

Question 1:

Given an integer array `nums` of $2n$ integers, group these integers into n pairs $(a_1, b_1), (a_2, b_2), \dots, (a_n, b_n)$ such that the sum of $\min(a_i, b_i)$ for all i is maximized. Return the maximized sum.

Example 1: Input: `nums = [1,4,3,2]` Output: 4

Explanation: All possible pairings (ignoring the ordering of elements) are:

1. $(1, 4), (2, 3) \rightarrow \min(1, 4) + \min(2, 3) = 1 + 2 = 3$
2. $(1, 3), (2, 4) \rightarrow \min(1, 3) + \min(2, 4) = 1 + 2 = 3$
3. $(1, 2), (3, 4) \rightarrow \min(1, 2) + \min(3, 4) = 1 + 3 = 4$ So the maximum possible sum is 4

```
class Solution {
    public int arrayPairSum(int[] nums) {
        int max = 0;
        Arrays.sort(nums);
        for(int i = 0; i < nums.length; i++)
            System.out.println(nums[i]);
        for(int i = 0; i < nums.length-1; i=i+2){
            max += Math.min(nums[i], nums[i+1]);
        }
        return max;
    }
}
```

Question 2:

Alice has n candies, where the i th 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.

Example 1: Input: `candyType = [1,1,2,2,3,3]` Output: 3 **Explanation:** Alice can only eat $6 / 2 = 3$ candies. Since there are only 3 types, she can eat one of each type.

```

class Solution {
    public int distributeCandies(int[] candyType) {
        HashSet<Integer> set= new HashSet<>();
        int eat= candyType.length /2;
        for(int i:candyType ) set.add(i);

        return Math.min(set.size(), eat);

    }
}

```

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. **Example 1:** Input: nums = [1,3,2,2,5,2,3,7] Output: 5 **Explanation:** The longest harmonious subsequence is [3,2,2,2,3].

```

class Solution {
    public int findLHS(int[] nums) {
        int max= 0;
        HashMap<Integer, Integer> map = new HashMap<>();
        for(int i= 0; i< nums.length; i++){
            map.put(nums[i], map.getOrDefault(nums[i], 0)+1);
        }
        for(int k: map.keySet()){
            if(map.containsKey(k+1)){
                max= Math.max(max, (map.get(k)+map.get(k+1)));
            }
        }
        return max;

    }
}

```

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. **Example 1:** Input: flowerbed = [1,0,0,0,1], n = 1 Output: true

```
class Solution {
    public boolean canPlaceFlowers(int[] flowerbed, int n) {
        if(n==0) return true;
        for(int i= 0; i<flowerbed.length; i++){
            if(flowerbed[i]==0 && (i==0||flowerbed[i-1]==0)
&&(i==flowerbed.length-1 ||
                                flowerbed[i+1]==0))
            {
                n--;
            }

            if(n==0) return true;
            flowerbed[i]=1;
        }
    }

    return false;
}
```

Question 5 Given an integer array nums, find three numbers whose product is maximum and return the maximum product. **Example 1:** Input: nums = [1,2,3] Output: 6

```
class Solution {
    public int maximumProduct(int[] nums) {
        int min1 = Integer.MAX_VALUE, min2 =
Integer.MAX_VALUE; // negative min values
        int max1 = Integer.MIN_VALUE, max2 =
Integer.MIN_VALUE, max3 = Integer.MIN_VALUE;
        for (int n: nums) {
            if (n <= min1) {
                min2 = min1;
                min1 = n;
            } else if (n <= min2) {           // n lies between min1
and min2
        }
```

```

        min2 = n;
    }
    if (n >= max1) {                // n is greater than
max1, max2 and max3
        max3 = max2;
        max2 = max1;
        max1 = n;
    } else if (n >= max2) {        // n lies between max1
and max2
        max3 = max2;
        max2 = n;
    } else if (n >= max3) {        // n lies between max2
and max3
        max3 = n;
    }
}
return Math.max(min1 * min2 * max1, max1 * max2 *
max3);

}
}

```

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. You must write an algorithm with $O(\log n)$ runtime complexity. Input: nums = [-1,0,3,5,9,12], target = 9 Output: 4
Explanation: 9 exists in nums and its index is 4

```

public int Binarysearch(int[] nums, int target) {
    int left = 0;
    int right = nums.length - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (nums[mid] == target) {
            return mid;
        } else if (nums[mid] < target) {
            left = mid + 1;
        } else {
            right = 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 $i \leq j$, `nums[i] ≤ nums[j]`. An array `nums` is monotone decreasing if for all $i \leq j$, `nums[i] ≥ nums[j]`. Given an integer array `nums`, return true if the given array is monotonic, or false otherwise. **Example 1:** Input: `nums = [1,2,2,3]` Output: true

```

class Solution {
    public boolean isMonotonic(int[] nums) {
        boolean inc = true;
        boolean dec = true;
        for (int i = 0; i < nums.length - 1; ++i) {
            if (nums[i] > nums[i+1])
                inc = false;
            if (nums[i] < nums[i+1])
                dec = false;
        }

        return inc || dec;
    }
}

```

Question 8 You are given an integer array `nums` and an integer `k`. In one operation, you can choose any index i where $0 \leq i < \text{nums.length}$ 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. **Example 1:** Input: `nums = [1]`, `k = 0` Output: 0

Explanation: The score is $\max(\text{nums}) - \min(\text{nums}) = 1 - 1 = 0$.

```

class Solution {
    public int smallestRangeI(int[] A, int K) {
        int min = A[0], max = A[0];

```

```
for (int x: A) {  
    min = Math.min(min, x);  
    max = Math.max(max, x);  
}  
  
System.out.println(max-min- 2 * K);  
return Math.max(0, max - min - 2 * K);  
}  
}
```