Assignment 6 (2D Arrays) Solution

Question 1

A permutation perm of n + 1 integers of all the integers in the range [0, n] can be represented as a string s of length n where:

- s[i] == 'I' if perm[i] < perm[i + 1], and
- s[i] == 'D' if perm[i] > perm[i + 1].

Given a string s, reconstruct the permutation perm and return it. If there are multiple valid permutations perm, return **any of them**.

Example 1:

Input: s = "IDID"

Output:

[0,4,1,3,2]

Solution:

```
 public int[] diStringMatch(String str) {


        int s = 0;

        int e = str.length();

        int[] nums = new int[str.length() + 1];


        for(int i = 0;i<str.length();i++){

            if(str.charAt(i) == 'I'){

                nums[i] = s;

                s++;

            }else{
                nums[i] = e;

                e--;

            }

        }

        nums[str.length()] = s;

        return nums;

    }
```

Question 2

You are given an m x n integer matrix matrix with the following two properties:

- Each row is sorted in non-decreasing order.
- The first integer of each row is greater than the last integer of the previous row.

Given an integer target, return true *if* target *is in* matrix *or* false *otherwise*.

You must write a solution in O(log(m * n)) time complexity.

Solution:

```
public boolean searchMatrix(int[][] matrix, int target) {

        if(matrix.length == 0) return false;


        int rows = matrix.length;

        int columns = matrix[0].length;


        int low = 0;
        int high = rows * columns;


        while(low < high) {
            int mid = (low+high)/2;


            if(matrix[mid/columns][mid%columns] == target) {
                return true;
            } else if (matrix[mid/columns][mid%columns] < target) {
                low = mid+1;
            } else {
                high = mid;
            }
        }
        return false;
    }
```

Question 3 Given an array of integers arr, return *true if and only if it is a valid mountain array*.

Recall that arr is a mountain array if and only if:

- arr.length >= 3
- There exists some i with 0 < i < arr.length - 1 such that:
  - arr[0] < arr[1] < ... < arr[i - 1] < arr[i]
  - arr[i] > arr[i + 1] > ... > arr[arr.length - 1]

Solution:

```
public boolean validMountainArray(int[] arr) {
      //if size is < 2 then it not mountain
    if(arr.length<3) return false;



    int topidx=0;
    int top=0;

    //find max value and that index
    for(int i=0;i<arr.length;i++)
        {
            if(arr[i]>top)
            {
            top = arr[i];
            topidx=i;
            }
        }

    //check that one side mountain or not .
    if(top==arr[arr.length-1] || top==arr[0]) return false;

            //check perfact mountain or not
            int i=0;
            while(i<topidx)
            {
                if(arr[i] >= arr[i+1]) return false;
                i++;


            }


            while(topidx<arr.length-1)
            {
                if(arr[topidx] <= arr[topidx+1]) return false;
                topidx++;


            }
            return true;
    }
```

Question 4

Given a binary array nums, return *the maximum length of a contiguous subarray with an equal number of* 0 *and* 1.

Solution:

```
public int findMaxLength(int[] nums) {
        int count = 0;
        for (int i = 0; i < nums.length; i++) {
            int zeros = 0, ones = 0;

            for (int j = i; j < nums.length; j++) {

                if (nums[j] == 0) {

                    zeros++;

                } else {

                    ones++;

                }

                if (zeros == ones) {

                    count = Math.max(count, j - i + 1);

                }

            }

        }

        return count;

    }
```

Question 5

The **product sum** of two equal-length arrays a and b is equal to the sum of a[i] * b[i] for all 0 <= i < a.length (**0-indexed**).

- For example, if a = [1,2,3,4] and b = [5,2,3,1], the **product sum** would be 1*5 + 2*2 + 3*3 + 4*1 = 22.

Given two arrays nums1 and nums2 of length n, return *the **minimum product sum** if you are allowed to **rearrange** the **order** of the elements in* nums1.

Solution:

```
public int minPairSum(int[] nums) {
        Arrays.sort(nums);

        int maxPairSum=0;

        for(int i=0;i<=nums.length/2;i++){

            maxPairSum=Math.max(maxPairSum, nums[i] + nums[nums.length-1-i] );

        }

        return maxPairSum;

    }
```

Question 6

An integer array original is transformed into a **doubled** array changed by appending **twice the value** of every element in original, and then randomly **shuffling** the resulting array.

Given an array changed, return original *if* changed *is a* ***doubled*** *array. If* changed *is not a* ***doubled*** *array, return an empty array. The elements in* original *may be returned in* ***any*** *order*.

Solution:

```
 public int[] findOriginalArray(int[] nums) {

       int[] vacarr = new int[0];

                   // when we need to return vacant array

        int n= nums.length;

                            // size of the array

        if(n%2!=0)

        {

            return vacarr;
```

```
                            // when we will have odd number of integer in our
input(double array can't be in odd number)


        }

        HashMap<Integer, Integer> hm = new HashMap<Integer,
Integer>();

                             // for storing the frequencies of each input

        int[] ans = new int[(nums.length/2)];

        // answer storing array


        for(int i=0;i<n;i++)

        {

            hm.put(nums[i], hm.getOrDefault(nums[i],0)+1);


        }

        int temp = 0;


        Arrays.sort(nums);


        for(int i: nums)

        {


            if(hm.get(i)<=0)

            {
                continue;
            }

            if(hm.getOrDefault(2*i,0)<=0)
            {   // if we have y but not y*2 return vacant array
                return vacarr;
            }
            ans[temp++] = i;
                            // if we have both y and y*2, store in our ans array
           // decrease the frequency of y and y*2
            hm.put(i, hm.get(i)-1);
            hm.put(2*i, hm.get(2*i)-1);
        }

        return ans;
    }
```

Question 7

Given a positive integer n, generate an n x n matrix filled with elements from 1 to n2 in spiral order.

Solution:

```
public int[][] generateMatrix(int n) {
        int[][] matrix = new int[n][n];
        int num = 1;
        int row = 0;
        int col = 0;
        int direction = 0;
        int[] dr = {0, 1, 0, -1};
        int[] dc = {1, 0, -1, 0};
        while (num <= n * n) {
            matrix[row][col] = num;
            num++;
            int nextRow = row + dr[direction];
            int nextCol = col + dc[direction];
            if (nextRow < 0 || nextRow >= n || nextCol < 0 || nextCol >= n || matrix[nextRow][nextCol] != 0) {
                direction = (direction + 1) % 4;
            }
            row += dr[direction];
            col += dc[direction];
        }
        return matrix;
    }
```

Question 8

Given two sparse matrices mat1 of size m x k and mat2 of size k x n, return the result of mat1 x mat2. You may assume that multiplication is always possible

Solution:

```
public int[][] multiply(int[][] A, int[][] B) {
        int m = A.length;
        int n = A[0].length;
        int l = B[0].length;
        int[][] C = new int[m][l];

        for(int i=0; i< m; i++){
            for(int j = 0; j< n; j++){
                if(A[i][j] != 0){
                    for(int k = 0; k<l; k++)
                        C[i][k] += A[i][j]* B[j][k];
                }
            }
        }

        return C;
    }
```