vector<int>> t;

    bool solve(string &s, int l, int r) {
        if (l >= r)
            return true;

        if (t[l][r] != -1) {
            return t[l][r];
```

```
        }

        if (s[l] == s[r]) {
            return t[l][r] = solve(s, l + 1, r - 1);
        }

        return t[l][r] = false;
    }

    string longestPalindrome(string s) {
        int n = s.length();

        int maxlen = INT_MIN;
        int startingIndex = 0;

        t = vector<vector<int>>(n, vector<int>(n, -1));

        for (int i = 0; i < n; i++) {
            for (int j = i; j < n; j++) {
                if (solve(s, i, j)) {
                    if (j - i + 1 > maxlen) {
                        startingIndex = i;
                        maxlen = j - i + 1;
                    }
                }
            }
        }

        return s.substr(startingIndex, maxlen);
    }
};
```

18) https://leetcode.com/problems/longest-palindromic-subsequence/description/

```
class Solution {
public:
    int LPS(string &s, int i, int j, vector<vector<int>> &t) {
        if (i > j)
            return 0;
        if (i == j)
            return 1;

        if (t[i][j] != -1)
            return t[i][j];
        if (s[i] == s[j])
            return t[i][j] = 2 + LPS(s, i + 1, j - 1, t);
        else
            return t[i][j] = max(LPS(s, i + 1, j, t), LPS(s, i, j - 1, t));
    }

    int longestPalindromeSubseq(string s) {
        int m = s.length();
        vector<vector<int>> t(m, vector<int>(m, -1));

        return LPS(s, 0, m - 1, t);
    }
};
```

19)

https://leetcode.com/problems/minimum-insertion-steps-to-make-a-string-palindrome/submissions/1341281891/

```
class Solution {
public:
    int solve(int i, int j, string &s, vector<vector<int>> &t) {
        if (i >= j)
            return 0;

        if (t[i][j] != -1)
            return t[i][j];

        if (s[i] == s[j])
            return t[i][j] = solve(i + 1, j - 1, s, t);

        return t[i][j] = 1 + min(solve(i, j - 1, s, t), solve(i + 1, j, s, t));
    }

    int minInsertions(string s) {
        int n = s.length();
        vector<vector<int>> t(n, vector<int>(n, -1));

        return solve(0, n - 1, s, t);
    }
};
```

Q20)

https://leetcode.com/problems/palindrome-partitioning/description/

solve by backtracking,DP approach is bottom up

CODE STORY WITH MIK over..

Now GFG+STRIVER

Q21)

https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/

```
class Solution {
public:
   int maxP(vector<int>& prices, int day, int n, int buy, vector<vector<int>>& t) {
      if (day >= n)
         return 0;

      int profit = 0;
      if (t[day][buy] != -1){
         return t[day][buy];
      }
      if(buy) { // buy
         int consider = maxP(prices, day + 1, n, false, t) - prices[day];
         int not_consider = maxP(prices, day + 1, n, true, t);
         profit = max({profit, consider, not_consider});
      } else { // sell
         int consider = maxP(prices, day + 2, n, true, t) + prices[day];
         int not_consider = maxP(prices, day + 1, n, false, t);
         profit = max({profit, consider, not_consider});
      }

      return t[day][buy] = profit;
   }

   int maxProfit(vector<int>& prices) {
      int n = prices.size();
      vector<vector<int>> t(n, vector<int>(2, -1));
      return maxP(prices, 0, n, true, t);
   }
};
```

Q22)

https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-transaction-fee/

```
#include <vector>
#include <algorithm>

using namespace std;

class Solution {
public:
   int FEE;

   int maxP(vector<int>& prices, int day, int n, int buy, vector<vector<int>>& t) {
      if (day >= n)
         return 0;

      if (t[day][buy] != -1) {
         return t[day][buy];
      }

      if (buy) { // buy
         int consider = maxP(prices, day + 1, n, 0, t) - prices[day];
         int not_consider = maxP(prices, day + 1, n, 1, t);
```

```
            t[day][buy] = max(consider,
not_consider);
        } else { // sell
            int consider = maxP(prices, day + 1, n,
1, t) + prices[day] - FEE;
            int not_consider = maxP(prices, day +
1, n, 0, t);
            t[day][buy] = max(consider,
not_consider);
        }

        return t[day][buy];
    }

    int maxProfit(vector<int>& prices, int fee) {
        int n = prices.size();
        vector<vector<int>> t(n, vector<int>(2, -
1));
        FEE = fee;
        return maxP(prices, 0, n, 1, t);
    }
};
```

Q23)

https://leetcode.com/problems/coin-change/

