LeetCode is a powerful tool for anyone looking to improve their coding skills and prepare for technical interview. It offers a various set of problems that span various difficulty levels & topics, including algorithms, data structures, databases & more.

Search in a Sorted 2D matrix

* Place the 2 pointers, low & high. low point to 0 and high point to (N*M)-1
* Calculate the mid. mid = (low + high)/2
* row = mid / M, col = mid % M

If matrix [row][col] == target, return true
If matrix [row][col] < target, low = mid+1
If matrix [row][col] > target, high = mid-1

```
bool SearchMatrix (vector<vector<int>>& matrix, int target){
    int n = matrix.size();
    int m = matrix[0].size();
    int low = 0, high = m*n -1;
    while (low <= high){
        int mid = (low+high)/2;
        int row = mid/m, col = mid%m;
        if (matrix[row][col] == target) return true;
        else if (matrix[row][col] < target) low = mid+1;
        else high = mid-1;
    }
    return false;
}
```

Word Search → Given an m*n grid of characters board and string word, return true if the word exists in the grid.

* Find the first character of the given string
* Start backtracking in all four directions until we find all the letters of sequentially adjacent cells.
* If we found our result return true else return false.

```
bool search (vector<vector<char>>& board, string word, int row, int
    col, int index, int m, int n){
    if (index == word.length()) return true;
    if (row < 0 || col < 0 || row == m || col == n
        || board[row][col] != word[index] or board[row][col] == '!')
            return false;
    char c = board[row][col];
    board[row][col] = '!';
    bool top = search(board, word, row-1, col, index+1, m, n);
    bool right = search(board, word, row, col+1, index+1, m, n);
    bool bottom = search(board, word, row+1, col, index+1, m, n);
    bool left = search(board, word, row, col-1, index+1, m, n);
    board[row][col] = c;
    return top || right || bottom || left;
}
bool exist(vector<vector<char>> board, string word){
        int m = board.size();
        int n = board[0].size();
        int index = 0;
```

```
for (int i = 0 ; i < m ; i++){
    for (int j = 0; j < n ; j++){
        if ( board[i][j] == word[index]){
            if ( search ( board, word, i, j, index, m, n))
                return true;
        }
    }
}
return false;
```

Longest Common Subsequence

s1 = a d e b c (ind1) s2 = d c a d m n (ind2)

f(ind1, ind2)

Explore all possibilities at a given index

if [s1[ind1] == s2[ind2]) return 1 + f(ind1-1, ind2-1)}

if (s1[ind1] != s2[ind2]) return 0 + max(f(ind1-1, ind2),
f(ind1, ind2-1))

Base Case

if (ind1 < 0 || ind2 < 0) return 0

```
int find ( string &s1 , string& s2 , int ind1 , int ind2 , vector< vector >>&dp)
{ if (ind1 < 0 || ind2 < 0) return 0;
  if ( dp [ind1][ind2] != -1) return dp[ind1][ind2];
  if ( s1[ind1] == s2[ind2]) return dp[ind1][ind2] = 1 +
                                find (s1, s2, ind1-1, ind2-1, dp);
  else
    return dp[ind1][ind2] = max ( find (s1, s2, ind1, ind2-1, dp),
                                  find (s1, s2, ind1-1, ind2, dp));
}
```