**Student:** Yerbolat Pazyl

**Group:** IS-1808K

**ID:** 26244

**Laboratory work #2. Time Complexity. Sorting.**

Solve this problems using sorting algorithms and define their time, space complexity.

Deadline: 15th September 2020 Week 3

**The Problem – 1.**

<https://leetcode.com/problems/increasing-decreasing-string/>

**Description**

**1370. Increasing Decreasing String**

Given a string s. You should re-order the string using the following algorithm:

1. Pick the **smallest** character from s and **append** it to the result.
2. Pick the **smallest** character from s which is greater than the last appended character to the result and **append** it.
3. Repeat step 2 until you cannot pick more characters.
4. Pick the **largest** character from s and **append** it to the result.
5. Pick the **largest** character from s which is smaller than the last appended character to the result and **append** it.
6. Repeat step 5 until you cannot pick more characters.
7. Repeat the steps from 1 to 6 until you pick all characters from s.

In each step, If the smallest or the largest character appears more than once you can choose any occurrence and append it to the result.

Return *the result string* after sorting s with this algorithm.

**Example 1:**

**Input:** s = "aaaabbbbcccc"

**Output:** "abccbaabccba"

**Explanation:** After steps 1, 2 and 3 of the first iteration, result = "abc"

After steps 4, 5 and 6 of the first iteration, result = "abccba"

First iteration is done. Now s = "aabbcc" and we go back to step 1

After steps 1, 2 and 3 of the second iteration, result = "abccbaabc"

After steps 4, 5 and 6 of the second iteration, result = "abccbaabccba"

**Solution**

**Time complexity :** O(n^3)

Sorting an array of characters. Open while loop with condition (s.size() > 0).

1. Add first element in result string and erase this element in s string s.erace(0, 1).
2. Opening while(true), in this loop opening also for loop for find the smallest character from s which is greater than the last appended character to the result. Use plag that if the flag is raised continue if not raised then we have reached the end of s.
3. Pick the largest character from s and append it to the result and erase in s.
4. The situation with (part-2) the smallest symbol continues here but now we add the largest character from s which is smaller than the last appended character to the result and append it. Also there is flag.
5. The largest character from s which is smaller than the last appended character to the result and append it.

**C++ Code**

class Solution {

public:

string sortString(string s) {

string result = "";

sort(s.begin(), s.end());

while(s.size() > 0) {

if(s.size() > 0) {

result += s[0];

s.erase(0, 1);

}

while(true){

bool flag = false;

for(int i =0; i < s.size(); ++i) {

if (result.back() < s[i]) {

result += s[i];

flag = true;

s.erase(i, 1);

break;

}

}

if(!flag)

break;

}

if (s.size() > 0) {

result += s[s.size() - 1];

s.erase(s.size() - 1, 1);

}

while (true) {

bool flag = false;

for(int i = int(s.size()) - 1; i >= 0; --i) {

if (result.back() > s[i]) {

result += s[i];

flag = true;

s.erase(i, 1);

break;

}

}

if(!flag)

break;

}

}

return result;

}

};

**The Problem – 2.**

<https://leetcode.com/problems/average-salary-excluding-the-minimum-and-maximum-salary/>

**Description**

**1491. Average Salary Excluding the Minimum and Maximum Salary**

Given an array of **unique** integers salary where salary[i] is the salary of the employee i.

Return the average salary of employees excluding the minimum and maximum salary.

**Example 1:**

**Input:** salary = [4000,3000,1000,2000]

**Output:** 2500.00000

**Explanation:** Minimum salary and maximum salary are 1000 and 4000 respectively.

Average salary excluding minimum and maximum salary is (2000+3000)/2= 2500

**Solution**

**Time complexity :** O(n)

We must first sort the salary array. After sorting the array the minimum will be at the beginning of the array and the maximum at the end of the array. We need to run through the loop from index 1 to (salary.length – 1), each time add sum salary[i]. When loop stop that return sum / salary.length – 2;

**Java Code**

class Solution {

public double average(int[] salary) {

double sum = 0;

Arrays.sort(salary);

for(int i = 1; i < salary.length - 1; i++)

sum += salary[i];

return (sum / (salary.length - 2));

}

}

**The Problem – 3.**

<https://leetcode.com/problems/relative-sort-array/>

**Description**

**1122. Relative Sort Array**

Given two arrays arr1 and arr2, the elements of arr2 are distinct, and all elements in arr2 are also in arr1.

