**Name:\_\_\_\_\_\_\_\_\_\_\_\_Shuqing Ye\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_UCI NET ID:\_\_\_\_\_\_\_shuqiny2\_\_\_\_\_\_\_\_**

|  |  |
| --- | --- |
| Test cases (including the edge cases):  input:[2,4,1], output: 3  input: [1], output: 2  input: [999, 998, … 3, 2], output: 1 | time complexity: O(n)  space complexity:O(n) |

|  |  |
| --- | --- |
| / \*main algorithm: XOR  If a == b, a ^ b == 0  If a != 0, a ^ 0 == a  Array A = {1,2,…n}, array B = {1,2,…n-1}  If we XOR every item in A and B, since we can always find a corresponding item in A that makes B[i] == A[j], then the result of B[i] ^ A[j] is equal to 0. Thus the final call will be A[j] ^ 0 = A[j], then we find the missing item.  \*/  int findMissingItem(List<Integer> numList) {  int n = numList.size();  // invalid input  if (n < 1)  return -1;  // set array = {1,2, …, n + 1}  int[] array = new int[n + 2];  for(int i = 1; i <= n + 1; i++)  array[i] = i;  return XOR(numList, array);  } | // helper func: XOR items  int XOR(List<Integer> numList, int[] array) {  int res = 0, i = 1;  for (int num : numList)  res ^= num ^ array[i++];  res ^= array[i];  return res;  } |