Problem 3684: Maximize Sum of At Most K Distinct Elements

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given a

positive

integer array

nums

and an integer

k

.

Choose at most

k

elements from

nums

so that their sum is maximized. However, the chosen numbers must be

distinct

.

Return an array containing the chosen numbers in

strictly descending

order.

Example 1:

Input:

nums = [84,93,100,77,90], k = 3

Output:

[100,93,90]

Explanation:

The maximum sum is 283, which is attained by choosing 93, 100 and 90. We rearrange them in strictly descending order as

[100, 93, 90]

.

Example 2:

Input:

nums = [84,93,100,77,93], k = 3

Output:

[100,93,84]

Explanation:

The maximum sum is 277, which is attained by choosing 84, 93 and 100. We rearrange them in strictly descending order as

[100, 93,

84

]

. We cannot choose 93, 100 and 93 because the chosen numbers must be distinct.

Example 3:

Input:

nums = [1,1,1,2,2,2], k = 6

Output:

[2,1]

Explanation:

The maximum sum is 3, which is attained by choosing 1 and 2. We rearrange them in strictly descending order as

[2, 1]

.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 10

9

1 <= k <= nums.length

Code Snippets

C++:

```
class Solution {
public:
vector<int> maxKDistinct(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public int[] maxKDistinct(int[] nums, int k) {

}
}
```

Python3:

```
class Solution:
def maxKDistinct(self, nums: List[int], k: int) -> List[int]:
```

Python:

```
class Solution(object):
def maxKDistinct(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} k
* @return {number[]}
*/
```

```
var maxKDistinct = function(nums, k) {

};
```

TypeScript:

```
function maxKDistinct(nums: number[], k: number): number[] {

};
```

C#:

```
public class Solution {
public int[] MaxKDistinct(int[] nums, int k) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* maxKDistinct(int* nums, int numsSize, int k, int* returnSize) {

}
```

Go:

```
func maxKDistinct(nums []int, k int) []int {

}
```

Kotlin:

```
class Solution {
fun maxKDistinct(nums: IntArray, k: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func maxKDistinct(_ nums: [Int], _ k: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn max_k_distinct(nums: Vec<i32>, k: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer[]}
def max_k_distinct(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer[]
*/
function maxKDistinct($nums, $k) {

}
}
```

Dart:

```
class Solution {
List<int> maxKDistinct(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def maxKDistinct(nums: Array[Int], k: Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_k_distinct(nums :: [integer], k :: integer) :: [integer]
def max_k_distinct(nums, k) do

end
end
```

Erlang:

```
-spec max_k_distinct(Nums :: [integer()], K :: integer()) -> [integer()].
max_k_distinct(Nums, K) ->
.
```

Racket:

```
(define/contract (max-k-distinct nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize Sum of At Most K Distinct Elements
* Difficulty: Easy
* Tags: array, greedy, hash, sort
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<int> maxKDistinct(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize Sum of At Most K Distinct Elements
* Difficulty: Easy
* Tags: array, greedy, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int[] maxKDistinct(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Sum of At Most K Distinct Elements
Difficulty: Easy
Tags: array, greedy, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""
```

```
class Solution:
def maxKDistinct(self, nums: List[int], k: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxKDistinct(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Maximize Sum of At Most K Distinct Elements
* Difficulty: Easy
* Tags: array, greedy, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} nums
* @param {number} k
* @return {number[]}
*/
var maxKDistinct = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize Sum of At Most K Distinct Elements
* Difficulty: Easy
* Tags: array, greedy, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function maxKDistinct(nums: number[], k: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Maximize Sum of At Most K Distinct Elements
* Difficulty: Easy
* Tags: array, greedy, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int[] MaxKDistinct(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximize Sum of At Most K Distinct Elements
* Difficulty: Easy
* Tags: array, greedy, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* maxKDistinct(int* nums, int numsSize, int k, int* returnSize) {

}
```

Go Solution:

```
// Problem: Maximize Sum of At Most K Distinct Elements
// Difficulty: Easy
// Tags: array, greedy, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func maxKDistinct(nums []int, k int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun maxKDistinct(nums: IntArray, k: Int): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func maxKDistinct(_ nums: [Int], _ k: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Maximize Sum of At Most K Distinct Elements
// Difficulty: Easy
// Tags: array, greedy, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn max_k_distinct(nums: Vec<i32>, k: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer[]}
def max_k_distinct(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer[]
*/
function maxKDistinct($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
List<int> maxKDistinct(List<int> nums, int k) {

}
}
```

Scala Solution:

```
object Solution {
def maxKDistinct(nums: Array[Int], k: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_k_distinct(nums :: [integer], k :: integer) :: [integer]
def max_k_distinct(nums, k) do

end
end
```

Erlang Solution:

```
-spec max_k_distinct(Nums :: [integer()], K :: integer()) -> [integer()].
max_k_distinct(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (max-k-distinct nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```