Sort the elements of arr1 such that the relative ordering of items in arr1 are the same as in arr2.  Elements that don't appear in arr2 should be placed at the end of arr1 in **ascending** order.

**Example 1:**

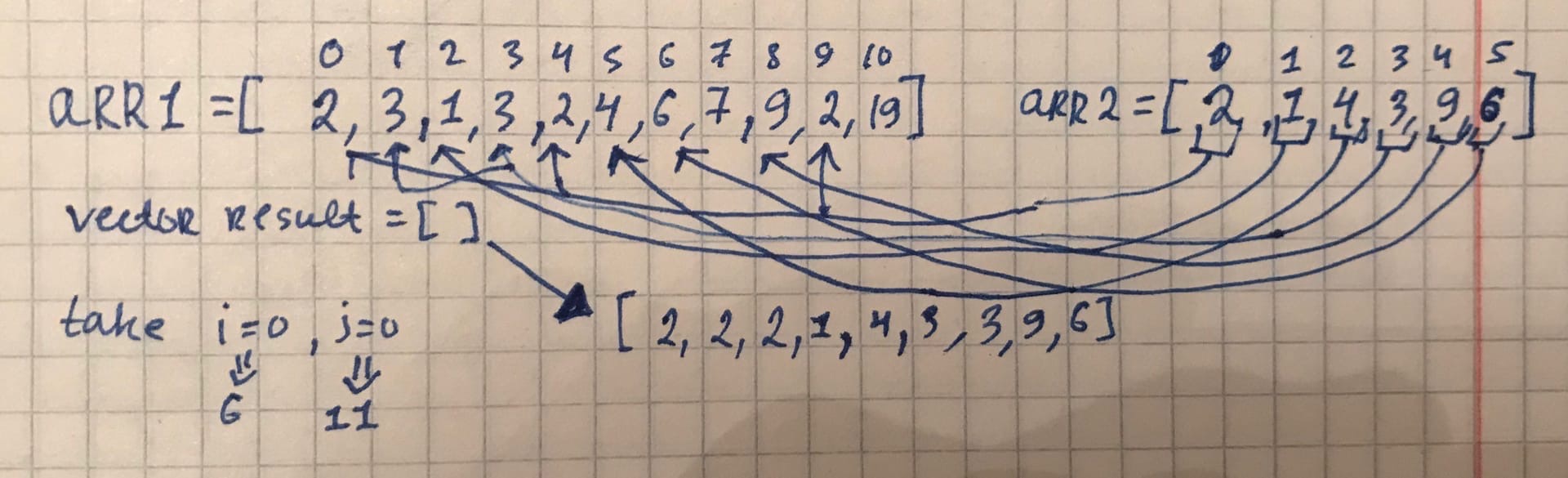
**Input:** arr1 = [2,3,1,3,2,4,6,7,9,2,19], arr2 = [2,1,4,3,9,6]

