Create Maximum Number

Given two arrays of length `m` and `n` with digits `0-9` representing two numbers. Create the maximum number of length `k from digits of the two. The relative order of the digits from the same array must be preserved. Return an array of the k digits. You should try to optimize your time and space complexity.`

Example 1:

nums1 = `[3, 4, 6, 5]`
nums2 = `[9, 1, 2, 5, 8, 3]`
k = `5`
return `[9, 8, 6, 5, 3]`

Example 2:

nums1 = `[6, 7]`
nums2 = `[6, 0, 4]`
k = `5`
return `[6, 7, 6, 0, 4]`

Example 3:

nums1 = `[3, 9]`
nums2 = `[8, 9]`
k = `3`
return `[9, 8, 9]`

Credits:
Special thanks to @dietpepsi for adding this problem and creating all test cases.

Solution 1

Many of the posts have the same algorithm. In short we can first solve 2 simpler problem

1. Create the maximum number of one array
2. Create the maximum number of two array using all of their digits.

For an long and detailed explanation see my blog here.

The algorithm is O((m+n)^3) in the worst case. It runs in 22 ms.

Java

```java
public int[] maxNumber(int[] nums1, int[] nums2, int k) {
    int n = nums1.length;
    int m = nums2.length;
    int[] ans = new int[k];
    for (int i = Math.max(0, k - m); i <= k && i <= n; ++i) {
        int[] candidate = merge(maxArray(nums1, i), maxArray(nums2, k - i), k);
        if (greater(candidate, 0, ans, 0)) ans = candidate;
    }
    return ans;
}
private int[] merge(int[] nums1, int[] nums2, int k) {
    int[] ans = new int[k];
    for (int i = 0, j = 0, r = 0; r < k; ++r)
        ans[r] = greater(nums1, i, nums2, j) ? nums1[i++] : nums2[j++];
    return ans;
}
public boolean greater(int[] nums1, int i, int[] nums2, int j) {
    while (i < nums1.length && j < nums2.length && nums1[i] == nums2[j]) {
        i++;
        j++;
    }
    return j == nums2.length || (i < nums1.length && nums1[i] > nums2[j]);
}
public int[] maxArray(int[] nums, int k) {
    int n = nums.length;
    int[] ans = new int[k];
    for (int i = 0, j = 0; i < n; ++i) {
        while (n - i + j > k && j > 0 && ans[j - 1] < nums[i]) j--;
        if (j < k) ans[j++] = nums[i];
    }
    return ans;
}
```

written by dietpepsi original link here

Solution 2

Python

```
def maxNumber(self, nums1, nums2, k):

    def prep(nums, k):
        drop = len(nums) - k
        out = []
        for num in nums:
            while drop and out and out[-1] < num:
                out.pop()
                drop -= 1
            out.append(num)
        return out[:k]

    def merge(a, b):
        return [max(a, b).pop(0) for _ in a+b]

    return max(merge(prep(nums1, i), prep(nums2, k-i))
               for i in range(k+1)
               if i <= len(nums1) and k-i <= len(nums2))
```

Solved it on my own but now I see others already posted this idea. Oh well, at least it's short, particularly my `merge` function.

The last two lines can be combined, but I find it rather ugly and not worth it: `for i in range(max(k-len(nums2), 0), min(k, len(nums1))+1))`

Ruby

```
def prep(nums, k)
  drop = nums.size - k
  out = [9]
  nums.each do |num|
    while drop > 0 && out[-1] < num
      out.pop
      drop -= 1
    end
    out << num
  end
  out[1..k]
end

def max_number(nums1, nums2, k)
  ([k-nums2.size, 0].max .. [nums1.size, k].min).map { |k1|
    parts = [prep(nums1, k1), prep(nums2, k-k1)]
    (1..k).map { parts.max.shift }
  }.max
end
```

