Problem 950: Reveal Cards In Increasing Order

Problem Information

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

Problem Description

You are given an integer array

deck

. There is a deck of cards where every card has a unique integer. The integer on the

i

th

card is

deck[i]

.

You can order the deck in any order you want. Initially, all the cards start face down (unrevealed) in one deck.

You will do the following steps repeatedly until all cards are revealed:

Take the top card of the deck, reveal it, and take it out of the deck.

If there are still cards in the deck then put the next top card of the deck at the bottom of the deck.

If there are still unrevealed cards, go back to step 1. Otherwise, stop.

Return

an ordering of the deck that would reveal the cards in increasing order

.

Note

that the first entry in the answer is considered to be the top of the deck.

Example 1:

Input:

deck = [17,13,11,2,3,5,7]

Output:

[2,13,3,11,5,17,7]

Explanation:

We get the deck in the order [17,13,11,2,3,5,7] (this order does not matter), and reorder it. After reordering, the deck starts as [2,13,3,11,5,17,7], where 2 is the top of the deck. We reveal 2, and move 13 to the bottom. The deck is now [3,11,5,17,7,13]. We reveal 3, and move 11 to the bottom. The deck is now [5,17,7,13,11]. We reveal 5, and move 17 to the bottom. The deck is now [7,13,11,17]. We reveal 7, and move 13 to the bottom. The deck is now [11,17,13]. We reveal 11, and move 17 to the bottom. The deck is now [13,17]. We reveal 13, and move 17 to the bottom. The deck is now [17]. We reveal 17. Since all the cards revealed are in increasing order, the answer is correct.

Example 2:

Input:

deck = [1,1000]

Output:

[1,1000]

Constraints:

1 <= deck.length <= 1000

1 <= deck[i] <= 10

6

All the values of

deck

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
vector<int> deckRevealedIncreasing(vector<int>& deck) {

}
};
```

Java:

```
class Solution {
public int[] deckRevealedIncreasing(int[] deck) {

}
}
```

Python3:

```
class Solution:
def deckRevealedIncreasing(self, deck: List[int]) -> List[int]:
```

Python:

```
class Solution(object):
def deckRevealedIncreasing(self, deck):
"""
:type deck: List[int]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} deck
* @return {number[]}
*/
var deckRevealedIncreasing = function(deck) {

};
```

TypeScript:

```
function deckRevealedIncreasing(deck: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] DeckRevealedIncreasing(int[] deck) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* deckRevealedIncreasing(int* deck, int deckSize, int* returnSize) {

}
```

Go:

```
func deckRevealedIncreasing(deck []int) []int {

}
```

Kotlin:

```
class Solution {
fun deckRevealedIncreasing(deck: IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func deckRevealedIncreasing(_ deck: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn deck_revealed_increasing(deck: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} deck
# @return {Integer[]}
def deck_revealed_increasing(deck)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[] $deck
* @return Integer[]
*/
function deckRevealedIncreasing($deck) {

}
}
```

Dart:

```
class Solution {
List<int> deckRevealedIncreasing(List<int> deck) {

}
}
```

Scala:

```
object Solution {
def deckRevealedIncreasing(deck: Array[Int]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec deck_revealed_increasing(deck :: [integer]) :: [integer]
def deck_revealed_increasing(deck) do

end
end
```

Erlang:

```
-spec deck_revealed_increasing(Deck :: [integer()]) -> [integer()].
deck_revealed_increasing(Deck) ->
.
```

Racket:

```
(define/contract (deck-revealed-increasing deck)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Reveal Cards In Increasing Order
* Difficulty: Medium
* Tags: array, sort, queue
*
* 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> deckRevealedIncreasing(vector<int>& deck) {

}
};
```

Java Solution:

```
/**
* Problem: Reveal Cards In Increasing Order
* Difficulty: Medium
* Tags: array, sort, queue
*
* 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[] deckRevealedIncreasing(int[] deck) {

}
}
```

Python3 Solution:

```
"""
Problem: Reveal Cards In Increasing Order
Difficulty: Medium
Tags: array, sort, queue

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 deckRevealedIncreasing(self, deck: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def deckRevealedIncreasing(self, deck):
"""
:type deck: List[int]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Reveal Cards In Increasing Order
* Difficulty: Medium
* Tags: array, sort, queue
*
```

```
* 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[]} deck
* @return {number[]}
*/
var deckRevealedIncreasing = function(deck) {

};
```

TypeScript Solution:

```
/**
* Problem: Reveal Cards In Increasing Order
* Difficulty: Medium
* Tags: array, sort, queue
*
* 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 deckRevealedIncreasing(deck: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Reveal Cards In Increasing Order
* Difficulty: Medium
* Tags: array, sort, queue
*
* 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[] DeckRevealedIncreasing(int[] deck) {

}
}
```

C Solution:

```
/*
* Problem: Reveal Cards In Increasing Order
* Difficulty: Medium
* Tags: array, sort, queue
*
* 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* deckRevealedIncreasing(int* deck, int deckSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Reveal Cards In Increasing Order
// Difficulty: Medium
// Tags: array, sort, queue
//
// 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 deckRevealedIncreasing(deck []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun deckRevealedIncreasing(deck: IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func deckRevealedIncreasing(_ deck: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Reveal Cards In Increasing Order
// Difficulty: Medium
// Tags: array, sort, queue
//
// 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 deck_revealed_increasing(deck: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} deck
# @return {Integer[]}
def deck_revealed_increasing(deck)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $deck
* @return Integer[]
*/
function deckRevealedIncreasing($deck) {

}
}
```

Dart Solution:

```
class Solution {
List<int> deckRevealedIncreasing(List<int> deck) {

}
}
```

Scala Solution:

```
object Solution {
def deckRevealedIncreasing(deck: Array[Int]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec deck_revealed_increasing(deck :: [integer]) :: [integer]
def deck_revealed_increasing(deck) do

end
end
```

Erlang Solution:

```
-spec deck_revealed_increasing(Deck :: [integer()]) -> [integer()].
deck_revealed_increasing(Deck) ->
.
```

Racket Solution:

```
(define/contract (deck-revealed-increasing deck)
(-> (listof exact-integer?) (listof exact-integer?))
)
```