**Output:** [2,2,2,1,4,3,3,9,6,7,19]

**Solution**

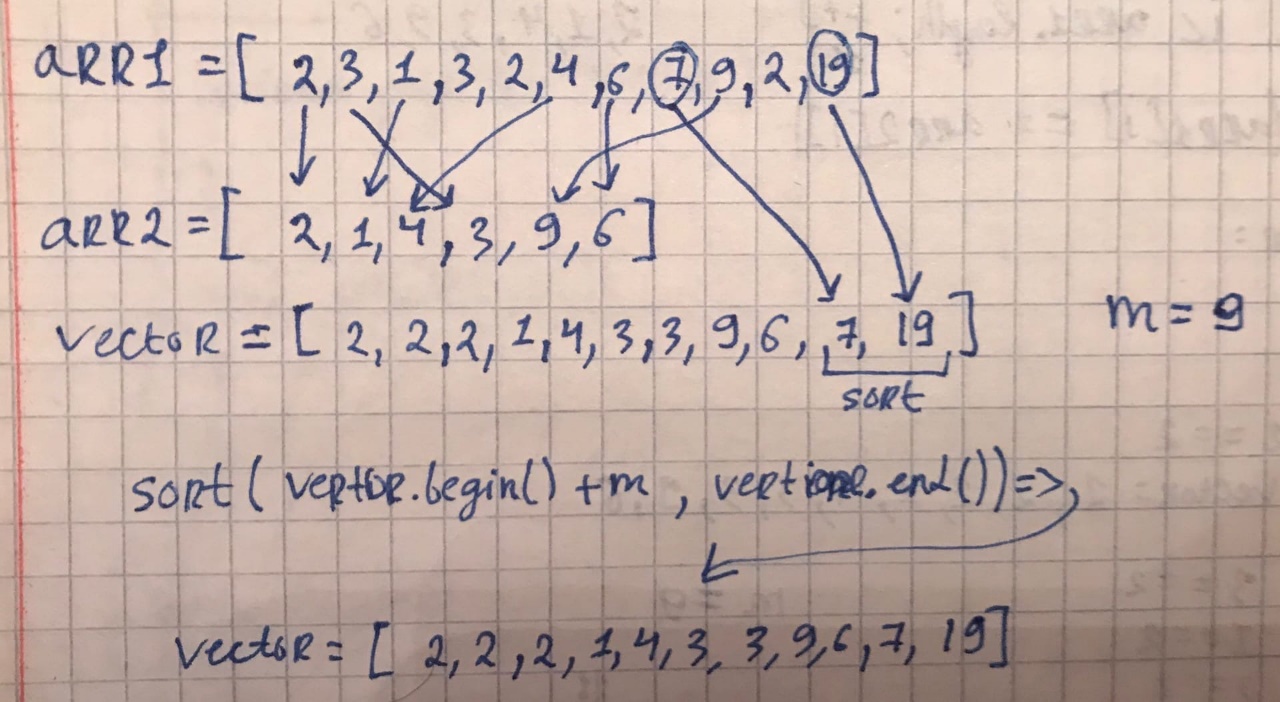
**Time complexity :** O(n \* m)

Сreating a vector “result”. From the array arr1, we take all the elements that are in arr2 and store them in order arr2 in the result vector.



After that, we store the size of the vector in the variable m. Now we do the opposite: we take elements from the array arr1 and compare them with elements from the array arr2.

To know if such an element has been encountered, we will use the flag. initially, the flag is 0 and if arr1[i] = = arr2[j], the flag becomes 1 and this means that this element has already been encountered. Then you need to sort the result vector with the index m to the end.



**C++ Code**

class Solution {

public:

vector<int> relativeSortArray(vector<int>& arr1, vector<int>& arr2) {

vector <int> result;

for (int i = 0; i < arr2.size(); i++) {

for (int j = 0; j < arr1.size(); ++j) {

if (arr1[j] == arr2[i])

result.push\_back(arr2[i]);

}

}

int m = result.size();

for(int i = 0; i < arr1.size(); i++) {

bool flag = false;

for(int j = 0; j < arr2.size(); j++)

if(arr1[i] == arr2[j]) {

flag = true;

break;

}

if (!flag)

result.push\_back(arr1[i]);

}

sort(result.begin() + m, result.end());

return result;

}

};

**The Problem – 4.**

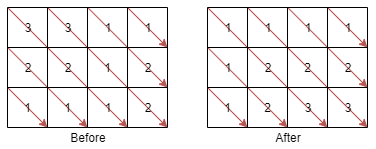
<https://leetcode.com/problems/sort-the-matrix-diagonally/>

**Description**

**1329. Sort the Matrix Diagonally**

Given a m \* n matrix mat of integers, sort it diagonally in ascending order from the top-left to the bottom-right then return the sorted array.

**Example 1:**



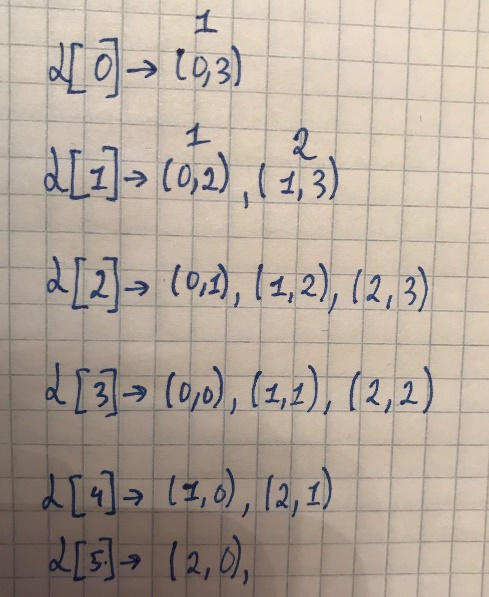
**Input:** mat = [[3,3,1,1],[2,2,1,2],[1,1,1,2]]

