79. Word Search
We will use dfs to solve this question. TC is O(n**2), SC is O(n**2)

```
var exist = function(board, word) {
    function dfs(cur_i, cur_j, visited, cur_index) {
      if (cur_index === word.length) {
        return true;
      }

      for (let [d_i, d_j] of [[0, 1], [0, -1], [1, 0], [-1, 0]]) {
        let new_i = cur_i + d_i, new_j = cur_j + d_j;
        if (new_i >= 0 && new_i < board.length && new_j >= 0 && new_j < board[0].length &&
!visited[[new_i,new_j]]) {
          if (board[new_i][new_j] === word[cur_index]) {
            visited[[new_i, new_j]] = true;
            if (dfs(new_i, new_j, visited, cur_index + 1)) {
              return true;
            }
            visited[[new_i, new_j]] = false;
          }
        }
      }
      return false;
    }
    const visited = {};
    for (let i = 0; i < board.length; i++) {
      for (let j = 0; j < board[0].length; j++) {
        if (board[i][j] === word[0]) {
          visited[[i, j]] = true;
          if (dfs(i, j, visited, 1)) {
            return true;
          }
          visited[[i,j]] = false;
        }
      }
    }
  return false;
};
```

17. Letter Combinations of a Phone Number
We will use dfs to go through all possibilities. TC is O(3**n).

```
/**
 * @param {string} digits
 * @return {string[]}
```

```
 */

var letterCombinations = function(digits) {
    const map = ['', '', 'abc', 'def', 'ghi', 'jkl', 'mno', 'pqrs', 'tuv', 'wxyz'];
    const result = [];
    const prefix = [];
    if (digits.length > 0) {
      traverse(0);
    }
    return result;

    function traverse(cur_idx) {
      if (cur_idx === digits.length) {
        result.push(prefix.join(''));
      } else {
        const str = map[digits[cur_idx] - '0'];
        for (let j = 0; j < str.length; j++) {
          prefix.push(str[j]);
          traverse(cur_idx + 1);
          prefix.pop();
        }
      }
    }
};
```

17. Letter Combinations of a Phone Number .2
We will use iteration solution to solve this question. TC is O(3**n)

```
var letterCombinations = function(digits) {
    const map = ['', '', 'abc', 'def', 'ghi', 'jkl', 'mno', 'pqrs', 'tuv', 'wxyz'];
    let result = [''];
    let next = [];
    for (let i = 0; i < digits.length; i++) {
      const s = map[digits[i] - '0'];
      next = [];
      result.forEach((cur_s) => {
        for (let j = 0; j < s.length; j++) {
          next.push(cur_s + s[j]);
        }
      })
      result = next;
    }
    return next;
};
```

46. Permutations
We will solve it iteratively. TC is O(n).

```
/**
 * @param {number[]} nums
 * @return {number[][]}
 */
var permute = function(nums) {
    let result = [[]];
    let next = [];
    for (let i = 0; i < nums.length; i++) {
        const cur_len = result[0].length;
        next = [];
        result.forEach((cur_a) => {
            for (let j = 0; j <= cur_len; j++) {
                next.push([...cur_a.slice(0, j), nums[i], ...cur_a.slice(j)] );
            }
        })
        result = next;
    }
    return next;
};
```

METHOD 2:
We will use recursive with a set to check whether our element exists in our set now. TC is O(n^2)

```
var permute = function(nums) {
    const result = [];
    helper([], new Set());
    return result;
    function helper(cur, cur_set) {
        if (cur.length === nums.length) {
            result.push(cur.slice(0));
        } else {
            for (let i of nums) {
                if (!cur_set.has(i)) {
                    cur.push(i);
                    cur_set.add(i);
                    helper(cur, cur_set);
                    cur_set.delete(i);
                    cur.pop();
                }
            }
        }
```

```
    }
};
```