Maximum Swap

Given a non-negative integer, you could swap two digits **at most** once to get the maximum valued number. Return the maximum valued number you could get.

Example 1:

```
Input: 2736
Output: 7236
Explanation: Swap the number 2 and the number 7.
```

Example 2:

```
Input: 9973
Output: 9973
Explanation: No swap.
```

Note:

1. The given number is in the range [0, 10^8]

Solution 1

Use buckets to record the last position of digit 0 ~ 9 in this num.

Loop through the num array from left to right. For each position, we check whether there exists a larger digit in this num (start from 9 to current digit). We also need to make sure the position of this larger digit is behind the current one. If we find it, simply swap these two digits and return the result.

```
class Solution {
    public int maximumSwap(int num) {
        char[] digits = Integer.toString(num).toCharArray();

        int[] buckets = new int[10];
        for (int i = 0; i < digits.length; i++) {
            buckets[digits[i] - '0'] = i;
        }

        for (int i = 0; i < digits.length; i++) {
            for (int k = 9; k > digits[i] - '0'; k--) {
                if (buckets[k] > i) {
                    char tmp = digits[i];
                    digits[i] = digits[buckets[k]];
                    digits[buckets[k]] = tmp;
                    return Integer.valueOf(new String(digits));
                }
            }
        }

        return num;
    }
}
```

written by joeztang original link here

Solution 2

The below algo runs in O(n) time. The performance is pretty good.

Logic :-

1. We keep neglecting the digits until they are decreasing in value because there are no 2 digits we can find in this subset which we can swap to get a greater number than original. For ex. 9631.
2. While iterating through the digits if we find the current > next digit, we store current as min and its index as minIndex. If we reach the end of the array which means that all the digits followed a decreasing trend, then there is nothing to swap and we return the original number.
3. From the minIndex, we start our search for the greatest digit in the remaining digits. This will be our max and its index, maxIndex. The max is the digit we want to swap, but whom to swap with? The minIndex? That would be incorrect, as seen in this ex. 97482, where the min is 4 and max is 8.
   (correct answer -> 98472)
4. There could be a digit < max whose index < minIndex. For the above ex., it is the digit 7. Hence we'll iterate through the digits from beginning till minIndex to find a digit that satisfies the above criteria. The index of that digit would be swapIndex and we want to swap the maxIndex with swapIndex to get the maximum number possible.

In all, steps 1,2, and 3 take one pass through the digits that requires O(n) time. Step 4, in worst case will require another pass, taking O(n) time.

```java
class Solution {
    public int maximumSwap(int num) {
        String temp = Integer.toString(num);     //Convert to int array (num -> digits[])
        int[] digits = new int[temp.length()];
        for (int i = 0; i < temp.length(); i++){
            digits[i] = temp.charAt(i) - '0';
        }                                        //Ignore all digits until decreasing, until (next digit > prev). store the min and minindex
        int min = digits[0], minIndex = 0;
        for (int i = 0; i < digits.length - 1; i++){
            if (digits[i + 1] > digits[i]) {
                minIndex = i;
                min = digits[i];
                break;
            } else if (i == digits.length - 2)  //Reached end. Nothing to swap. Return original number.
                return num;
        }                                        //Starting from minindex find the largest digit in the remaining digits.
        int max = min, maxIndex = -1;
        for (int j = minIndex; j < digits.length; j++){
            if (digits[j] >= max) {
                max = digits[j];
                maxIndex = j;
            }
        }                                        //Iterate through the array till minIndex to find any digit that might be lesser than max
        int result = 0, swapindex = minIndex;
        for (int i = 0; i <= minIndex; i++){
            if (digits[i] < max) {
                swapindex = i;
                break;
            }
        }                                        //Swap the maxIndex digit with swapIndex
        int tmp = digits[swapindex];
        digits[swapindex] = digits[maxIndex];
        digits[maxIndex] = tmp;
                                                 //Convert the result into integer(digits -> result)
        for (int i = digits.length - 1, j = 0; i >= 0; i--) {
            result = result + (digits[j] * ((int) Math.pow(10, i)));
            j++;
        }
        return result;
    }
}
```

written by enigma.sidd original link here

Solution 3

We only need to record a position array (pos[i]) which represent the largest number index form the i position.

For instance,
input 9 8 6 3 8
pos 0 4 4 4 4
input[pos] 9 8 8 8 8

Therefore, we only need to find the first input[i] != input[pos[i]]
and swap it.

Another example
input 5 4 3 1 2
pos 0 1 2 4 4
input[pos] 5 4 3 2 2

```
int maximumSwap(int num) {
    string numString = to_string(num);
    int n = numString.length();
    vector<int> dpPosition(n, -1);

    int curMaxPos = n - 1;
    for (int i = n - 1; i >= 0; i--) {
        if (numString[i] > numString[curMaxPos]) {
            curMaxPos = i;
        }
        dpPosition[i] = curMaxPos;
    }

    for (int i = 0; i < n; i++) {
        if(numString[dpPosition[i]] != numString[i]) {
            swap(numString[i], numString[dpPosition[i]]);
            break;
        }
    }

    return stoi(numString);
}
```

written by suilan0602 original link here

From Leetcoder.