

# Problem 321: Create Maximum Number

## Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

You are given two integer arrays

nums1

and

nums2

of lengths

m

and

n

respectively.

nums1

and

nums2

represent the digits of two numbers. You are also given an integer

k

Create the maximum number of length

$k \leq m + n$

from digits of the two numbers. The relative order of the digits from the same array must be preserved.

Return an array of the

k

digits representing the answer.

Example 1:

Input:

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

Output:

[9,8,6,5,3]

Example 2:

Input:

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

Output:

[6,7,6,0,4]

Example 3:

Input:

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

Output:

[9,8,9]

Constraints:

m == nums1.length

n == nums2.length

1 <= m, n <= 500

0 <= nums1[i], nums2[i] <= 9

1 <= k <= m + n

nums1

and

nums2

do not have leading zeros.

## Code Snippets

C++:

```
class Solution {  
public:  
vector<int> maxNumber(vector<int>& nums1, vector<int>& nums2, int k) {  
}  
};
```

**Java:**

```
class Solution {  
    public int[] maxNumber(int[] nums1, int[] nums2, int k) {  
  
    }  
}
```

**Python3:**

```
class Solution:  
    def maxNumber(self, nums1: List[int], nums2: List[int], k: int) -> List[int]:
```

**Python:**

```
class Solution(object):  
    def maxNumber(self, nums1, nums2, k):  
        """  
        :type nums1: List[int]  
        :type nums2: List[int]  
        :type k: int  
        :rtype: List[int]  
        """
```

**JavaScript:**

```
/**  
 * @param {number[]} nums1  
 * @param {number[]} nums2  
 * @param {number} k  
 * @return {number[]}  
 */  
var maxNumber = function(nums1, nums2, k) {  
  
};
```

**TypeScript:**

```
function maxNumber(nums1: number[], nums2: number[], k: number): number[] {  
  
};
```

**C#:**

```
public class Solution {  
    public int[] MaxNumber(int[] nums1, int[] nums2, int k) {  
  
    }  
}
```

**C:**

```
/**  
 * Note: The returned array must be malloced, assume caller calls free().  
 */  
int* maxNumber(int* nums1, int nums1Size, int* nums2, int nums2Size, int k,  
int* returnSize) {  
  
}
```

**Go:**

```
func maxNumber(nums1 []int, nums2 []int, k int) []int {  
  
}
```

**Kotlin:**

```
class Solution {  
    fun maxNumber(nums1: IntArray, nums2: IntArray, k: Int): IntArray {  
  
    }  
}
```

**Swift:**

```
class Solution {  
    func maxNumber(_ nums1: [Int], _ nums2: [Int], _ k: Int) -> [Int] {  
  
    }  
}
```

**Rust:**

```
impl Solution {  
    pub fn max_number(nums1: Vec<i32>, nums2: Vec<i32>, k: i32) -> Vec<i32> {  
        }  
    }  
}
```

### Ruby:

```
# @param {Integer[]} nums1  
# @param {Integer[]} nums2  
# @param {Integer} k  
# @return {Integer[]}  
def max_number(nums1, nums2, k)  
  
end
```

### PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums1  
     * @param Integer[] $nums2  
     * @param Integer $k  
     * @return Integer[]  
     */  
    function maxNumber($nums1, $nums2, $k) {  
  
    }  
}
```

### Dart:

```
class Solution {  
    List<int> maxNumber(List<int> nums1, List<int> nums2, int k) {  
        }  
    }
```

### Scala:

```
object Solution {  
    def maxNumber(nums1: Array[Int], nums2: Array[Int], k: Int): Array[Int] = {
```

```
}
```

```
}
```

### Elixir:

```
defmodule Solution do
  @spec max_number(nums1 :: [integer], nums2 :: [integer], k :: integer) :: [integer]
  def max_number(nums1, nums2, k) do
    end
    end
```

### Erlang:

```
-spec max_number(Nums1 :: [integer()], Nums2 :: [integer()], K :: integer()) -> [integer()].
max_number(Nums1, Nums2, K) ->
.
```

### Racket:

```
(define/contract (max-number numsl nums2 k)
  (-> (listof exact-integer?) (listof exact-integer?) exact-integer? (listof
  exact-integer?)))
  )
```

## Solutions

### C++ Solution:

```
/*
 * Problem: Create Maximum Number
 * Difficulty: Hard
 * Tags: array, greedy, stack
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
*/
```

```

*/



class Solution {
public:
vector<int> maxNumber(vector<int>& nums1, vector<int>& nums2, int k) {

}
};

```

### Java Solution:

```

/**
 * Problem: Create Maximum Number
 * Difficulty: Hard
 * Tags: array, greedy, stack
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

class Solution {
public int[] maxNumber(int[] nums1, int[] nums2, int k) {

}
}

```

### Python3 Solution:

```

"""
Problem: Create Maximum Number
Difficulty: Hard
Tags: array, greedy, stack

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def maxNumber(self, nums1: List[int], nums2: List[int], k: int) -> List[int]:

```

```
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
    def maxNumber(self, nums1, nums2, k):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :type k: int
        :rtype: List[int]
        """

```

### JavaScript Solution:

```
/**
 * Problem: Create Maximum Number
 * Difficulty: Hard
 * Tags: array, greedy, stack
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

/**
 * @param {number[]} nums1
 * @param {number[]} nums2
 * @param {number} k
 * @return {number[]}
 */
var maxNumber = function(nums1, nums2, k) {
}
```

### TypeScript Solution:

```
/**
 * Problem: Create Maximum Number
 * Difficulty: Hard

```

```

* Tags: array, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

```

```

function maxNumber(nums1: number[], nums2: number[], k: number): number[] {
}

```

### C# Solution:

```

/*
* Problem: Create Maximum Number
* Difficulty: Hard
* Tags: array, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

```

```

public class Solution {
    public int[] MaxNumber(int[] nums1, int[] nums2, int k) {
}
}

```

### C Solution:

```

/*
* Problem: Create Maximum Number
* Difficulty: Hard
* Tags: array, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

```

```

/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* maxNumber(int* nums1, int nums1Size, int* nums2, int nums2Size, int k,
int* returnSize) {

}

```

### Go Solution:

```

// Problem: Create Maximum Number
// Difficulty: Hard
// Tags: array, greedy, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func maxNumber(nums1 []int, nums2 []int, k int) []int {
}

```

### Kotlin Solution:

```

class Solution {
    fun maxNumber(nums1: IntArray, nums2: IntArray, k: Int): IntArray {
        }
    }

```

### Swift Solution:

```

class Solution {
    func maxNumber(_ nums1: [Int], _ nums2: [Int], _ k: Int) -> [Int] {
        }
    }

```

### Rust Solution:

```

// Problem: Create Maximum Number
// Difficulty: Hard
// Tags: array, greedy, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn max_number(nums1: Vec<i32>, nums2: Vec<i32>, k: i32) -> Vec<i32> {

}
}

```

### Ruby Solution:

```

# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @param {Integer} k
# @return {Integer[]}
def max_number(nums1, nums2, k)

end

```

### PHP Solution:

```

class Solution {

/**
 * @param Integer[] $nums1
 * @param Integer[] $nums2
 * @param Integer $k
 * @return Integer[]
 */
function maxNumber($nums1, $nums2, $k) {

}
}

```

### Dart Solution:

```
class Solution {  
    List<int> maxNumber(List<int> nums1, List<int> nums2, int k) {  
        }  
    }  
}
```

### Scala Solution:

```
object Solution {  
    def maxNumber(nums1: Array[Int], nums2: Array[Int], k: Int): Array[Int] = {  
        }  
    }  
}
```

### Elixir Solution:

```
defmodule Solution do  
    @spec max_number(nums1 :: [integer], nums2 :: [integer], k :: integer) ::  
        [integer]  
    def max_number(nums1, nums2, k) do  
  
    end  
    end
```

### Erlang Solution:

```
-spec max_number(Nums1 :: [integer()], Nums2 :: [integer()], K :: integer())  
-> [integer()].  
max_number(Nums1, Nums2, K) ->  
.
```

### Racket Solution:

```
(define/contract (max-number nums1 nums2 k)  
  (-> (listof exact-integer?) (listof exact-integer?) exact-integer? (listof  
    exact-integer?))  
)
```