**Output:** [[1,1,1,1],[1,2,2,2],[1,2,3,3]]

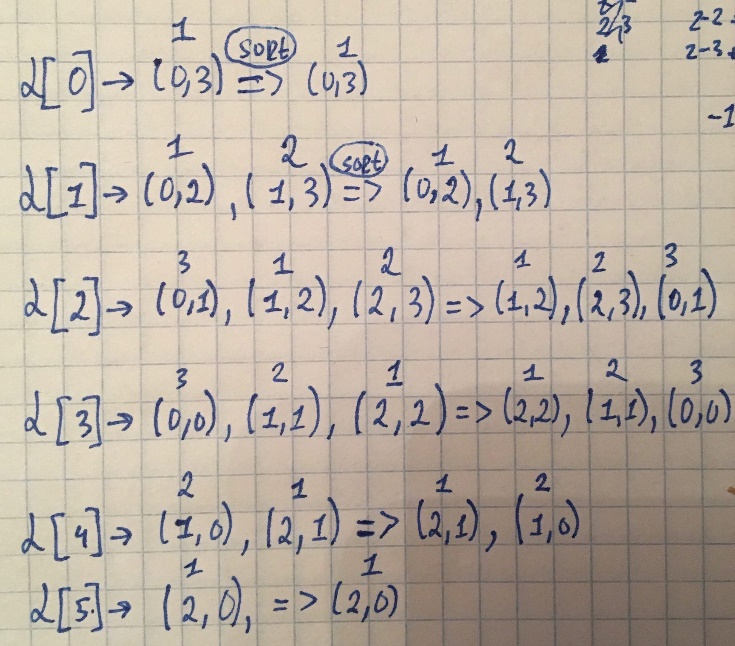
**Solution**

**Time complexity :** O(n \* m)

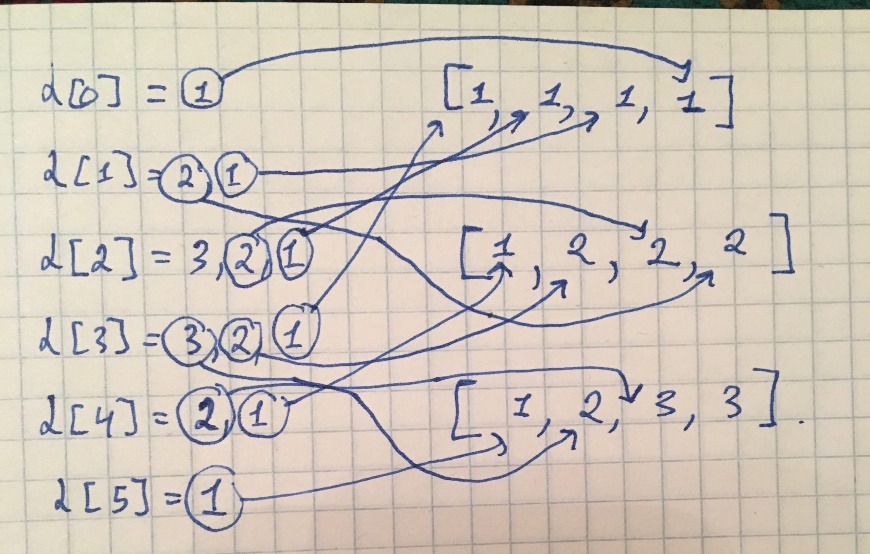
Сreating an array of vectors the size will be with the number of diagonals. Number of diagonals is equal (mat.size() + mat[0].size() – 1). Elements that are in the same diagonal have the same (i - j). To get rid of negative (i - j), add “m - 1”. Adding elements to your array d[i – j + (m - 1)] = mat[i][j].



Sort each array d[] and make a reserve.



