410. Split Array Largest Sum

給你一個數組和n，分為n個子數組，每個數組各自求和sum[]，使sum[]最大值最小化

思路：剛開始sum最小是left=Max(nums), 最大是right=Sum(nums),然後取mid=(left+right)/2.

如果可以split，right=mid 記住，因為mid可能已經是答案，不能取mid-1

如果不能split，left=mid+1，取mid可以，但會變慢。

cansplit函數就是掃描整個數組，看能不能分為n個數組而且，每個數組的sum不超過參數要求的sum

public:

int splitArray(vector<int>& nums, int m) {

long long left = 0, right = 0;

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

left = max((int)left, nums[i]);

right += nums[i];

}

while (left < right) {

long long mid = left + (right - left) / 2;

if (can\_split(nums, m, mid)) right = mid;

else left = mid + 1;

}

return left;

}

bool can\_split(vector<int>& nums, int m, int sum) {

int cnt = 1, curSum = 0;

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

curSum += nums[i];

if (curSum > sum) {

curSum = nums[i];

++cnt;

if (cnt > m) return false;

}

}

return true;

}

81. Search in Rotated Sorted ArrayII(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).在當中找到target的數字

思路：二分法 堤防567555這類情況

start， mid， end

如果num[mid]=num[start]=num[end]不知取哪邊，那就令end=end-1;

如果左邊有序。當num[start]<=target<num[mid] 在左邊找。否則在右邊找

如果左邊無序，即是右邊有序，如果num[mid]<target<=num[end]在右邊找。否則在左邊找

class Solution {

public:

bool search(int A[], int n, int target) {

int lo =0, hi = n-1;

int mid = 0;

while(lo<hi){

mid=(lo+hi)/2;

if(A[mid]==target) return true;

if(A[mid]>A[hi]){

if(A[mid]>target && A[lo] <= target) hi = mid;

else lo = mid + 1;

}else if(A[mid] < A[hi]){

if(A[mid]<target && A[hi] >= target) lo = mid + 1;

else hi = mid;

}else{

hi--;

}

}

return A[lo] == target ? true : false;

}

};

367. Valid Perfect Square

long r = x;

while (r\*r > x)

r = (r + x/r) / 2;

return r\*r == x;

29. Divide Two Integers

class Solution {

public:

int divide(int dividend, int divisor) {

if (!divisor || (dividend == INT\_MIN && divisor == -1))

return INT\_MAX;

int sign = ((dividend < 0) ^ (divisor < 0)) ? -1 : 1;

long long dvd = labs(dividend);

long long dvs = labs(divisor);

int res = 0;

while (dvd >= dvs) {

long long temp = dvs, multiple = 1;

while (dvd >= (temp << 1)) {

temp <<= 1;

multiple <<= 1;

}

dvd -= temp;

res += multiple;

}

return sign == 1 ? res : -res;

}

};

167. Two Sum II - Input array is sorted

給出target，在nums[]中找出兩個數加起來是target，返回兩個數的index。nums[]是有序

vector<int> twoSum(vector<int>& numbers, int target) {

int lo=0, hi=numbers.size()-1;

while (numbers[lo]+numbers[hi]!=target){

if (numbers[lo]+numbers[hi]<target){

lo++;

} else {

hi--;

}

}

return vector<int>({lo+1,hi+1});

}

162. Find Peak Element

一個數組，peak element滿足nums[peak]>nums[peak-1] && nums[peak]>nums[peak+1]。如果存在多個peak element返回任意一個即可

思路：binary search

如果nums[mid]>nums[mid+1]，在[start, mid]找

如果nums[mid]<nums[mid+1]，在[mid+1,end]找

int findPeakElement(const vector<int> &num) {

if (num.size() <= 1) return 0;

int mid = 0, l = 0, h = num.size() - 1;

while (l < h) {

mid = (l + h) / 2;

if (num[mid] > num[mid + 1])

h = mid;

else if (num[mid] < num[mid + 1])

l = mid + 1;

}

return l;

}

153. Find Minimum in Rotated Sorted Array

int findMin(vector<int> &num) {

int lo =0, hi = num.size()-1;

while(lo<hi){

int mid=(lo+hi)/2;

if(num[mid]>num[hi]) lo=mid+1;

else hi=mid;

}

return num[lo];

