Problem 1191: K-Concatenation Maximum Sum

Problem Information

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

Problem Description

Given an integer array

arr

and an integer

k

, modify the array by repeating it

k

times.

For example, if

arr = [1, 2]

and

k = 3

then the modified array will be

[1, 2, 1, 2, 1, 2]

.

Return the maximum sub-array sum in the modified array. Note that the length of the sub-array can be

0

and its sum in that case is

0

.

As the answer can be very large, return the answer

modulo

10

9

+ 7

.

Example 1:

Input:

arr = [1,2], k = 3

Output:

9

Example 2:

Input:

arr = [1,-2,1], k = 5

Output:

2

Example 3:

Input:

arr = [-1,-2], k = 7

Output:

0

Constraints:

1 <= arr.length <= 10

5

1 <= k <= 10

5

-10

4

<= arr[i] <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int kConcatenationMaxSum(vector<int>& arr, int k) {

}
};
```

Java:

```
class Solution {
public int kConcatenationMaxSum(int[] arr, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} arr
* @param {number} k
* @return {number}
*/
var kConcatenationMaxSum = function(arr, k) {

};
```

TypeScript:

```
function kConcatenationMaxSum(arr: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int KConcatenationMaxSum(int[] arr, int k) {

}
}
```

C:

```
int kConcatenationMaxSum(int* arr, int arrSize, int k) {

}
```

Go:

```
func kConcatenationMaxSum(arr []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun kConcatenationMaxSum(arr: IntArray, k: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @param {Integer} k
# @return {Integer}
def k_concatenation_max_sum(arr, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $k
* @return Integer
*/
function kConcatenationMaxSum($arr, $k) {

}
}
```

Dart:

```
class Solution {
int kConcatenationMaxSum(List<int> arr, int k) {

}
}
```

Scala:

```
object Solution {
def kConcatenationMaxSum(arr: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec k_concatenation_max_sum(arr :: [integer], k :: integer) :: integer
def k_concatenation_max_sum(arr, k) do

end
end
```

Erlang:

```
-spec k_concatenation_max_sum(Arr :: [integer()], K :: integer()) ->
integer().
k_concatenation_max_sum(Arr, K) ->
.
```

Racket:

```
(define/contract (k-concatenation-max-sum arr k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: K-Concatenation Maximum Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
```

```
int kConcatenationMaxSum(vector<int>& arr, int k) {

}
};
```

Java Solution:

```
/**
* Problem: K-Concatenation Maximum Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int kConcatenationMaxSum(int[] arr, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: K-Concatenation Maximum Sum
Difficulty: Medium
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: K-Concatenation Maximum Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: K-Concatenation Maximum Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function kConcatenationMaxSum(arr: number[], k: number): number {
};
```

C# Solution:

```
/*
* Problem: K-Concatenation Maximum Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int KConcatenationMaxSum(int[] arr, int k) {

}
}
```

C Solution:

```
/*
* Problem: K-Concatenation Maximum Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int kConcatenationMaxSum(int* arr, int arrSize, int k) {

}
```

Go Solution:

```
// Problem: K-Concatenation Maximum Sum
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func kConcatenationMaxSum(arr []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun kConcatenationMaxSum(arr: IntArray, k: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: K-Concatenation Maximum Sum
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn k_concatenation_max_sum(arr: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @param {Integer} k
# @return {Integer}
def k_concatenation_max_sum(arr, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $k
* @return Integer
*/
function kConcatenationMaxSum($arr, $k) {

}
}
```

Dart Solution:

```
class Solution {
int kConcatenationMaxSum(List<int> arr, int k) {

}
}
```

Scala Solution:

```
object Solution {
def kConcatenationMaxSum(arr: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec k_concatenation_max_sum(arr :: [integer], k :: integer) :: integer
def k_concatenation_max_sum(arr, k) do

end
end
```

Erlang Solution:

```
-spec k_concatenation_max_sum(Arr :: [integer()], K :: integer()) ->
integer().
k_concatenation_max_sum(Arr, K) ->
.
```

Racket Solution:

```
(define/contract (k-concatenation-max-sum arr k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```