Now we rearrange the elements back. This process happens like this: from the array d[i - j + delta], we take the farthest one and put mat[i][j] in its place.



**C++ Code**

class Solution {

public:

vector<vector<int>> diagonalSort(vector<vector<int>>& mat) {

int n = mat.size();

int m = mat[0].size();

vector < int > d[n + m - 1];

int delta = m - 1;

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

for(int j = 0; j < m; ++j) {

int x = i - j + delta;

d[x].push\_back(mat[i][j]);

}

}

for(int k = n - 1; k >= 1 - m; --k) {

sort(d[k + delta].begin(), d[k + delta].end());

reverse(d[k + delta].begin(), d[k + delta].end());

}

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

for (int j = 0; j < m; ++j) {

int x = i - j + delta;

mat[i][j] = d[x].back();

d[x].pop\_back();

}

}

return mat;

}

};

**The Problem – 5.**

<https://leetcode.com/problems/maximum-number-of-coins-you-can-get/>

**Description**

**1561. Maximum Number of Coins You Can Get**

There are 3n piles of coins of varying size, you and your friends will take piles of coins as follows:

* In each step, you will choose **any**3 piles of coins (not necessarily consecutive).
* Of your choice, Alice will pick the pile with the maximum number of coins.
* You will pick the next pile with maximum number of coins.
* Your friend Bob will pick the last pile.
* Repeat until there are no more piles of coins.

Given an array of integers piles where piles[i] is the number of coins in the ith pile.

Return the maximum number of coins which you can have.

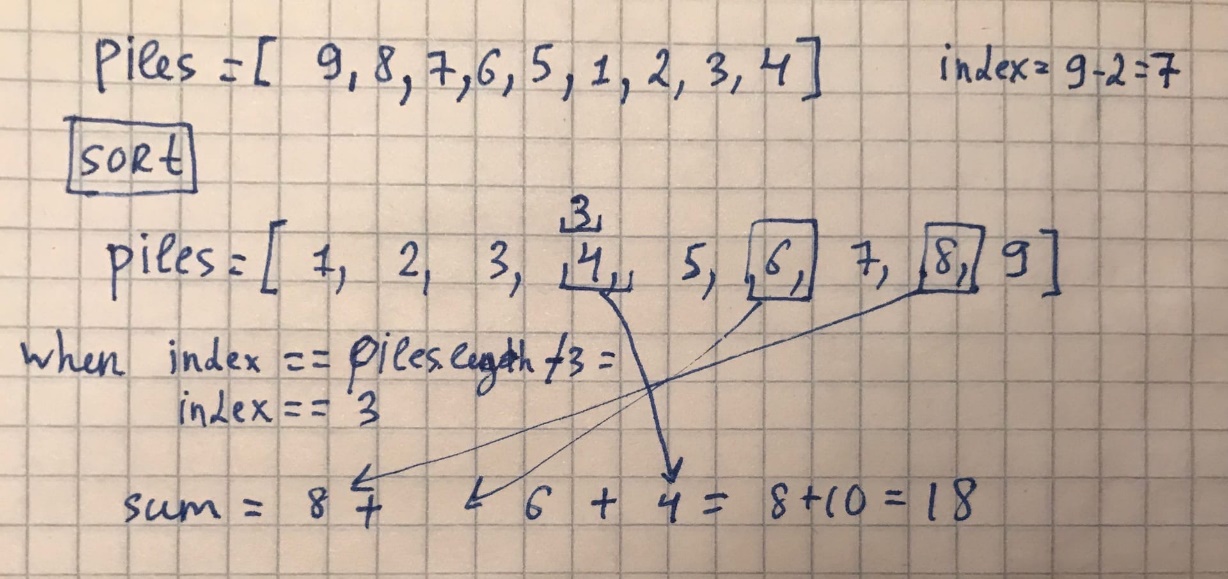
**Input:** piles = [9,8,7,6,5,1,2,3,4]

**Output:** 18

**Solution**

**Time complexity :** O(n / 3)

The first thing to do is sort the array piles. Create variable index = piles.length – 2. To get the maximum number of coins, you need to go from right to left until the index is equal to files.length / 3. When index == piles.length / 3 loop stop. Each time index -= 2.



