Assignment 2 (Arrays) Solution

Question 1 Given an integer array nums of 2n integers, group these integers into n pairs (a1, b1), (a2, b2),..., (an, bn) such that the sum of min(ai, bi) for all i is maximized. Return the maximized sum

Solution:

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

        Arrays.sort(nums);

        int sum = 0;

        for(int i = 0;i<nums.length;i= i +2){
            sum = sum + nums[i];
        }

    return sum;
    }
```

Question 2 Alice has n candies, where the ith candy is of type candyType[i]. Alice noticed that she started to gain weight, so she visited a doctor. The doctor advised Alice to only eat n / 2 of the candies she has (n is always even). Alice likes her candies very much, and she wants to eat the maximum number of different types of candies while still following the doctor's advice. Given the integer array candyType of length n, return the maximum number of different types of candies she can eat if she only eats n / 2 of them.

Solution:

```
 public int distributeCandies(int[] candyType) {

         HashSet<Integer> set = new HashSet<>();

         for(int i = 0;i<candyType.length;i++){
             set.add(candyType[i]);
```

```
        }

       int distinctCan = set.size();

       int noOfCandies = candyType.length/2;

       if(noOfCandies >= distinctCan){
          return distinctCan;
       }else{
          return noOfCandies;
       }

    }
```

Question 3 We define a harmonious array as an array where the difference between its maximum value and its minimum value is exactly 1. Given an integer array nums, return the length of its longest harmonious subsequence among all its possible subsequences. A subsequence of an array is a sequence that can be derived from the array by deleting some or no elements without changing the order of the remaining elements.
Solution:

```
public int findLHS(int[] nums) {
       HashMap<Integer,Integer> map = new HashMap<Integer,
Integer>();

       int result = 0;

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

          if(map.containsKey(nums[i])){
             map.put(nums[i],map.get(nums[i])+1);
          }else{
             map.put(nums[i],1);
             // arr.add(nums[i]);
          }
        }

        for(int i = 0;i<nums.length;i++){
          int val = map.get(nums[i]);
          if(val + map.getOrDefault(nums[i]+1,Integer.MIN_VALUE) >
result ){
            result = val +
map.getOrDefault(nums[i]+1,Integer.MIN_VALUE);
          }
          else if(val + map.getOrDefault(nums[i]-1,Integer.MIN_VALUE) >
result){
            result = val +
map.getOrDefault(nums[i]-1,Integer.MIN_VALUE);
          }
        }

        return result;


    }
```

Question 4 You have a long flowerbed in which some of the plots are planted, and some are not. However, flowers cannot be planted in adjacent plots. Given an integer array flowerbed containing 0's and 1's, where 0 means empty and 1 means not empty, and an integer n, return true if n new flowers can be planted in the flowerbed without violating the no-adjacent-flowers rule and false otherwise.

Solution:

```
 public boolean canPlaceFlowers(int[] flowerbed, int n) {
        int count = 0;
        int m = flowerbed.length;
        if(m==1 && flowerbed[0]==0) {
          return true;
        }
        for(int i=0; i<m; i++) {
            if(flowerbed[i] == 0) {
                if(i==0 && flowerbed[i+1]!=1) {
                    count++;
                    flowerbed[i]=1;
                }
                else if(i==m-1 && flowerbed[i-1]!=1) {
                    count++;
                    flowerbed[i]=1;
                }
                else if(i!=m-1 && i!=0 && flowerbed[i+1]!=1 &&
flowerbed[i-1]!=1) {
                    count++;
                    flowerbed[i]=1;
                }
            }
        }
        System.out.println(count);
        if(count>=n) {
            return true;
        }
        else {
            return false;
        }
    }
```

Question 5 Given an integer array nums, find three numbers whose product is maximum and return the maximum product.

Solution:

```
 public int maximumProduct(int[] nums) {
        Arrays.sort(nums);
        int case1 = nums[0]*nums[1]*nums[nums.length-1];
        int case2 =
nums[nums.length-1]*nums[nums.length-2]*nums[nums.length-3];

        int maxProduct = Integer.max(case1, case2);
        return maxProduct;
    }
```

Question 6 Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1.

Solution:

```
public int search(int[] nums, int target) {

        int high = nums.length - 1;
        int low = 0;

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

            if(nums[mid] == target){
                return mid;
            }
            else if(nums[mid] > target){
                high = mid -1;
            }else{
                low = mid + 1;
            }

        }
        return -1;
    }
```

Question 7 An array is monotonic if it is either monotone increasing or monotone decreasing. An array nums is monotone increasing if for all i <= j, nums[i] <= nums[j]. An array nums is monotone decreasing if for all

i <= j, nums[i] >= nums[j]. Given an integer array nums, return true if the given array is monotonic, or false otherwise.
Solution:

```
public boolean isMonotonic(int[] nums) {

        int increasingMono = 0;
        int decreasingMono = 0;

        for(int i = 0;i<nums.length-1;i++){

            if(nums[i] <= nums[i+1]){
                increasingMono++;
            }
            if(nums[i] >= nums[i+1]){
                decreasingMono++;
            }
        }

        System.out.println(decreasingMono);
        System.out.println(increasingMono);


        if(increasingMono == nums.length - 1 || decreasingMono ==
nums.length -1){
            return true;
        }
        return false;

    }
```

Question 8 You are given an integer array nums and an integer k. In one operation, you can choose any index i where 0 <= i < nums.length and change nums[i] to nums[i] + x where x is an integer from the range [-k, k]. You can apply this operation at most once for each index i. The score of nums is the difference between the maximum and minimum elements in nums. Return the minimum score of nums after applying the mentioned operation at most once for each index in it.

Solution:

```
 public int smallestRangeI(int[] nums, int k) {
        int min = 10001, max = -1;
        // 'max - k' would give us the minimum max
        // 'min + k' would give us the maximum min
        // Their difference would give us the minimum score (difference)
        for (int i : nums) {
            min = Math.min(min, i);
            max = Math.max(max, i);
        }

        int ans = (max - k) - (min + k);
        return Math.max(0, ans);
    }
```