Problem 2528: Maximize the Minimum Powered City

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

stations

of length

n

, where

stations[i]

represents the number of power stations in the

i

th

city.

Each power station can provide power to every city in a fixed

range

. In other words, if the range is denoted by

r

, then a power station at city

i

can provide power to all cities

j

such that

|i - j| <= r

and

0 <= i, j <= n - 1

.

Note that

|x|

denotes

absolute

value. For example,

|7 - 5| = 2

and

|3 - 10| = 7

.

The

power

of a city is the total number of power stations it is being provided power from.

The government has sanctioned building

k

more power stations, each of which can be built in any city, and have the same range as the pre-existing ones.

Given the two integers

r

and

k

, return

the

maximum possible minimum power

of a city, if the additional power stations are built optimally.

Note

that you can build the

k

power stations in multiple cities.

Example 1:

Input:

stations = [1,2,4,5,0], r = 1, k = 2

Output:

5

Explanation:

One of the optimal ways is to install both the power stations at city 1. So stations will become [1,4,4,5,0]. - City 0 is provided by 1 + 4 = 5 power stations. - City 1 is provided by 1 + 4 + 4 = 9 power stations. - City 2 is provided by 4 + 4 + 5 = 13 power stations. - City 3 is provided by 5 + 4 = 9 power stations. - City 4 is provided by 5 + 0 = 5 power stations. So the minimum power of a city is 5. Since it is not possible to obtain a larger power, we return 5.

Example 2:

Input:

stations = [4,4,4,4], r = 0, k = 3

Output:

4

Explanation:

It can be proved that we cannot make the minimum power of a city greater than 4.

Constraints:

n == stations.length

1 <= n <= 10

5

0 <= stations[i] <= 10

5

0 <= r <= n - 1

0 <= k <= 10

9

Code Snippets

C++:

```
class Solution {
public:
long long maxPower(vector<int>& stations, int r, int k) {

}
};
```

Java:

```
class Solution {
public long maxPower(int[] stations, int r, int k) {

}
}
```

Python3:

```
class Solution:
def maxPower(self, stations: List[int], r: int, k: int) -> int:
```

Python:

```
class Solution(object):
def maxPower(self, stations, r, k):
```

```
"""
:type stations: List[int]
:type r: int
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} stations
* @param {number} r
* @param {number} k
* @return {number}
*/
var maxPower = function(stations, r, k) {

};
```

TypeScript:

```
function maxPower(stations: number[], r: number, k: number): number {

};
```

C#:

```
public class Solution {
public long MaxPower(int[] stations, int r, int k) {

}
}
```

C:

```
long long maxPower(int* stations, int stationsSize, int r, int k) {

}
```

Go:

```
func maxPower(stations []int, r int, k int) int64 {

}
```

Kotlin:

```
class Solution {
fun maxPower(stations: IntArray, r: Int, k: Int): Long {

}
}
```

Swift:

```
class Solution {
func maxPower(_ stations: [Int], _ r: Int, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_power(stations: Vec<i32>, r: i32, k: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} stations
# @param {Integer} r
# @param {Integer} k
# @return {Integer}
def max_power(stations, r, k)

end
```

PHP:

```
class Solution {

/**
```

```
* @param Integer[] $stations
* @param Integer $r
* @param Integer $k
* @return Integer
*/
function maxPower($stations, $r, $k) {

}
}
```

Dart:

```
class Solution {
int maxPower(List<int> stations, int r, int k) {

}
}
```

Scala:

```
object Solution {
def maxPower(stations: Array[Int], r: Int, k: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_power(stations :: [integer], r :: integer, k :: integer) :: integer
def max_power(stations, r, k) do

end
end
```

Erlang:

```
-spec max_power(Stations :: [integer()], R :: integer(), K :: integer()) ->
integer().
max_power(Stations, R, K) ->
.
```

Racket:

```
(define/contract (max-power stations r k)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize the Minimum Powered City
* Difficulty: Hard
* Tags: array, greedy, search, 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:
long long maxPower(vector<int>& stations, int r, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize the Minimum Powered City
* Difficulty: Hard
* Tags: array, greedy, search, 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 long maxPower(int[] stations, int r, int k) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Maximize the Minimum Powered City
Difficulty: Hard
Tags: array, greedy, search, 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 maxPower(self, stations: List[int], r: int, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxPower(self, stations, r, k):
"""
:type stations: List[int]
:type r: int
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximize the Minimum Powered City
* Difficulty: Hard
* Tags: array, greedy, search, 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[]} stations
* @param {number} r
* @param {number} k
* @return {number}
*/
var maxPower = function(stations, r, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize the Minimum Powered City
* Difficulty: Hard
* Tags: array, greedy, search, 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 maxPower(stations: number[], r: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximize the Minimum Powered City
* Difficulty: Hard
* Tags: array, greedy, search, 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 long MaxPower(int[] stations, int r, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximize the Minimum Powered City
* Difficulty: Hard
* Tags: array, greedy, search, 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
*/

long long maxPower(int* stations, int stationsSize, int r, int k) {

}
```

Go Solution:

```
// Problem: Maximize the Minimum Powered City
// Difficulty: Hard
// Tags: array, greedy, search, 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 maxPower(stations []int, r int, k int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxPower(stations: IntArray, r: Int, k: Int): Long {
```

```
}
}
```

Swift Solution:

```
class Solution {
func maxPower(_ stations: [Int], _ r: Int, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximize the Minimum Powered City
// Difficulty: Hard
// Tags: array, greedy, search, 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 max_power(stations: Vec<i32>, r: i32, k: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} stations
# @param {Integer} r
# @param {Integer} k
# @return {Integer}
def max_power(stations, r, k)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $stations
* @param Integer $r
* @param Integer $k
* @return Integer
*/
function maxPower($stations, $r, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maxPower(List<int> stations, int r, int k) {

}
}
```

Scala Solution:

```
object Solution {
def maxPower(stations: Array[Int], r: Int, k: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_power(stations :: [integer], r :: integer, k :: integer) :: integer
def max_power(stations, r, k) do

end
end
```

Erlang Solution:

```
-spec max_power(Stations :: [integer()], R :: integer(), K :: integer()) ->
integer().
max_power(Stations, R, K) ->
```

```
.
```

Racket Solution:

```
(define/contract (max-power stations r k)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```