Problem 2616: Minimize the Maximum Difference of Pairs

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

and an integer

p

. Find

p

pairs of indices of

nums

such that the

maximum

difference amongst all the pairs is

minimized

. Also, ensure no index appears more than once amongst the

p

pairs.

Note that for a pair of elements at the index

i

and

j

, the difference of this pair is

|nums[i] - nums[j]|

, where

|x|

represents the

absolute

value

of

x

.

Return

the

minimum

maximum

difference among all

p

pairs.

We define the maximum of an empty set to be zero.

Example 1:

Input:

nums = [10,1,2,7,1,3], p = 2

Output:

1

Explanation:

The first pair is formed from the indices 1 and 4, and the second pair is formed from the indices 2 and 5. The maximum difference is max(|nums[1] - nums[4]|, |nums[2] - nums[5]|) = max(0, 1) = 1. Therefore, we return 1.

Example 2:

Input:

nums = [4,2,1,2], p = 1

Output:

0

Explanation:

Let the indices 1 and 3 form a pair. The difference of that pair is |2 - 2| = 0, which is the minimum we can attain.

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

9

0 <= p <= (nums.length)/2

Code Snippets

C++:

```
class Solution {
public:
int minimizeMax(vector<int>& nums, int p) {

}
};
```

Java:

```
class Solution {
public int minimizeMax(int[] nums, int p) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} p
* @return {number}
*/
var minimizeMax = function(nums, p) {

};
```

TypeScript:

```
function minimizeMax(nums: number[], p: number): number {

};
```

C#:

```
public class Solution {
public int MinimizeMax(int[] nums, int p) {

}
}
```

C:

```
int minimizeMax(int* nums, int numsSize, int p) {

}
```

Go:

```
func minimizeMax(nums []int, p int) int {

}
```

Kotlin:

```
class Solution {
fun minimizeMax(nums: IntArray, p: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} p
# @return {Integer}
def minimize_max(nums, p)

end
```

PHP:

```
class Solution {
```

```
/**
* @param Integer[] $nums
* @param Integer $p
* @return Integer
*/
function minimizeMax($nums, $p) {

}
}
```

Dart:

```
class Solution {
int minimizeMax(List<int> nums, int p) {

}
}
```

Scala:

```
object Solution {
def minimizeMax(nums: Array[Int], p: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimize_max(nums :: [integer], p :: integer) :: integer
def minimize_max(nums, p) do

end
end
```

Erlang:

```
-spec minimize_max(Nums :: [integer()], P :: integer()) -> integer().
minimize_max(Nums, P) ->
.
```

Racket:

```
(define/contract (minimize-max nums p)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimize the Maximum Difference of Pairs
* Difficulty: Medium
* Tags: array, dp, greedy, sort, search
*
* 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 minimizeMax(vector<int>& nums, int p) {

}
};
```

Java Solution:

```
/**
* Problem: Minimize the Maximum Difference of Pairs
* Difficulty: Medium
* Tags: array, dp, greedy, sort, search
*
* 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 minimizeMax(int[] nums, int p) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimize the Maximum Difference of Pairs
Difficulty: Medium
Tags: array, dp, greedy, sort, search

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 minimizeMax(self, nums: List[int], p: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimize the Maximum Difference of Pairs
* Difficulty: Medium
* Tags: array, dp, greedy, sort, search
*
* 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[]} nums
* @param {number} p
* @return {number}
*/
var minimizeMax = function(nums, p) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimize the Maximum Difference of Pairs
* Difficulty: Medium
* Tags: array, dp, greedy, sort, search
*
* 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 minimizeMax(nums: number[], p: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimize the Maximum Difference of Pairs
* Difficulty: Medium
* Tags: array, dp, greedy, sort, search
*
* 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 MinimizeMax(int[] nums, int p) {

}
}
```

C Solution:

```c
/*
* Problem: Minimize the Maximum Difference of Pairs
* Difficulty: Medium
* Tags: array, dp, greedy, sort, search
*
* 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 minimizeMax(int* nums, int numsSize, int p) {

}
```

Go Solution:

```go
// Problem: Minimize the Maximum Difference of Pairs
// Difficulty: Medium
// Tags: array, dp, greedy, sort, search
//
// 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 minimizeMax(nums []int, p int) int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun minimizeMax(nums: IntArray, p: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimize the Maximum Difference of Pairs
// Difficulty: Medium
// Tags: array, dp, greedy, sort, search
//
// 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 minimize_max(nums: Vec<i32>, p: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} p
# @return {Integer}
def minimize_max(nums, p)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $p
* @return Integer
*/
function minimizeMax($nums, $p) {
}
}
```

Dart Solution:

```
class Solution {
int minimizeMax(List<int> nums, int p) {

}
}
```

Scala Solution:

```
object Solution {
def minimizeMax(nums: Array[Int], p: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimize_max(nums :: [integer], p :: integer) :: integer
def minimize_max(nums, p) do

end
end
```

Erlang Solution:

```
-spec minimize_max(Nums :: [integer()], P :: integer()) -> integer().
minimize_max(Nums, P) ->
.
```

Racket Solution:

```
(define/contract (minimize-max nums p)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```