```
class Solution {
public:
    int solve(int i, vector<int>& coins, int
amount, vector<vector<int>>& t) {
        if (amount == 0)
            return 0;
        if (amount < 0 || i == coins.size())
            return INT_MAX;

        if (t[i][amount] != -1)
            return t[i][amount];

        int take = INT_MAX;
        if (coins[i] <= amount) {
            int res = solve(i, coins, amount -
coins[i], t);
            if (res != INT_MAX) {
                take = res + 1;
            }
        }

        int skip = solve(i + 1, coins, amount, t);

        return t[i][amount] = min(take, skip);
    }

    int coinChange(vector<int>& coins, int
amount) {
        int n = coins.size();
        vector<vector<int>> t(n + 1,
vector<int>(amount + 1, -1));
        int result = solve(0, coins, amount, t);
        return result == INT_MAX ? -1 : result;
    }
};
```

Q24)

https://leetcode.com/problems/coin-change-ii/

```
class Solution {
public:
    int n;

    int solve(int i, vector<int>& coins, int amount, vector<vector<int>>& t) {

        if (amount == 0)
            return t[i][amount] = 1;

        if (i == n || amount < 0)
            return 0;

        if (t[i][amount] != -1)
            return t[i][amount];

        if (coins[i] > amount)
            return t[i][amount] = solve(i + 1, coins, amount, t);

        int take = solve(i, coins, amount - coins[i], t);
        int skip = solve(i + 1, coins, amount, t);

        return t[i][amount] = take + skip;
    }

    int change(int amount, vector<int>& coins) {
        n = coins.size();
        vector<vector<int>> t(n + 1, vector<int>(amount+1, -1)); // Using vector instead of array
        return solve(0, coins, amount, t);
    }
};
```

25)

https://leetcode.com/problems/number-of-dice-rolls-with-target-sum/description/

```
class Solution {
public:
    long M = 1e9 + 7;

    int solve(int n, int k, int target, vector<vector<int>>& t) {
        if (target < 0 || n == 0)
            return target == 0 ? 1 : 0;

        if (t[n][target] != -1)
            return t[n][target];

        int sum = 0;

        for (int i = 1; i <= k; i++) {
            sum = (sum + solve(n - 1, k, target - i, t)) % M;
        }

        return t[n][target] = sum;
    }

    int numRollsToTarget(int n, int k, int target) {
```

```
        vector<vector<int>> t(n + 1, vector<int>(target + 1, -1));
        return solve(n, k, target, t);
    }
};
```

Q26)

https://leetcode.com/problems/target-sum/

```
class Solution {
public:
    int solve(vector<int>& nums, int target, int sum, int i, vector<vector<int>>& t) {
        if (i == nums.size()) {
            return sum == target ? 1 : 0;
        }

        if (t[i][sum + 1000] != -1) {
            return t[i][sum + 1000];
        }

        int add = solve(nums, target, sum + nums[i], i + 1, t);
        int subtract = solve(nums, target, sum - nums[i], i + 1, t);

        t[i][sum + 1000] = add + subtract;
        return t[i][sum + 1000];
    }

    int findTargetSumWays(vector<int>& nums, int target) {
        int n = nums.size();
        vector<vector<int>> t(n, vector<int>(2001, -1)); // Adjusted size for possible sums from -1000 to 1000
        return solve(nums, target, 0, 0, t);
    }
};
```

27)

https://leetcode.com/problems/partition-equal-subset-sum/

```
class Solution {
public:
    bool solve(vector<int>& nums, int target, int i, vector<vector<int>>& t) {
        if (target == 0) {
            return true;
        }
        if (i >= nums.size() || target < 0) {
            return false;
        }

        if (t[i][target] != -1) {
            return t[i][target];
        }

        // Option 1: Include the current number in the subset
        bool include = solve(nums, target - nums[i], i + 1, t);
        // Option 2: Exclude the current number from the subset
        bool exclude = solve(nums, target, i + 1, t);
```

```
        t[i][target] = include || exclude;
        return t[i][target];
    }

    bool canPartition(vector<int>& nums) {
        int total = 0;
        for (int num : nums) {
            total += num;
        }

        if (total % 2 != 0) {
            return false; // Total sum is odd, cannot partition
        }

        int target = total / 2;
        vector<vector<int>> t(nums.size(), vector<int>(target + 1, -1));
        return solve(nums, target, 0, t);
    }
};
```

28)

https://leetcode.com/problems/minimum-cost-to-cut-a-stick/

