**922. Sort Array By Parity II**

Given an array A of non-negative integers, half of the integers in A are odd, and half of the integers are even.

Sort the array so that whenever A[i] is odd, i is odd; and whenever A[i] is even, i is even.

You may return any answer array that satisfies this condition.

我的答案：

|  |
| --- |
| class Solution {  public:  vector<int> sortArrayByParityII(vector<int>& A) {  vector<int> sw[2];  /\*  \*\* sw[0] save the even swap number;  \*\* sw[1] save the odd swap number;  \*/  for(int i = 0; i < A.size(); i++) {  if((i + A[i]) % 2 != 0) {  bool flag = i % 2;  if(!sw[!flag].empty()) {  int k = A[sw[!flag].back()];  A[sw[!flag].back()] = A[i];  A[i] = k;  sw[!flag].pop\_back();  }  else  sw[flag].push\_back(i);  }  }  return A;  }  }; |
| Runtime: 100 ms, faster than 77.30% of C++ online submissions for Sort Array By Parity II. |
| 分析：  是否是因为使用容器的操作过多导致运行时间增加？ |
| 快速一些的答案：  vector<int> sortArrayByParityII(vector<int>& A) {  vector<int> sortArrayByParityII(A.size());  int even=0, odd=1;  for (int i=0; i<A.size(); i++) {  if (A[i]%2==0) {  sortArrayByParityII[even]=A[i];  even+=2;  } else {  sortArrayByParityII[odd]=A[i];  odd+=2;  }  }  return sortArrayByParityII;  } |
| 官方推荐的两种方法：  Approach 1: Two Pass  **Intuition and Algorithm**  Read all the even integers and put them into places ans[0], ans[2], ans[4], and so on.  Then, read all the odd integers and put them into places ans[1], ans[3], ans[5], etc.  class Solution(object):  def sortArrayByParityII(self, A):  N = len(A)  ans = [None] \* N  t = 0  for i, x in enumerate(A):  if x % 2 == 0:  ans[t] = x  t += 2  t = 1  for i, x in enumerate(A):  if x % 2 == 1:  ans[t] = x  t += 2  # We could have also used slice assignment:  # ans[::2] = (x for x in A if x % 2 == 0)  # ans[1::2] = (x for x in A if x % 2 == 1)  return ans Approach 2: Read / Write Heads **Intuition**  We are motivated (perhaps by the interviewer) to pursue a solution where we modify the original array A in place.  First, it is enough to put all even elements in the correct place, since all odd elements will be in the correct place too. So let's only focus on A[0], A[2], A[4], ...  Ideally, we would like to have some partition where everything to the left is already correct, and everything to the right is undecided.  Indeed, this idea works if we separate it into two slices even = A[0], A[2], A[4], ... and odd = A[1], A[3], A[5], .... Our invariant will be that everything less than i in the even slice is correct, and everything less than j in the odd slice is correct.  **Algorithm**  For each even i, let's make A[i] even. To do it, we will draft an element from the odd slice. We pass j through the odd slice until we find an even element, then swap. Our invariant is maintained, so the algorithm is correct.  class Solution(object):  def sortArrayByParityII(self, A):  j = 1  for i in xrange(0, len(A), 2):  if A[i] % 2:  while A[j] % 2:  j += 2  A[i], A[j] = A[j], A[i]  return A |