**ARRAY ASSIGNMENT QUESTIONS**

**PROBLEM 1:**

class Solution {

public List<Boolean> kidsWithCandies(int[] candies, int extraCandies) {

int maxCandy = 0 ;

for(int candy: candies) {

if(candy > maxCandy) {

maxCandy = candy;

}

}

List<Boolean> result = new ArrayList<>();

for(int currentCandy : candies) {

if(currentCandy + extraCandies >= maxCandy) {

result.add(true);

} else {

result.add(false);

}

}

return result;

}

}

**PROBLEM 2:**

class Solution {

public int maxArea(int[] height) {

int i = 0;

int j = height.length-1;

int maxAmt = 0;

int iH;

int jH;

int cal;

while(j>i)

{

iH = height[i];

jH = height[j];

if(iH>jH)

{

cal = jH\*(j-i);

j--;

}

else

{

cal = iH\*(j-i);

i++;

}

if(cal>maxAmt)

maxAmt = cal;

}

return maxAmt;

}

}

**PROBLEM 3:**

class Solution {

public int threeSumClosest(int[] nums, int target) {

int min = Integer.MAX\_VALUE;

int result = 0;

Arrays.sort(nums);

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

{

int j = i + 1;

int k = nums.length - 1;

while (j < k)

{

int sum = nums[i] + nums[j] + nums[k];

int diff = Math.abs(sum - target);

if(diff == 0)

return sum;

if (diff < min)

{

min = diff;

result = sum;

}

if (sum <= target)

{

j++;

}

else

{

k--;

}

}

}

return result;

}

}

PROBLEM 7:

class Solution {

public int maxSubArray(int[] nums) {

int max = Integer.MIN\_VALUE;

int meh = 0;

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

meh = meh + nums[i];

if (meh > max) {

max = meh;

}

if (meh < 0) {

meh = 0;

}

}

return max;

}

}

PROBLEM 8:

class Solution {

public int[][] merge(int[][] intervals) {

List<int[]> ans = new ArrayList<>();

Arrays.sort(intervals, (a, b) -> a[0] - b[0]);

if (intervals.length == 0) {

return ans.toArray(new int[0][]);

}

int[] temp = intervals[0];

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

if (intervals[i][0] <= temp[1]) {

temp[1] = Math.max(temp[1], intervals[i][1]);

} else {

ans.add(temp);

temp = intervals[i];

}

}

ans.add(temp);

return ans.toArray(new int[0][]);

}

}

**PROBLEM 9:**

class Solution {

public void sortColors(int[] nums) {

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

for (int j = 0; j <= i - 1; j++) {

if (nums[j] > nums[j + 1]) {

int temp = nums[j + 1];

nums[j + 1] = nums[j];

nums[j] = temp;

}

}

}

}

}

**PROBLEM 10:**

class Solution {

public int maxProfit(int[] prices) {

int profit = 0;

int minPrice = Integer.MAX\_VALUE;

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

minPrice = Math.min(minPrice, prices[i]);

if (prices[i] - minPrice > 0) {

profit += prices[i] - minPrice;

minPrice = prices[i];

}

}

return profit;

}

}

**PROBLEM 11:**

class Solution {

public int findPeakElement(int[] nums) {

int n = nums.length;

if(n == 1 || nums[0] > nums[1]) return 0;

if(nums[n - 1] > nums[n - 2]) return n - 1;

int l = 0;

int r = n - 1;

while(l <= r)

{

int m = (l + r)/2;

int minus = m == 0 ? Integer.MIN\_VALUE : nums[m-1];

int plusUltra = m == n - 1 ? Integer.MIN\_VALUE : nums[m+1];

if(minus < nums[m] && plusUltra < nums[m]) return m;

if(minus < nums[m]) l = m + 1;

else r = m - 1;

}

return l;

}

}

**PROBLEM 12:**

class Solution {

public double findMedianSortedArrays(int[] nums1, int[] nums2) {

int n1 = nums1.length;

int n2 = nums2.length;

int n = n1 + n2;

int[] new\_arr = new int[n];

int i=0, j=0, k=0;

while (i<=n1 && j<=n2) {

if (i == n1) {

while(j<n2) new\_arr[k++] = nums2[j++];

break;

} else if (j == n2) {

while (i<n1) new\_arr[k++] = nums1[i++];

break;

}

if (nums1[i] < nums2[j]) {

new\_arr[k++] = nums1[i++];

} else {

new\_arr[k++] = nums2[j++];

}

}

if (n%2==0) return (float)(new\_arr[n/2-1] + new\_arr[n/2])/2;

else return new\_arr[n/2];

}

}

**PROBLEM 13:**

Public class ContainerWithMostWater {

public int maxArea(int[] height) {

int maximumArea = Integer.MIN\_VALUE;

int left = 0;

int right = height.length - 1;

while (left < right) {

int shorterLine = Math.min(height[left], height[right]);

maximumArea = Math.max(maximumArea, shorterLine \* (right - left));

if (height[left] < height[right]) {

left++;

} else {

right--;

}

}

return maximumArea;

}

}

**PROBLEM 14:**

class Solution {

public int findMaxConsecutiveOnes(int[] nums) {

int maxOne=0;

int ch=0;

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

{

if(nums[i]==1)

{

ch++;

maxOne=Math.max(maxOne,ch);

}

else if(nums[i]==0)

{

ch=0;

}

}

return maxOne;

}

}

**PROBLEM 15:**

class Solution {

public int findKthLargest(int[] nums, int k) {

Arrays.sort(nums);

int fromLast = nums.length - k;

return nums[fromLast];

}

}