Global and Local Inversions

We have some permutation `A` of `[0, 1, ..., N - 1]`, where `N` is the length of `A`.

The number of (global) inversions is the number of `i < j` with `0 <= i < j < N` and `A[i] > A[j]`.

The number of local inversions is the number of `i` with `0 <= i < N` and `A[i] > A[i+1]`.

Return `true` if and only if the number of global inversions is equal to the number of local inversions.

Example 1:

```
Input: A = [1,0,2]
Output: true
Explanation: There is 1 global inversion, and 1 local inversion.
```

Example 2:

```
Input: A = [1,2,0]
Output: false
Explanation: There are 2 global inversions, and 1 local inversion.
```

Note:

- `A` will be a permutation of `[0, 1, ..., A.length - 1]`.
- `A` will have length in range `[1, 5000]`.
- The time limit for this problem has been reduced.

Solution 1

The original order should be [0, 1, 2, 3, 4...], the number i should be on the position i. We just check the offset of each number, if the absolute value is larger than 1, means the number of global inversion must be bigger than local inversion, because a local inversion is a global inversion, but a global one may not be local.

```
class Solution {
public:
    bool isIdealPermutation(vector<int>& A) {
 for (int i = 0; i < A.size(); ++i) {
            if (abs(A[i] - i) > 1) return false;
        }
 return true;
    }
};
```

written by grandyang original link here

Solution 2

Key insights:

- every local inversion is also a global inversion
- so "local inversions == global inversions" can be interpreted as "there are only local inversions"
- if there are only local inversions, the array will be sorted after making all local inversions
- if there are inversions that are not local, the array won't be sorted after making all local inversions

```
class Solution(object):
    def isIdealPermutation(self, A):
        """
        :type A: List[int]
        :rtype: bool
        """
        N=len(A)
        for i in xrange(1, N):
            if A[i-1]==A[i]+1:
                A[i], A[i-1]=A[i-1], A[i]
            elif A[i-1]!=i-1:
                return False
        return True
```

written by erjoalgo original link here

Solution 3

All local inversions are global inversions.
If the number of global inversions is equal to the number of local inversions,
it means that all global inversions in permutations are local inversions.
It also means that we can not find `A[i] > A[j]` with `i+2<=j`.
In other words, `max(A[i]) < A[i+2]`

In this first solution, I traverse `A` and keep the current biggest number `cmax`.
Then I check the condition `cmax < A[i+2]`

Here come this solutions:

C++:

```
bool isIdealPermutation(vector<int>& A) {
        int cmax = 0, n = A.size();
        for (int i = 0; i < n - 2; ++i) {
            cmax = max(cmax, A[i]);
            if (cmax > A[i + 2]) return false;
        }
        return true;
    }
```

Java:

```
public boolean isIdealPermutation(int[] A) {
        int cmax = 0;
        for (int i = 0; i < A.size() - 2; ++i) {
            cmax = Math.max(cmax, A[i]);
            if (cmax > A[i + 2]) return false;
        }
        return true;
    }
```

Python:

```
def isIdealPermutation(self, A):
        cmax = 0
        for i in range(len(A) - 2):
            cmax = max(cmax, A[i])
            if cmax > A[i + 2]:
                return False
        return True
```

Basic on this idea, I tried to arrange an ideal permutation. Then I found that to place number `i`,
I could only place `i` at `A[i-1]`, `A[i]` or `A[i+1]`. So it came up to me, It will be easier just to check if all `A[i] - i` equals to -1, 0 or 1.

C++:

```
bool isIdealPermutation(vector<int>& A) {
        for (int i = 0; i < A.size(); ++i) if (abs(A[i] - i) > 1) return false;
        return true;
 }
```

Java:

```
public boolean isIdealPermutation(int[] A) {
        for (int i = 0; i < A.length; ++i) if (Math.abs(A[i] - i) > 1) return false
;
        return true;
}
```

Python:

```
def isIdealPermutation(self, A):
        return all(abs(i - v) <= 1 for i, v in enumerate(A))
```

written by lee215 original link here