**Merge k Sorted Lists**

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

**public** **class** Solution {

**public** ListNode mergeKLists(ListNode[] lists) {

ListNode res = **new** ListNode(0);

**if**(lists==**null** || lists.length==0)

**return** **null**;

**int** len = lists.length;

**int** head = 0;

**int** tail = len-1;

res = helper(head, tail, lists);

**return** res;

}

**public** ListNode helper(**int** head, **int** tail, ListNode[] lists){

**if**(head==tail)

**return** lists[head];

**int** mid = head + (tail-head)/2;

ListNode leftList = helper(head, mid, lists);

ListNode rightList = helper(mid+1, tail, lists);

ListNode resList = mergeList(leftList, rightList);

**return** resList;

}

**public** ListNode mergeList(ListNode leftList, ListNode rightList){

ListNode dummy = **new** ListNode(0);

ListNode cur = dummy;

ListNode leftCur = leftList;

ListNode rightCur = rightList;

**while**(leftCur!=**null** && rightCur!=**null**){

**if**(leftCur.val <= rightCur.val){

cur.next = leftCur;

leftCur = leftCur.next;

cur = cur.next;

}

**else**{

cur.next = rightCur;

rightCur = rightCur.next;

cur = cur.next;

}

}

**if**(leftCur==**null**)

cur.next = rightCur;

**if**(rightCur==**null**)

cur.next = leftCur;

**return** dummy.next;

}

}

**Maximal Square**

// for each [i][j], how large the square is depends on its left & up & leftup

// if current elem == 1: [i][j] = min(left, up, leftup)+1

// because we need the left & up elem, we have to process 1col & 1row seperately

**public** **class** Solution {

**public** **int** maximalSquare(**char**[][] matrix) {

**if**(matrix==**null** || matrix.length==0 || matrix[0].length==0)

**return** 0;

**int** row = matrix.length;

**int** col = matrix[0].length;

**int**[][] resMatrix = **new** **int**[row][col];

**int** res = 0;

// process first row

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

resMatrix[0][i] = matrix[0][i] - '0';

res = Math.max(res, resMatrix[0][i]);

}

// process first col

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

resMatrix[i][0] = matrix[i][0] - '0';

res = Math.max(res, resMatrix[i][0]);

}

// process the rest

**for**(**int** i=1; i<row; i++){

**for**(**int** j=1; j<col; j++){

**if**(matrix[i][j]=='1'){ // 忘记判断了

resMatrix[i][j] =

Math.min(resMatrix[i-1][j-1],

Math.min(resMatrix[i-1][j], resMatrix[i][j-1])) + 1;

res = Math.max(res, resMatrix[i][j]);

}

**else**{

resMatrix[i][j] = 0;

}

}

}

**return** res \* res;

}

}

**House Robber**

// i-th house: 1. rob [i-2] & [i]

// 2. rob [i-1], no [i]

// sum[0] = nums[0]

// sum[1] = max(num[0], num[1])

// sum[i] = max(sum[i-1], sum[i-2]+num[i])

**public** **class** Solution {

**public** **int** rob(**int**[] nums) {

**if**(nums==**null** || nums.length==0)

**return** 0;

**int** len = nums.length;

**int** res[] = **new** **int**[len];

res[0] = nums[0];

**if**(len>1)

res[1] = Math.*max*(nums[0], nums[1]);

**for**(**int** i=2; i<len; i++){

res[i] = Math.*max*(res[i-2]+nums[i], res[i-1]);

}

**return** res[len-1];

}

}