**Java Code**

class Solution {

public int maxCoins(int[] piles) {

Arrays.sort(piles);

int index = piles.length - 2;

int sum = 0;

while(true) {

sum += piles[index];

if(piles.length / 3 == index)

return sum;

index -= 2;

}

}

}

**The Problem – 6.**

<https://leetcode.com/problems/sort-integers-by-the-power-value/>

**Description**

**1387. Sort Integers by The Power Value**

The power of an integer x is defined as the number of steps needed to transform x into 1 using the following steps:

* if x is even then x = x / 2
* if x is odd then x = 3 \* x + 1

For example, the power of x = 3 is 7 because 3 needs 7 steps to become 1 (3 --> 10 --> 5 --> 16 --> 8 --> 4 --> 2 --> 1).

Given three integers lo, hi and k. The task is to sort all integers in the interval [lo, hi] by the power value in **ascending order**, if two or more integers have **the same** power value sort them by **ascending order**.

Return the k-th integer in the range [lo, hi] sorted by the power value.

Notice that for any integer x (lo <= x <= hi) it is **guaranteed** that x will transform into 1 using these steps and that the power of x is will **fit** in 32 bit signed integer.

**Example 1:**

**Input:** lo = 12, hi = 15, k = 2

**Output:** 13

**Explanation:** The power of 12 is 9 (12 --> 6 --> 3 --> 10 --> 5 --> 16 --> 8 --> 4 --> 2 --> 1)

The power of 13 is 9

The power of 14 is 17

The power of 15 is 17

The interval sorted by the power value [12,13,14,15]. For k = 2 answer is the second element which is 13.

Notice that 12 and 13 have the same power value and we sorted them in ascending order. Same for 14 and 15.

**Solution**

**Time complexity :** O( (hi – lo)lghi )

Used structure which has value and powerValue variables. Constructor determines power value each number. Create custom cmp fuction for sort. If they are the powerValue same then compare by value. Then we take the element with the index (K - 1).

**Java Code**

class Solution {

public int getKth(int lo, int hi, int k) {

Power [] powers = new Power [hi + 1 - lo];

for(int i = lo, j = 0; i <= hi; ++i, ++j)

powers[j] = new Power(i);

Arrays.sort(powers, new PowerComparator());

return powers[k - 1].value;

}

class Power {

int value;

long power;

Power(){}

Power(int a) {

this.value = a;

power = 0;

while(a != 1)

{

if(a % 2 == 0)

a = a / 2;

else

a = (3 \* a) + 1;

++power;

}

}

}

class PowerComparator implements Comparator<Power>{

public int compare(Power s1, Power s2) {

if (s1.power > s2.power)

return 1;

else if (s1.power < s2.power)

return -1;

return 0;

}

}

}

**The Problem – 7.**

<https://leetcode.com/problems/largest-perimeter-triangle/>

**Description**

**976. Largest Perimeter Triangle**

Given an array A of positive lengths, return the largest perimeter of a triangle with **non-zero area**, formed from 3 of these lengths.

If it is impossible to form any triangle of non-zero area, return 0.

**Example 3:**

**Input:** [3,2,3,4]

**Output:** 10

**Example 4:**

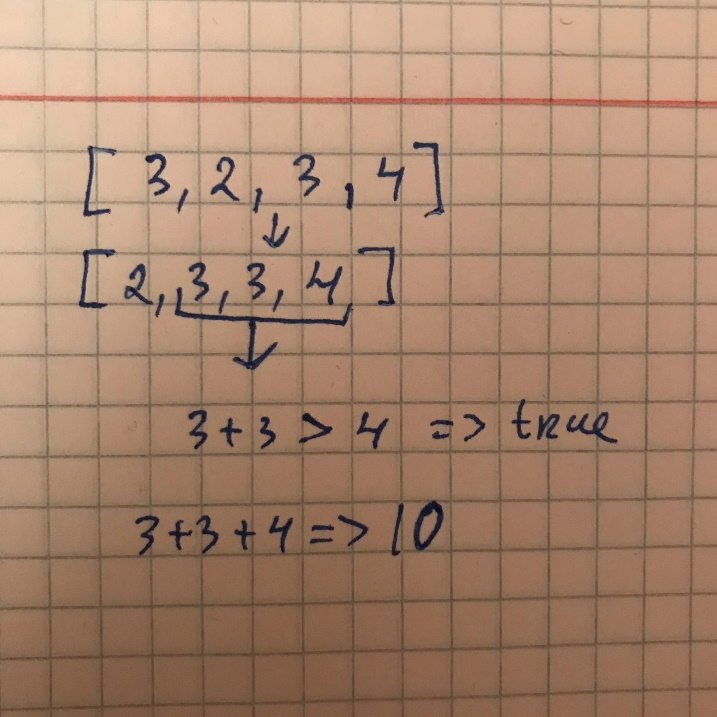
**Input:** [3,6,2,3]

