{1} Maximum Product of Three Numbers

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

Example 2:

Input: nums = [1,2,3,4]

Output: 24

Example 3:

Input: nums = [-1,-2,-3]

Output: -6

class Solution {

public int maximumProduct(int[] nums) {

int n=nums.length;

Arrays.sort(nums);

return Math.max(nums[n-1]\*nums[n-2]\*nums[n-3],nums[n-1]\*nums[1]\*nums[0]);

}

}

{2}Self Dividing Numbers

A self-dividing number is a number that is divisible by every digit it contains.

For example, 128 is a self-dividing number because 128 % 1 == 0, 128 % 2 == 0, and 128 % 8 == 0.

A self-dividing number is not allowed to contain the digit zero.

Given two integers left and right, return a list of all the self-dividing numbers in the range [left, right].

Example 1:

Input: left = 1, right = 22

Output: [1,2,3,4,5,6,7,8,9,11,12,15,22]

Example 2:

Input: left = 47, right = 85

Output: [48,55,66,77]

class Solution {

public List<Integer> selfDividingNumbers(int left, int right) {

List<Integer>al=new ArrayList<>();

for(int i=left;i<=right;i++)

{

if(selfDividing(i)){

al.add(i);

}

}

return al;

}

public boolean selfDividing(int n)

{

int temp=n;

while(temp>0){

int ld=temp%10;

if(ld==0){

return false;

}

if(n%ld!=0)

{

return false;

}

temp=temp/10;

}

return true;

}

}

{3}Prime Number of Set Bits in Binary Representation

Given two integers left and right, return the count of numbers in the inclusive range [left, right] having a prime number of set bits in their binary representation.

Recall that the number of set bits an integer has is the number of 1's present when written in binary.

For example, 21 written in binary is 10101, which has 3 set bits.

Example 1:

Input: left = 6, right = 10

Output: 4

Explanation:

6 -> 110 (2 set bits, 2 is prime)

7 -> 111 (3 set bits, 3 is prime)

8 -> 1000 (1 set bit, 1 is not prime)

9 -> 1001 (2 set bits, 2 is prime)

10 -> 1010 (2 set bits, 2 is prime)

4 numbers have a prime number of set bits.

Example 2:

Input: left = 10, right = 15

Output: 5

Explanation:

10 -> 1010 (2 set bits, 2 is prime)

11 -> 1011 (3 set bits, 3 is prime)

12 -> 1100 (2 set bits, 2 is prime)

13 -> 1101 (3 set bits, 3 is prime)

14 -> 1110 (3 set bits, 3 is prime)

15 -> 1111 (4 set bits, 4 is not prime)

5 numbers have a prime number of set bits.

class Solution {

public int countPrimeSetBits(int left, int right) {

int countPrime = 0;

for (int i = left; i <= right; i++) {

if (isPrime(Integer.bitCount(i))) countPrime++;

}

return countPrime;

}

public boolean isPrime(int bit) {

if (bit == 2) return true;

for (int i = 2; i < bit; i++) {

if (bit % i == 0) return false;

}

return bit < 2 ? false : true;

}

}

{4} Largest Triangle Area

Given an array of points on the X-Y plane points where points[i] = [xi, yi], return the area of the largest triangle that can be formed by any three different points. Answers within 10-5 of the actual answer will be accepted.

Example 1:

Input: points = [[0,0],[0,1],[1,0],[0,2],[2,0]]

Output: 2.00000

Explanation: The five points are shown in the above figure. The red triangle is the largest.

Example 2:

Input: points = [[1,0],[0,0],[0,1]]

Output: 0.50000

class Solution {

public double largestTriangleArea(int[][] points) {

double result = 0;

for (int x = 0; x < points.length; x++) {

for (int y = x + 1; y < points.length; y++) {

for (int z = y + 1; z < points.length; z++) {

double area = Math.abs((points[x][0] \* points[y][1] +

points[y][0] \* points[z][1] +

points[z][0] \* points[x][1]) -

(points[x][1] \* points[y][0] +

points[y][1] \* points[z][0] +

points[z][1] \* points[x][0])) / (double) 2;

result = Math.max(result, area);

}

}

}

return result;

}

}

{5}Rectangle Overlap

An axis-aligned rectangle is represented as a list [x1, y1, x2, y2], where (x1, y1) is the coordinate of its bottom-left corner, and (x2, y2) is the coordinate of its top-right corner. Its top and bottom edges are parallel to the X-axis, and its left and right edges are parallel to the Y-axis.

Two rectangles overlap if the area of their intersection is positive. To be clear, two rectangles that only touch at the corner or edges do not overlap.

Given two axis-aligned rectangles rec1 and rec2, return true if they overlap, otherwise return false.

Example 1:

Input: rec1 = [0,0,2,2], rec2 = [1,1,3,3]

Output: true

Example 2:

Input: rec1 = [0,0,1,1], rec2 = [1,0,2,1]

Output: false

Example 3:

Input: rec1 = [0,0,1,1], rec2 = [2,2,3,3]

Output: false

class Solution {

public boolean isRectangleOverlap(int[] rec1, int[] rec2) {

return(rec1[0]<rec2[2]&&rec2[0]<rec1[2]&&rec1[1]<rec2[3]&&rec2[1]<rec1[3]);

}

}

{6}Projection Area of 3D Shapes

You are given an n x n grid where we place some 1 x 1 x 1 cubes that are axis-aligned with the x, y, and z axes.

Each value v = grid[i][j] represents a tower of v cubes placed on top of the cell (i, j).

We view the projection of these cubes onto the xy, yz, and zx planes.

A projection is like a shadow, that maps our 3-dimensional figure to a 2-dimensional plane. We are viewing the "shadow" when looking at the cubes from the top, the front, and the side.

