Problem 1340: Jump Game V

Problem Information

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

Problem Description

Given an array of integers

arr

and an integer

d

. In one step you can jump from index

i

to index:

i + x

where:

i + x < arr.length

and

0 < x <= d

.

i - x

where:

i - x >= 0

and

0 < x <= d

.

In addition, you can only jump from index

i

to index

j

if

arr[i] > arr[j]

and

arr[i] > arr[k]

for all indices

k

between

i

and

j

(More formally

min(i, j) < k < max(i, j)

).

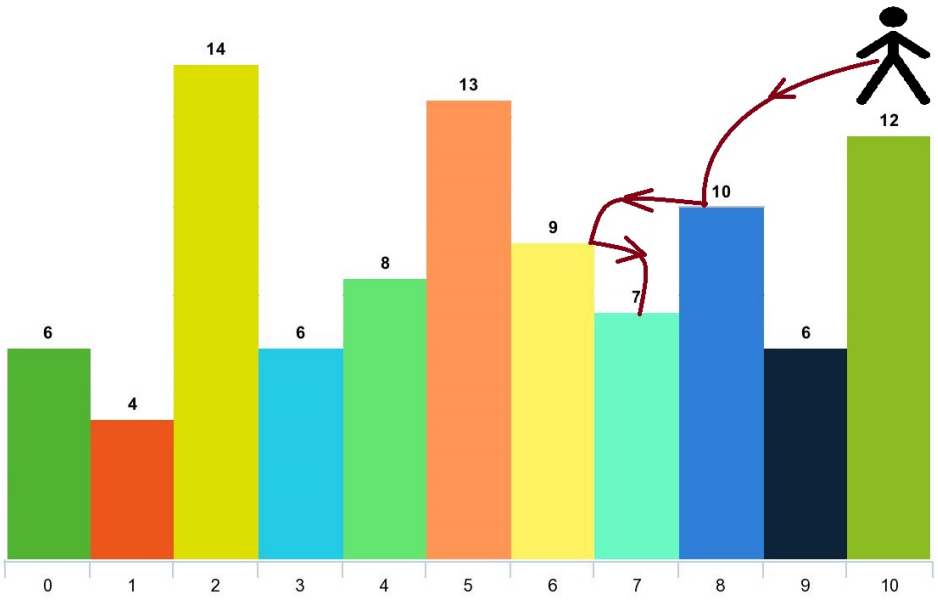
You can choose any index of the array and start jumping. Return

the maximum number of indices

you can visit.

Notice that you can not jump outside of the array at any time.

Example 1:



Input:

arr = [6,4,14,6,8,13,9,7,10,6,12], d = 2

Output:

4

Explanation:

You can start at index 10. You can jump 10 --> 8 --> 6 --> 7 as shown. Note that if you start at index 6 you can only jump to index 7. You cannot jump to index 5 because 13 > 9. You cannot jump to index 4 because index 5 is between index 4 and 6 and 13 > 9. Similarly You cannot jump from index 3 to index 2 or index 1.

Example 2:

Input:

arr = [3,3,3,3,3], d = 3

Output:

1

Explanation:

You can start at any index. You always cannot jump to any index.

Example 3:

Input:

arr = [7,6,5,4,3,2,1], d = 1

Output:

7

Explanation:

Start at index 0. You can visit all the indicies.

Constraints:

1 <= arr.length <= 1000

1 <= arr[i] <= 10

5

1 <= d <= arr.length

Code Snippets

C++:

```
class Solution {
public:
int maxJumps(vector<int>& arr, int d) {

}
};
```

Java:

```
class Solution {
public int maxJumps(int[] arr, int d) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxJumps(arr: number[], d: number): number {

};
```

C#:

```
public class Solution {
public int MaxJumps(int[] arr, int d) {

}
}
```

C:

```
int maxJumps(int* arr, int arrSize, int d) {

}
```

Go:

```
func maxJumps(arr []int, d int) int {

}
```

Kotlin:

```
class Solution {
fun maxJumps(arr: IntArray, d: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @param {Integer} d
# @return {Integer}
def max_jumps(arr, d)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $d
* @return Integer
*/
function maxJumps($arr, $d) {

}
}
```

Dart:

```
class Solution {
int maxJumps(List<int> arr, int d) {
```

```
}
}
```

Scala:

```
object Solution {
def maxJumps(arr: Array[Int], d: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_jumps(arr :: [integer], d :: integer) :: integer
def max_jumps(arr, d) do

end
end
```

Erlang:

```
-spec max_jumps(Arr :: [integer()], D :: integer()) -> integer().
max_jumps(Arr, D) ->
.
```

Racket:

```
(define/contract (max-jumps arr d)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Jump Game V
* Difficulty: Hard
```

```
* Tags: array, dp, sort
*
* 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 maxJumps(vector<int>& arr, int d) {

}
};
```

Java Solution:

```
/**
* Problem: Jump Game V
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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 maxJumps(int[] arr, int d) {

}
}
```

Python3 Solution:

```
"""
Problem: Jump Game V
Difficulty: Hard
Tags: array, dp, sort

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 maxJumps(self, arr: List[int], d: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Jump Game V
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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} d
* @return {number}
*/
var maxJumps = function(arr, d) {

};
```

TypeScript Solution:

```
/**
* Problem: Jump Game V
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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 maxJumps(arr: number[], d: number): number {

};
```

C# Solution:

```
/*
* Problem: Jump Game V
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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 MaxJumps(int[] arr, int d) {

}
}
```

C Solution:

```
/*
* Problem: Jump Game V
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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 maxJumps(int* arr, int arrSize, int d) {

}
```

Go Solution:

```
// Problem: Jump Game V
// Difficulty: Hard
// Tags: array, dp, sort
//
// 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 maxJumps(arr []int, d int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxJumps(arr: IntArray, d: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Jump Game V
// Difficulty: Hard
// Tags: array, dp, sort
```

```
//
// 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 max_jumps(arr: Vec<i32>, d: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @param {Integer} d
# @return {Integer}
def max_jumps(arr, d)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $d
* @return Integer
*/
function maxJumps($arr, $d) {

}
}
```

Dart Solution:

```
class Solution {
int maxJumps(List<int> arr, int d) {

}
}
```

Scala Solution:

```
object Solution {
def maxJumps(arr: Array[Int], d: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_jumps(arr :: [integer], d :: integer) :: integer
def max_jumps(arr, d) do

end
end
```

Erlang Solution:

```
-spec max_jumps(Arr :: [integer()], D :: integer()) -> integer().
max_jumps(Arr, D) ->
.
```

Racket Solution:

```
(define/contract (max-jumps arr d)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```