JavaScript Problem Set 6 (PS6) Solutions

Problem 1 Fibonacci (Recursion & DP)

Write a function fib(n) to return the n-th Fibonacci number.

Solution:

```
// Recursive solution
function fib(n) {
  if (n <= 1) return n;
  return fib(n - 1) + fib(n - 2);
}

// Optimized with memoization
function fibMemo(n, memo = {}) {
  if (n in memo) return memo[n];
  if (n <= 1) return n;
  memo[n] = fibMemo(n - 1, memo) + fibMemo(n - 2, memo);
  return memo[n];
}</pre>
```

Problem 2 Climbing Stairs (DP)

You are climbing a staircase. It takes 1 or 2 steps at a time. Write a function climbStairs(n) that returns the number of ways to reach the top.

```
function climbStairs(n) {
  let dp = Array(n + 1).fill(0);
  dp[0] = 1;
  dp[1] = 1;
  for (let i = 2; i <= n; i++) {
    dp[i] = dp[i - 1] + dp[i - 2];
  }
  return dp[n];
}</pre>
```

Problem 3 Binary Search

Write a function binarySearch(arr, target) that returns the index of the target value in a sorted array, or -1 if the target isn't found.

Solution:

```
function binarySearch(arr, target) {
  let left = 0, right = arr.length - 1;
  while (left <= right) {
    let mid = Math.floor((left + right) / 2);
    if (arr[mid] === target) return mid;
    else if (arr[mid] < target) left = mid + 1;
    else right = mid - 1;
  }
  return -1;
}</pre>
```

Problem 4 Permutations (Backtracking)

Write a function permute(nums) that returns all possible permutations of an array of numbers.

Solution:

```
function permute(nums) {
  let result = [];
  function backtrack(temp = []) {
    if (temp.length === nums.length) {
      result.push([...temp]);
      return;
    }
    for (let i = 0; i < nums.length; i++) {
      if (temp.includes(nums[i])) continue;
      temp.push(nums[i]);
      backtrack(temp);
      temp.pop();
    }
 backtrack();
  return result;
}
```

Problem 5 Max Subarray Product

Write a function maxProduct(nums) that returns the maximum product of a contiguous subarray.

```
function maxProduct(nums) {
  let maxProd = nums[0], minProd = nums[0], result = nums[0];
  for (let i = 1; i < nums.length; i++) {
    let temp = maxProd;
    maxProd = Math.max(nums[i], maxProd * nums[i], minProd * nums[i]);
    minProd = Math.min(nums[i], temp * nums[i], minProd * nums[i]);
    result = Math.max(result, maxProd);
  }
  return result;
}</pre>
```

Problem 6 Coin Change (DP)

Given an integer array coins representing coins of different denominations and an integer amount, return the fewest number of coins that you need to make up that amount.

Solution:

```
function coinChange(coins, amount) {
  let dp = Array(amount + 1).fill(Infinity);
  dp[0] = 0;
  for (let coin of coins) {
    for (let i = coin; i <= amount; i++) {
       dp[i] = Math.min(dp[i], dp[i - coin] + 1);
    }
  }
  return dp[amount] === Infinity ? -1 : dp[amount];
}</pre>
```

Problem 7 Find All Binaries

Write a function generateBinary(n) that returns all possible binary combinations.

```
function generateBinary(n) {
  let result = [];
  function backtrack(current = '') {
    if (current.length === n) {
      result.push(current);
      return;
    }
    backtrack(current + '0'); // choose 0
    backtrack(current + '1'); // choose 1
```

```
}
backtrack();
return result;
}
```

Problem 8 Find All Subsets (Backtracking)

Write a function subsets(nums) that returns all possible subsets of a list of numbers.

Solution:

```
function subsets(nums) {
  let result = [];
  function backtrack(start = 0, temp = []) {
    result.push([...temp]);
    for (let i = start; i < nums.length; i++) {
       temp.push(nums[i]);
       backtrack(i + 1, temp);
       temp.pop();
    }
  }
  backtrack();
  return result;
}</pre>
```

Problem 9 Divide and Conquer – Merge Sort

Implement the merge sort algorithm. It should return the sorted array.

```
function mergeSort(arr) {
  if (arr.length <= 1) return arr;
  let mid = Math.floor(arr.length / 2);
  let left = mergeSort(arr.slice(0, mid));
  let right = mergeSort(arr.slice(mid));
  return merge(left, right);
}

function merge(left, right) {
  let result = [], i = 0, j = 0;
  while (i < left.length && j < right.length) {
    if (left[i] < right[j]) result.push(left[i++]);
    else result.push(right[j++]);
  }
  return result.concat(left.slice(i), right.slice(j));
}</pre>
```

Problem 10 Search in Rotated Sorted Array

Write a function search(nums, target) that searches for a target in a rotated sorted array. Return the index if found, or -1.

Solution:

```
function search(nums, target) {
  let left = 0, right = nums.length - 1;
  while (left <= right) {</pre>
    let mid = Math.floor((left + right) / 2);
    if (nums[mid] === target) return mid;
    if (nums[left] <= nums[mid]) {</pre>
      if (nums[left] <= target && target < nums[mid]) right = mid -
         1:
      else left = mid + 1;
    } else {
      if (nums[mid] < target && target <= nums[right]) left = mid +</pre>
      else right = mid - 1;
    }
  }
  return -1;
}
```

Problem 11 N-Queens (Backtracking)

The N-Queens problem is a puzzle where you place N queens on an $N \times N$ chessboard so that no two queens threaten each other. Write a function solveNQueens(n) that returns all distinct solutions to the N-Queens puzzle.

```
function solveNQueens(n) {
  let result = [];
  function backtrack(row = 0, board = []) {
    if (row === n) {
      result.push(board.map(row => row.join('')));
      return;
  }
  for (let col = 0; col < n; col++) {
    if (isValid(board, row, col)) {
      board[row][col] = 'Q';
      backtrack(row + 1, board);
      board[row][col] = '.';
    }
  }
  }
}</pre>
```

```
function isValid(board, row, col) {
   for (let i = 0; i < row; i++) {
     if (board[i][col] === 'Q' ||
        board[i][col - (row - i)] === 'Q' ||
        board[i][col + (row - i)] === 'Q') {
        return false;
     }
   }
   return true;
}
backtrack();
return result;
}</pre>
```