First Missing Positive

Given an unsorted integer array, find the first missing positive integer.

For example,
Given `[1,2,0]` return `3`,
and `[3,4,-1,1]` return `2`.

Your algorithm should run in $O(n)$ time and uses constant space.

Solution 1

Put each number in its right place.

For example:

When we find 5, then swap it with A[4].

At last, the first place where its number is not right, return the place + 1.

```
class Solution
{
public:
    int firstMissingPositive(int A[], int n)
    {
        for(int i = 0; i < n; ++ i)
            while(A[i] > 0 && A[i] <= n && A[A[i] - 1] != A[i])
                swap(A[i], A[A[i] - 1]);

        for(int i = 0; i < n; ++ i)
            if(A[i] != i + 1)
                return i + 1;

        return n + 1;
    }
};
```

written by makuiyu original link here

Solution 2

Share my O(n)/O(1) solution

The basic idea is***for any k positive numbers (duplicates allowed), the first missing positive number must be within [1,k+1]***. The reason is like you put k balls into k+1 bins, there must be a bin empty, the empty bin can be viewed as the missing number.

---

1. Unfortunately, there are 0 and negative numbers in the array, so firstly I think of using partition technique (used in quick sort) to put all positive numbers together in one side. This can be finished in O(n) time, O(1) space.
2. After partition step, you get all the positive numbers lying within A[0,k-1]. Now, According to the basic idea, I infer the first missing number must be within [1,k+1]. I decide to use A[i] (0<=i<=k-1) to indicate whether the number (i+1) exists. But here I still have to main the original information A[i] holds. Fortunately, A[i] are all positive numbers, so I can set them to negative to indicate the existence of (i+1) and I can still use abs(A[i]) to get the original information A[i] holds.
3. After step 2, I can again scan all elements between A[0,k-1] to find the first positive element A[i], that means (i+1) doesn't exist, which is what I want.

```
 public int firstMissingPositive(int[] A) {
    int n=A.length;
    if(n==0)
        return 1;
    int k=partition(A)+1;
    int temp=0;
    int first_missing_Index=k;
    for(int i=0;i<k;i++){
        temp=Math.abs(A[i]);
        if(temp<=k)
            A[temp-1]=(A[temp-1]<0)?A[temp-1]:-A[temp-1];
    }
    for(int i=0;i<k;i++){
        if(A[i]>0){
            first_missing_Index=i;
            break;
        }
    }
    return first_missing_Index+1;
}

public int partition(int[] A){
    int n=A.length;
    int q=-1;
    for(int i=0;i<n;i++){
        if(A[i]>0){
            q++;
            swap(A,q,i);
        }
    }
    return q;
}

public void swap(int[] A, int i, int j){
    if(i!=j){
        A[i]^=A[j];
        A[j]^=A[i];
        A[i]^=A[j];
    }
}
```

written by yuyibestman original link here

Solution 3

time complexity is O(N) and space complexity is O(1).
Link: http://stackoverflow.com/questions/1586858/find-the-smallest-integer-not-in-a-list
Posted by Ants Aasma on Oct 20 '09.

The code is pasted here:

```
#Pass 1, move every value to the position of its value
for cursor in range(N):
    target = array[cursor]
    while target < N and target != array[target]:
        new_target = array[target]
        array[target] = target
        target = new_target

#Pass 2, find first location where the index doesn't match the value
for cursor in range(N):
    if array[cursor] != cursor:
        return cursor
return N
```

written by yzhao original link here