**Laboratory work #2.**

1. **Increasing Decreasing String**

<https://leetcode.com/problems/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.

**Example:**

**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"

**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**;

}

}

}

};

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

<https://leetcode.com/problems/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:**

**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

**Code:**

**class** Solution {

**public**:

**double** average(vector<**int**>& salary) {

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

**double** sum = 0;

**for** (**int** i = 1; i < salary.size() - 1; ++i){

sum += salary[i];

}

**return** sum / (salary.size() - 2);

}

};

1. **Relative Sort Array**

<https://leetcode.com/problems/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 :**

**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]

**Code:**

**class** Solution {

**public**:

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

vector < **int** > res;

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

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

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

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

}

**int** m = res.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)

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

}

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

**return** res;

}

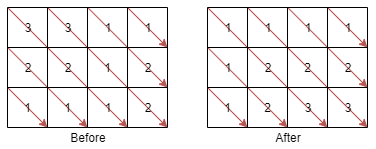
};

1. **Sort the Matrix Diagonally**

<https://leetcode.com/problems/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:**



**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]]

**Code:** class Solution {

public:

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

int m = mat.size();

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

for(int i=m-2; i>=0; i--){

diagsort(mat, i, 0, m, n);

}

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

diagsort(mat, 0, j, m, n);

}

return mat;

}

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

<https://leetcode.com/problems/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.

**Example:**

**Input:** piles = [2,4,1,2,7,8]

**Output:** 9

**Explanation:** Choose the triplet (2, 7, 8), Alice Pick the pile with 8 coins, you the pile with **7** coins and Bob the last one.

Choose the triplet (1, 2, 4), Alice Pick the pile with 4 coins, you the pile with **2** coins and Bob the last one.

The maximum number of coins which you can have are: 7 + 2 = 9.

On the other hand if we choose this arrangement (1, **2**, 8), (2, **4**, 7) you only get 2 + 4 = 6 coins which is not optimal.

**Code:**

**class Solution {**

**public:**

**int maxCoins(vector<int>& piles) {**

**sort(piles.begin(), piles.end());**

**int ans = 0;**

**for (int i = piles.size() / 3; i < piles.size(); i += 2)**

**ans += piles[i];**

**return ans;**

**}**

**};**

Time: O(NlogN)

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

<https://leetcode.com/problems/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:**

**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.

**Code:**

**class Solution {**

**public:**

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

**vector < pair <int, int> > v;**

**for (int i = lo; i <= hi; i++){**

**int x = i;**

**int cnt = 0;**

**while (x != 1){**

**if (x % 2 == 0)**

**x /= 2;**

**else**

**x = x \* 3 + 1;**

**cnt++;**

**}**

**v.push\_back(make\_pair(cnt, i));**

**}**

**sort(v.begin(), v.end());**

**return v[k - 1].second;**

**}**

**};**

1. **Largest Perimeter Triangle**

<https://leetcode.com/problems/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:**

**Input:** [2,1,2]

**Output:** 5

**Code:**

**class Solution {**

**public:**

**int largestPerimeter(vector<int>& A) {**

**int mx = 0;**

**sort(A.begin(), A.end());**

**for (int i = 0; i + 2 < A.size(); i++)**

**if (A[i] + A[i + 1] > A[i + 2] && A[i] + A[i + 2] > A[i + 1] && A[i + 2] + A[i + 1] > A[i])**

**mx = max(mx, A[i] + A[i + 1] + A[i + 2]);**

**return mx;**

**}**

**};**

1. **Intersection of Two Arrays**

<https://leetcode.com/problems/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]

**Code:**

**class Solution {**

**public:**

**vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {**

**int mx = 0;**

**for (int i = 0; i < nums1.size(); i++)**

**mx = max(mx, nums1[i]);**

**for (int i = 0; i < nums2.size(); i++)**

**mx = max(mx, nums2[i]);**

**bool was[mx + 1];**

**for (int i = 0; i <= mx; i++)**

**was[i] = false;**

**for (int i = 0; i < nums1.size(); i++)**

**was[nums1[i]] = true;**

**vector <int> ans;**

**for (int i = 0; i < nums2.size(); i++)**

**if (was[nums2[i]]){**

**was[nums2[i]] = false;**

**ans.push\_back(nums2[i]);**

**}**

**return ans;**

**}**

**};**

1. **K Closest Points to Origin**

<https://leetcode.com/problems/k-closest-points-to-origin/>

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

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

**Example:**

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

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

**Explanation:**

The distance between (1, 3) and the origin is sqrt(10).

The distance between (-2, 2) and the origin is sqrt(8).

Since sqrt(8) < sqrt(10), (-2, 2) is closer to the origin.

We only want the closest K = 1 points from the origin, so the answer is just [[-2,2]].

**Code:**

**class Solution {**

**public:**

**vector<vector<int>> kClosest(vector<vector<int>>& points, int K) {**

**vector < pair <int, vector <int> > > v;**

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

**int x = points[i][0];**

**int y = points[i][1];**

**v.push\_back(make\_pair(x \* x + y \* y, points[i]));**

**}**

**sort(v.begin(), v.end());**

**vector <vector <int>> ans;**

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

**ans.push\_back(v[i].second);**

**}**

**return ans;**

**}**

**};**

1. **Largest Number**

<https://leetcode.com/problems/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"

**Code:**

bool cmp(string x, string y){

return (x + y < y + x);

}

class Solution {

public:

string largestNumber(vector<int>& nums) {

string s = "", ans = "";

vector <string> v;

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

int x = nums[i];

if (x == 0){

v.push\_back("0");

continue;

}

s = "";

while (x > 0){

s += (x % 10 + '0');

x /= 10;

}

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

v.push\_back(s);

}

sort(v.begin(), v.end(), cmp);

reverse(v.begin(), v.end());

if (v[0] == "0"){

return "0";

}

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

ans += v[i];

}

return ans;

}

};