C++

Translated it to C++ as well now. Not as short anymore, but still decent. And C++ allows different functions with the same name, so I chose to do that here to show how nicely the `maxNumber(nums1, nums2, k)` problem can be based on the problems `maxNumber(nums, k)` and `maxNumber(nums1, nums2)`, which would make fine problems on their own.

```
vector<int> maxNumber(vector<int>& nums1, vector<int>& nums2, int k) {
    int n1 = nums1.size(), n2 = nums2.size();
    vector<int> best;
    for (int k1=max(k-n2, 0); k1<=min(k, n1); ++k1)
        best = max(best, maxNumber(maxNumber(nums1, k1),
                                   maxNumber(nums2, k-k1)));
    return best;
}

vector<int> maxNumber(vector<int> nums, int k) {
    int drop = nums.size() - k;
    vector<int> out;
    for (int num : nums) {
        while (drop && out.size() && out.back() < num) {
            out.pop_back();
            drop--;
        }
        out.push_back(num);
    }
    out.resize(k);
    return out;
}

vector<int> maxNumber(vector<int> nums1, vector<int> nums2) {
    vector<int> out;
    while (nums1.size() + nums2.size()) {
        vector<int>& now = nums1 > nums2 ? nums1 : nums2;
        out.push_back(now[0]);
        now.erase(now.begin());
    }
    return out;
}
```

An alternative for `maxNumber(nums1, nums2)`:

```
vector<int> maxNumber(vector<int> nums1, vector<int> nums2) {
    vector<int> out;
    auto i1 = nums1.begin(), end1 = nums1.end();
    auto i2 = nums2.begin(), end2 = nums2.end();
    while (i1 != end1 || i2 != end2)
        out.push_back(lexicographical_compare(i1, end1, i2, end2) ? *i2++ : *i1++
);
    return out;
}
```

written by StefanPochmann original link here

Solution 3

To find the maximum ,we can enumerate how digits we should get from nums1 , we suppose it is i.

So , the digits from nums2 is K - i.

And we can use a stack to get the get maximum number(x digits) from one array.

OK, Once we choose two maximum subarray , we should combine it to the answer.

It is just like merger sort, but we should pay attention to the case: the two digital are equal.

we should find the digits behind it to judge which digital we should choose now.

In other words,we should judge which subarry is bigger than the other.

That's all.

If you have any question or suggest, I am happy you can comment on my blog : Create Maximum Number.

Thanks, merry christmas :)

update:use stack to find max sub array and it runs 21ms now.(thanks to @dietpepsi)

```
 /**  * Created by hrwhisper on 2015/11/23.  * http://www.hrwhisper.me/leetcode-c
reate-maximum-number/  */


public class Solution {
    public int[] maxNumber(int[] nums1, int[] nums2, int k) {
        int get_from_nums1 = Math.min(nums1.length, k);
        int[] ans = new int[k];
        for (int i = Math.max(k - nums2.length, 0); i <= get_from_nums1; i++) {
            int[] res1 = new int[i];
            int[] res2 = new int[k - i];
            int[] res = new int[k];
            res1 = solve(nums1, i);
            res2 = solve(nums2, k - i);
            int pos1 = 0, pos2 = 0, tpos = 0;

            while (res1.length > 0 && res2.length > 0 && pos1 < res1.length && po
s2 < res2.length) {
                if (compare(res1, pos1, res2, pos2))
                    res[tpos++] = res1[pos1++];
                else
                    res[tpos++] = res2[pos2++];
            }
            while (pos1 < res1.length)
                res[tpos++] = res1[pos1++];
            while (pos2 < res2.length)
                res[tpos++] = res2[pos2++];

            if (!compare(ans, 0, res, 0))
                ans = res;
        }

        return ans;
    }

    public boolean compare(int[] nums1, int start1, int[] nums2, int start2) {
        for (; start1 < nums1.length && start2 < nums2.length; start1++, start2++
) {
            if (nums1[start1] > nums2[start2]) return true;
            if (nums1[start1] < nums2[start2]) return false;
        }
        return start1 != nums1.length;
    }

    public int[] solve(int[] nums, int k) {
        int[] res = new int[k];
        int len = 0;
        for (int i = 0; i < nums.length; i++) {
            while (len > 0 && len + nums.length - i > k && res[len - 1] < nums[i]
) {
                len--;
            }
            if (len < k)
                res[len++] = nums[i];
        }
        return res;
    } }
```

written by hrwhisper original link here