Return the total area of all three projections.

Example 1:

Input: grid = [[1,2],[3,4]]

Output: 17

Explanation: Here are the three projections ("shadows") of the shape made with each axis-aligned plane.

Example 2:

Input: grid = [[2]]

Output: 5

Example 3:

Input: grid = [[1,0],[0,2]]

Output: 8

class Solution {

public int projectionArea(int[][] grid) {

int n=grid.length;

int ans=0;

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

{

int best\_row=0;

int best\_col=0;

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

{

if(grid[i][j]>0)

{

ans++;

}

best\_row=Math.max(best\_row,grid[i][j]);

best\_col=Math.max(best\_col,grid[j][i]);

}

ans=ans+best\_row+best\_col;

}

return ans;

}

}

{7}Surface Area of 3D Shapes

You are given an n x n grid where you have placed some 1 x 1 x 1 cubes. Each value v = grid[i][j] represents a tower of v cubes placed on top of cell (i, j).

After placing these cubes, you have decided to glue any directly adjacent cubes to each other, forming several irregular 3D shapes.

Return the total surface area of the resulting shapes.

Note: The bottom face of each shape counts toward its surface area.

Example 1:

Input: grid = [[1,2],[3,4]]

Output: 34

Example 2:

Input: grid = [[1,1,1],[1,0,1],[1,1,1]]

Output: 32

Example 3:

Input: grid = [[2,2,2],[2,1,2],[2,2,2]]

Output: 46

class Solution {

public int surfaceArea(int[][] grid) {

int sum = 0;

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

for (int j = 0; j < grid[0].length; j++) {

int h = grid[i][j];

int fullS = h > 0 ? h \* 4 + 2 : 0;

//check adjacent and substract touching surface

//check left

if (cellExists(i, j - 1, grid)) {

if (h <= grid[i][j - 1]) {

fullS -= h;

} else {

fullS -= grid[i][j - 1];

}

}

//check up

if (cellExists(i - 1, j, grid)) {

if (h <= grid[i - 1][j]) {

fullS -= h;

} else {

fullS -= grid[i - 1][j];

}

}

//check right

if (cellExists(i, j + 1, grid)) {

if (h <= grid[i][j + 1]) {

fullS -= h;

} else {

fullS -= grid[i][j + 1];

}

}

//check down

if (cellExists(i + 1, j, grid)) {

if (h <= grid[i + 1][j]) {

fullS -= h;

} else {

fullS -= grid[i + 1][j];

}

}

sum += fullS;

}

}

return sum;

}

private static boolean cellExists(int row, int col, int[][] array) {

return (row <= array.length - 1 && row >= 0) && (col <= array[0].length - 1 && col >= 0);

}

}

{8} Smallest Range I

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.

Example 1:

Input: nums = [1], k = 0

Output: 0

Explanation: The score is max(nums) - min(nums) = 1 - 1 = 0.

Example 2:

Input: nums = [0,10], k = 2

Output: 6

Explanation: Change nums to be [2, 8]. The score is max(nums) - min(nums) = 8 - 2 = 6.

Example 3:

Input: nums = [1,3,6], k = 3

Output: 0

Explanation: Change nums to be [4, 4, 4]. The score is max(nums) - min(nums) = 4 - 4 = 0.

class Solution {

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

int mi=Integer.MAX\_VALUE,ma=Integer.MIN\_VALUE;

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

mi=Math.min(mi,nums[i]);

ma=Math.max(ma,nums[i]);

}

return Math.max(0,ma-mi-2\*k);

}

}

{9}X of a Kind in a Deck of Cards

You are given an integer array deck where deck[i] represents the number written on the ith card.

Partition the cards into one or more groups such that:

Each group has exactly x cards where x > 1, and

All the cards in one group have the same integer written on them.

Return true if such partition is possible, or false otherwise.

Example 1:

Input: deck = [1,2,3,4,4,3,2,1]

Output: true

Explanation: Possible partition [1,1],[2,2],[3,3],[4,4].

Example 2:

Input: deck = [1,1,1,2,2,2,3,3]

Output: false

Explanation: No possible partition.

class Solution {

public boolean hasGroupsSizeX(int[] deck) {

int N = deck.length;

int[] count = new int[10000];

for (int c: deck)

count[c]++;

List<Integer> values = new ArrayList();

for (int i = 0; i < 10000; ++i)

if (count[i] > 0)

values.add(count[i]);

search: for (int X = 2; X <= N; ++X)

if (N % X == 0) {

for (int v: values)

if (v % X != 0)

continue search;

return true;

}

return false;

}

}

{10}Largest Perimeter Triangle

Given an integer array nums, return the largest perimeter of a triangle with a non-zero area, formed from three of these lengths. If it is impossible to form any triangle of a non-zero area, return 0.

Example 1:

Input: nums = [2,1,2]

Output: 5

Explanation: You can form a triangle with three side lengths: 1, 2, and 2.

Example 2:

Input: nums = [1,2,1,10]

Output: 0

Explanation:

You cannot use the side lengths 1, 1, and 2 to form a triangle.

You cannot use the side lengths 1, 1, and 10 to form a triangle.

You cannot use the side lengths 1, 2, and 10 to form a triangle.

As we cannot use any three side lengths to form a triangle of non-zero area, we return 0.

class Solution {

public int largestPerimeter(int[] nums) {

int n=nums.length;

Arrays.sort(nums);

for(int k=n-1;k>=2;k--)

{

int c=nums[k];

int b=nums[k-1];

int a=nums[k-2];

if(this.isValid(a,b,c))

{

return a+b+c;

}

}

return 0;

}

private boolean isValid(int a,int b,int c){

return a+b>c;

}

}