321. Create Maximum Number

Total Accepted: **6508** Total Submissions: **30792** Difficulty: **Hard**

Given two arrays of length m and n with digits 0-9 representing two numbers. Create the maximum number of length k <= m + n 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]

```
//C++
//ZZW
class Solution {
public:
    void getMax(int *nums,int len, int *result,int t,int &sortedLen)
    {
        int n,top=0;
        int needs2drop = len - t;
        result[0] = nums[0];
        for(int i=1;i<len;i++)
        {
            n = nums[i];
            while(top>0 && (i-top<=needs2drop) && result[top]<n) --top;
            if(i-top>needs2drop)
            {
                sortedLen = max(top,1);
```

```
            while(++top<t){result[top] = nums[i++];}
            return;
        }
        if(++top<t) result[top] = n;
        else{
            top = t-1;
        }
    }
}

// k is final maximum number
void dp(int *nums,int len, int &sortedLen, int minL, int maxL,int *res,int k)
{
    int j, *head, *prehead = res;
    const int soi = sizeof(int);
    getMax(nums,len,res,maxL,sortedLen);
    for(int l=maxL;l>max(minL,1);l--)
    {
        head = prehead + k;
        memcpy(head,prehead,l*soi);
        for(j=sortedLen;j<l;j++)
        {
            if(head[j]>head[j-1])
            {
                sortedLen = max(1,j-1);
                memcpy(head+j-1,prehead+j,soi*(l-j));
                break;
            }
        }
        if(j == l) sortedLen = l;
        prehead = head;
    }

}
void merge(int *nums1,int len1,int *nums2,int len2,int *res, int ressize)
{
    int i=0,j=0,k=0;//i -> result, j -> nums1, k -> nums2;
    while(i<ressize)
    {
        if(j<len1 && k<len2)
        {
            if(nums1[j]>nums2[k])
            {
                res[i++] = nums1[j++];
            }
            else if(nums1[j]<nums2[k])
            {
                res[i++] = nums2[k++];
            }else
            {
                int remaining1 = len1-j;int tmp = nums1[j];
                int remaining2 = len2-k;
                int flag = memcmp(nums1+j,nums2+k,sizeof(int)*min(remaining1,remaining2));
                flag = (flag==0?(remaining1>remaining2?1:-1):flag);
                int *nums = flag>0?  nums1:nums2;
                int &cnt = flag>0? j:k;
                int len = flag>0?len1:len2;
                while(nums[cnt]==tmp && i<ressize && cnt<len) res[i++]=nums[cnt++];
            }
        }
        else if(j<len1) res[i++] = nums1[j++];
        else res[i++] = nums2[k++];
    }
}

vector<int> maxNumber(vector<int>& nums1, vector<int>& nums2, int k){
          int soi = sizeof(int);
          int len1 = nums1.size();
          int len2 = nums2.size();
```

```
            int step = k*soi;
            int minL1 = max(0,k-len2);int maxL1=min(k,len1);
            int minL2 = k - maxL1; int maxL2 = k - minL1;
            int range = maxL1 - minL1 + 1;
            int *res = new int[range*2*k + 2*k];int *dp1 = res+k; int *dp2 = res+range*k+k;
            int *tmp = res + range*2*k+k;
            memset(res,0,step);
            int sortedLen1=1; int sortedLen2=1;
            if(len1==0 && len2>0) getMax(&nums2[0],len2,res,k,sortedLen2);
            else if(len1>0 && len2==0) getMax(&nums1[0],len1,res,k,sortedLen1);
            else if(len1>0 && len2>0)
            {
                    dp(&nums1[0], len1, sortedLen1, minL1, maxL1, dp1,k);
          dp(&nums2[0], len2, sortedLen2, minL2, maxL2, dp2,k);
                    if(sortedLen1+sortedLen2>k)
                    {
                            merge(dp1+k*(maxL1-sortedLen1),sortedLen1,dp2+k*(maxL2-
sortedLen2),sortedLen2,res,k);
                    }else{
                            for (int i=minL1;i<=maxL1;i++)
                            {
                                    merge(dp1+k*(maxL1-i),i,dp2+k*(maxL2-(k-i)),k-i,tmp,k);
                                    if(memcmp(res,tmp,step)<0){
                                            memcpy(res,tmp,step);
                                    }
                            }
                    }
            }
            vector<int> resv(res, res + k);
            delete res;
            return resv;
      }
};
```