```
class Solution {
public:
    // Update the function signature to include the memoization table as a parameter
    int solve(const vector<int>& cuts, int left, int right, vector<vector<int>>& t) {
        if (right - left == 1)
            return 0;
        if (t[left][right] != -1)
            return t[left][right];

        int result = INT_MAX;

        for (int index = left + 1; index <= right - 1; index++) {
            int cost = solve(cuts, left, index, t) + solve(cuts, index, right, t) + (cuts[right] - cuts[left]);
            result = min(result, cost);
        }

        return t[left][right] = result;
    }

    int minCost(int n, vector<int>& cuts) {
        sort(cuts.begin(), cuts.end());

        cuts.insert(cuts.begin(), 0);
        cuts.push_back(n);

        int m = cuts.size();
        vector<vector<int>> t(m, vector<int>(m, -1)); // Initialize the memoization table with size based on number of cuts

        return solve(cuts, 0, m - 1, t);
    }
};
```

29)

https://leetcode.com/problems/unique-paths/submissions/1343649957/

```
class Solution {
public:
    int solve(int m, int n, int i, int j, vector<vector<int>>& t) {
        if(i >= m || j >= n || i < 0 || j < 0)
            return 0;

        if(i == m-1 && j == n-1)
            return 1;

        if(t[i][j] != -1)
            return t[i][j];

        return t[i][j] = solve(m, n, i+1, j, t) + solve(m, n, i, j+1, t);
    }

    int uniquePaths(int m, int n) {
        vector<vector<int>> t(m, vector<int>(n, -1));
        return solve(m, n, 0, 0, t);
    }
};
```

30)

https://leetcode.com/problems/pascals-triangle-ii/description/

```
class Solution {
public:
    vector<int> getRow(int rowIndex) {
        vector<vector<int>> result(rowIndex+1);

        for(int i = 0; i<rowIndex+1; i++) {
            result[i] = vector<int>(i+1, 1);

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

            if(i == rowIndex) //Just return when you find rowIndex
                return result[i];
        }

        return {};
    }
};
```

Q31)
https://leetcode.com/problems/unique-paths-ii/description/

```
class Solution {
public:
    int m, n;

    int solve(vector<vector<int>>& obstacleGrid, int i, int j, vector<vector<int>>& t) {
        if(i >= m || j >= n || obstacleGrid[i][j] != 0) {
            return 0;
        }
```

```
        if(t[i][j] != -1)
            return t[i][j];

        if(i == m-1 && j == n-1)
            return 1;

        int right = solve(obstacleGrid, i, j+1, t);
        int down  = solve(obstacleGrid, i+1, j, t);

        return t[i][j] = right + down;
    }

    int uniquePathsWithObstacles(vector<vector<int>>& obstacleGrid) {
        m = obstacleGrid.size();
        n = obstacleGrid[0].size();

        vector<vector<int>> t(m, vector<int>(n, -1));

        return solve(obstacleGrid, 0, 0, t);
    }
};
```

32)
https://leetcode.com/problems/minimum-falling-path-sum/description/

Will give tle in normal
https://en.wikipedia.org/wiki/Seam_carving
this algo is used here kinda interesting test case to think about

```
class Solution {
public:
    int MFS(vector<vector<int>>& A, int row, int col, vector<vector<int>>& t) {
        if (row == A.size() - 1) // last row
            return A[row][col]; // return the value
        if (t[row][col] != -1)
            return t[row][col];

        int minSum = INT_MAX;

        for (int shift = -1; shift <= 1; shift++) {
            if (col + shift >= 0 && col + shift < A[row].size()) {
                minSum = min(minSum, A[row][col] + MFS(A, row + 1, col + shift, t));
            }
        }

        return t[row][col] = minSum;
    }

    int minFallingPathSum(vector<vector<int>>& A) {
        int m = A.size(); // row
        int n = m; // column
        vector<vector<int>> t(101, vector<int>(101, -1)); // Initialize with -1

        int result = INT_MAX;
        for (int col = 0; col < n; col++) {//trying all possible things
            result = min(result, MFS(A, 0, col, t));
        }
```

```
        return result;
    }
};
```

33)

https://leetcode.com/problems/triangle/description/

```
class Solution {
public:
    int solve(vector<vector<int>>& triangle, int row, int col, vector<vector<int>>& t) {
        if (row == triangle.size() - 1) // If we are at the last row
            return triangle[row][col];  // Return the value at the current position

        if (t[row][col] != -1) // If the result is already computed
            return t[row][col];

        // Calculate the minimum path sum for the current position
        int leftPath = solve(triangle, row + 1, col, t);
        int rightPath = solve(triangle, row + 1, col + 1, t);

        t[row][col] = triangle[row][col] + min(leftPath, rightPath);
        return t[row][col];
    }
    int minimumTotal(vector<vector<int>>& triangle) {
        int n = triangle.size(); // Number of rows in the triangle
        vector<vector<int>> t(n, vector<int>(n, -1)); // Initialize the memoization table with -1

        // Start from the top of the triangle
        return solve(triangle, 0, 0, t);
    }
};
```

34)

https://leetcode.com/problems/maximum-number-of-points-with-cost/description/

try this one