**Output:** 8

**Solution**

**Time complexity :** O(n \* lgn)

Sorting array of A. Open the loop and take the last three elements of the array and check that the sum of the two sides of the triangle must always be greater than the third side. If the conditions are violated we make an decrement i-- take three more elements but not with the index i + 1. If is true return A[i] + A[i-1] + A[i- 2].



**Java Code**

class Solution {

public int largestPerimeter(int[] A) {

Arrays.sort(A);

for(int i = A.length - 1; i >= 2; --i){

if(A[i] < A[i - 1] + A[i - 2])

return A[i] + A[i - 1] + A[i - 2];

}

return 0;

}

}

**The Problem – 8.**

<https://leetcode.com/problems/intersection-of-two-arrays/>

**Description**

**349. Intersection of Two Arrays**

Given two arrays, write a function to compute their intersection.

**Example 1:**

**Input:** nums1 = [1,2,2,1], nums2 = [2,2]

**Output:** [2]

**Solution**

**Time complexity :** O(n+m)

Сreating two Hashsets for two arrays nums1 and nums2. HashSet stores only unique elements. Removing duplicate elements from nums1 and nums2.

After that, create an array res with the size set 1. size(). using the shortened for loop, we find the intersection.

The res array may not be fully. Do fully array Arrays.copyOf(res, i).

**Java Code**

class Solution {

public int[] intersection(int[] nums1, int[] nums2) {

HashSet<Integer> set1 = new HashSet<Integer>();

HashSet<Integer> set2 = new HashSet<Integer>();

for(int a : nums1)

set1.add(a);

for(int a : nums2)

set2.add(a);

int [] res = new int[set1.size()];

int i = 0;

for (Integer a : set1)

if (set2.contains(a))

res[i++] = a;

return Arrays.copyOf(res, i);

}

}

**The Problem – 9.**  
<https://leetcode.com/problems/k-closest-points-to-origin/>

**Description**

**973. K Closest Points to Origin**

We have a list of points on the plane.  Find the K closest points to the origin (0, 0).

(Here, the distance between two points on a plane is the Euclidean distance.)

You may return the answer in any order.  The answer is guaranteed to be unique (except for the order that it is in.)

**Example 2:**

**Input:** points = [[3,3],[5,-1],[-2,4]], K = 2

**Output:** [[3,3],[-2,4]]

(The answer [[-2,4],[3,3]] would also be accepted.)

**Solution**

**Time complexity :** O(points.length+K)

Created class Point with variables index (to store a position in points array) and length (to store a distance [0, 0]). Сreate an array myPoints with the size of the points.length and type Point calss. Used for loops insert data from points to myPoins. Create new custom Comparator for sort array by length and not lose position.

And create result 2D array with size of [K][2]. Used for loops index start 0 and end K. for(int i = 0; i < K; i++) {

int j = myPoints[i].index;

result[i][0] = points[j][0];

result[i][1] = points[j][1];

}

Return result array.

**Java Code**

class Solution {

public int[][] kClosest(int[][] points, int K) {

int [][] result = new int [K][2];

Point [] myPoints = new Point [points.length];

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

myPoints[i] = new Point(i, (int)(Math.pow(points[i][0], 2) + Math.pow(points[i][1], 2)));

}

Arrays.sort(myPoints, new PointComparator());

for(int i = 0; i < K; i++) {

int j = myPoints[i].index;

result[i][0] = points[j][0];

result[i][1] = points[j][1];

}

return result;

}

class Point {

int index;

int length;

Point(){}

Point(int i, int l){

index = i;

length = l;

}

}

class PointComparator implements Comparator<Point>{

public int compare(Point s1, Point s2) {

if (s1.length > s2.length)

return 1;

else if (s1.length < s2.length)

return -1;

return 0;

}

}

}

**The Problem – 10.**

<https://leetcode.com/problems/largest-number/>

**Description**

**179. Largest Number**

Given a list of non negative integers, arrange them such that they form the largest number.