}

50. Pow(x, n)

不簡單。使用分治算法，pow(double x, int m)是返回準確的計算值。

所以使用pow(x,m/2)計算下一級的返回值，如果當前m是偶數直接將返回值平方。

如果是奇數，假如m是正，返回值平方后乘上x，假如是負，返回值平方除以x

public class Solution {

public double pow(double x, int m) {

double temp=x;

if(m==0)

return 1;

temp=pow(x,m/2);

if(m%2==0)

return temp\*temp;

else

{

if(m > 0)

return x\*temp\*temp;

else

return (temp\*temp)/x;

}

}

35. Search Insert Position

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

class Solution {

public:

int searchInsert(vector<int>& nums, int target) {

int low = 0, high = nums.size()-1;

// Invariant: the desired index is between [low, high+1]

while (low <= high) {

int mid = low + (high-low)/2;

if (nums[mid] < target)

low = mid+1;

else

high = mid-1;

}

// (1) At this point, low > high. That is, low >= high+1

// (2) From the invariant, we know that the index is between [low, high+1], so low <= high+1. Follwing from (1), now we know low == high+1.

// (3) Following from (2), the index is between [low, high+1] = [low, low], which means that low is the desired index

// Therefore, we return low as the answer. You can also return high+1 as the result, since low == high+1

return low;

}

};

34. Search for a Range

Given [5, 7, 7, 8, 8, 10] and target value 8,  
return [3, 4]. 如果找不到return {-1，-1}

思路：子函數找出第一個>=target的位置，注意high必須是A.length

使用兩次查找，第一次找target，第二次找target+1位置，然後將這個位置-1

private static int firstGreaterEqual(int[] A, int target) {

int low = 0, high = A.length;

while (low < high) {

int mid = low + ((high - low) >> 1);

//low <= mid < high

if (A[mid] < target) {

low = mid + 1;

} else {

//should not be mid-1 when A[mid]==target.

//could be mid even if A[mid]>target because mid<high.

high = mid;

}

}

return low;

}

public int[] searchRange(int[] A, int target) {

int start = Solution.firstGreaterEqual(A, target);

if (start == A.length || A[start] != target) {

return new int[]{-1, -1};

}

return new int[]{start, Solution.firstGreaterEqual(A, target + 1) - 1};

}

454. 4Sum II

給你4個list ABCD，不是sorted，長度一樣，找出有多少組合滿足，A[i]+B[j]+C[k]+D[l]==0

思路：用hashmap記錄前兩個數組的所有combination sum的出現次數

之後掃描後面兩個數組的和，找出對應在hashmap出現次數，count+=次數

public int fourSumCount(int[] A, int[] B, int[] C, int[] D) {

Map<Integer,Integer> sums = new HashMap<>();

int count = 0;

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

for(int j=0;j<B.length;j++){

int sum = A[i]+B[j];

if(sums.containsKey(sum)) {

sums.put(sum, sums.get(sum)+1);

} else {

sums.put(sum, 1);

}

}

}

for(int k=0; k<A.length;k++) {

for(int z=0;z<C.length;z++){

int sum = -(C[k]+D[z]);

if(sums.containsKey(sum)) {

count+=sums.get(sum);

}

}

}

return count;

}

**Median of Two Sorted Arrays**

思路： 從兩個數組長度可以推知，從大到小排列中查出第K個數就是中位數。

如何找第K大的數？

partA=Min( k/2, A.length) ; partB=K –partA;

if (A[partA - 1] < B[partB - 1]) A的左邊刪去

否則B的左邊刪去

public class Solution {

private int findNth(int A[], int B[], int k){

if (A.length > B.length) {

return findNth(B, A, k);

}

if (A.length == 0) {

return B[k - 1];

}

if (B.length == 0) {

return A[k - 1];

}

if (k == 1) {

return Math.min(A[0], B[0]);

}

int partA = Math.min(k/2, A.length);

int partB = k - partA;

if (A[partA - 1] < B[partB - 1]) {

return findNth(Arrays.copyOfRange(A, partA, A.length), B, k - partA);

} else {

return findNth(A, Arrays.copyOfRange(B, partB, B.length), k - partB);

}

}

public double findMedianSortedArrays(int A[], int B[]) {

int lenSum = A.length + B.length;

if (lenSum % 2 == 1) {

return findNth(A, B, lenSum / 2 + 1);

} else {

return (findNth(A, B, lenSum / 2) + findNth(A, B, lenSum / 2 + 1)) / 2.0;

}

}

}