**Example 1:**

**Input:** [10,2]

**Output:** "210"

**Example 2:**

**Input:** [3,30,34,5,9]

**Output:** "9534330"

**Solution**

**Time complexity :** O(n \* lgn)

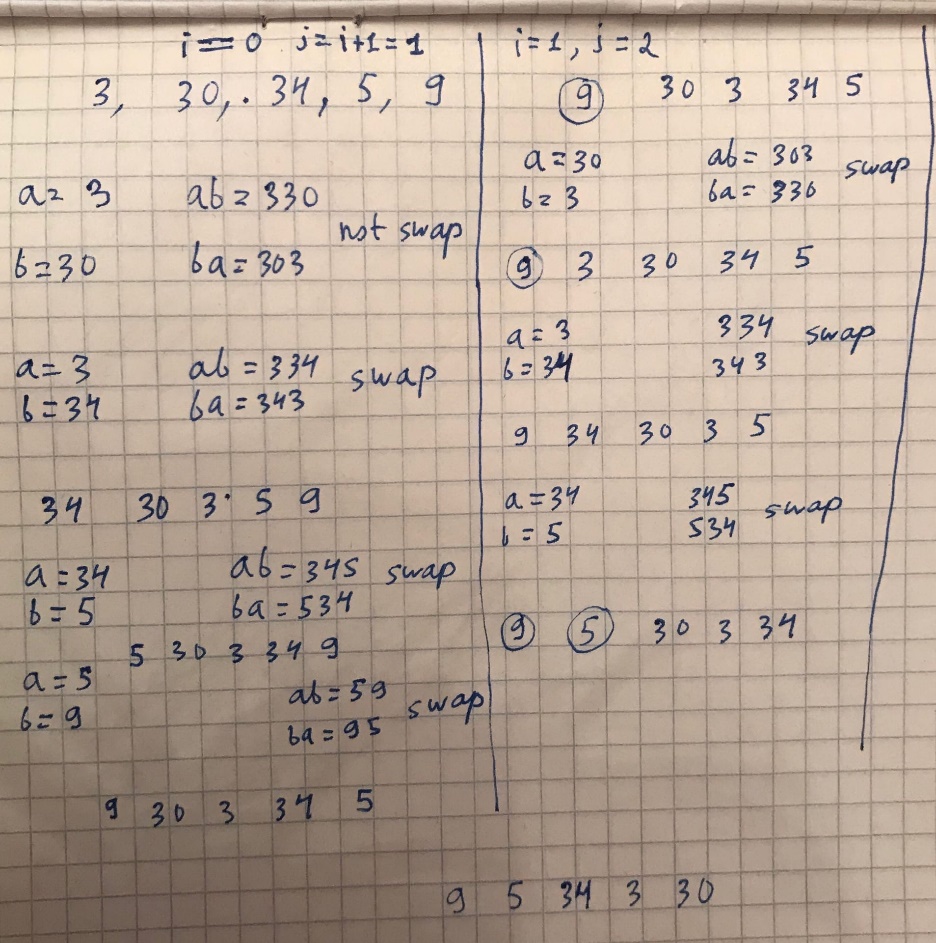
Сreating a double loop. for(i=0; i < n-1; i++) and for(j = i + 1; j < n; j++).

String a = nums[i], String b = nums[j]. If ab < ba, value nums[i] and nums[j] do swap.

Check nums[0] == 0, answer will be “0”.

After open for(int a : nums), each time string result += String.valueOf(a).

String result is answer.



**Java Code**

class Solution {

public String largestNumber(int[] nums) {

int n = nums.length;

String a, b, result = "";

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

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

a = String.valueOf(nums[i]);

b = String.valueOf(nums[j]);

if(Long.parseLong(a + b) < Long.parseLong(b + a)) {

int temp = nums[i];

nums[i] = nums[j];

nums[j] = temp;

}

}

}

if(nums[0] == 0)

return "0";

for(int i : nums) {

result += String.valueOf(i);

}

return result;

}

}