Problem 740: Delete and Earn

Problem Information

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

Problem Description

You are given an integer array

nums

. You want to maximize the number of points you get by performing the following operation any number of times:

Pick any

nums[i]

and delete it to earn

nums[i]

points. Afterwards, you must delete

every

element equal to

nums[i] - 1

and

every

element equal to

nums[i] + 1

.

Return

the

maximum number of points

you can earn by applying the above operation some number of times

.

Example 1:

Input:

nums = [3,4,2]

Output:

6

Explanation:

You can perform the following operations: - Delete 4 to earn 4 points. Consequently, 3 is also deleted. nums = [2]. - Delete 2 to earn 2 points. nums = []. You earn a total of 6 points.

Example 2:

Input:

nums = [2,2,3,3,3,4]

Output:

9

Explanation:

You can perform the following operations: - Delete a 3 to earn 3 points. All 2's and 4's are also deleted. nums = [3,3]. - Delete a 3 again to earn 3 points. nums = [3]. - Delete a 3 once more to earn 3 points. nums = []. You earn a total of 9 points.

Constraints:

1 <= nums.length <= 2 * 10

4

1 <= nums[i] <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int deleteAndEarn(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int deleteAndEarn(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function deleteAndEarn(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int DeleteAndEarn(int[] nums) {

}
}
```

C:

```
int deleteAndEarn(int* nums, int numsSize) {

}
```

Go:

```
func deleteAndEarn(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun deleteAndEarn(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func deleteAndEarn(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn delete_and_earn(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def delete_and_earn(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
```

```
*/
function deleteAndEarn($nums) {

}
}
```

Dart:

```
class Solution {
int deleteAndEarn(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def deleteAndEarn(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec delete_and_earn(nums :: [integer]) :: integer
def delete_and_earn(nums) do

end
end
```

Erlang:

```
-spec delete_and_earn(Nums :: [integer()]) -> integer().
delete_and_earn(Nums) ->
.
```

Racket:

```
(define/contract (delete-and-earn nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Delete and Earn
* Difficulty: Medium
* Tags: array, dp, hash
*
* 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 deleteAndEarn(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Delete and Earn
* Difficulty: Medium
* Tags: array, dp, hash
*
* 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 deleteAndEarn(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Delete and Earn
Difficulty: Medium
Tags: array, dp, hash

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Delete and Earn
* Difficulty: Medium
* Tags: array, dp, hash
*
* 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
* @return {number}
*/
var deleteAndEarn = function(nums) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Delete and Earn
* Difficulty: Medium
* Tags: array, dp, hash
*
* 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 deleteAndEarn(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Delete and Earn
* Difficulty: Medium
* Tags: array, dp, hash
*
* 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 DeleteAndEarn(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Delete and Earn
* Difficulty: Medium
```

```
* Tags: array, dp, hash
*
* 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 deleteAndEarn(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Delete and Earn
// Difficulty: Medium
// Tags: array, dp, hash
//
// 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 deleteAndEarn(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun deleteAndEarn(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func deleteAndEarn(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Delete and Earn
// Difficulty: Medium
// Tags: array, dp, hash
//
// 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 delete_and_earn(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def delete_and_earn(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function deleteAndEarn($nums) {

}
}
```

Dart Solution:

```
class Solution {
int deleteAndEarn(List<int> nums) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def deleteAndEarn(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec delete_and_earn(nums :: [integer]) :: integer
def delete_and_earn(nums) do

end
end
```

Erlang Solution:

```
-spec delete_and_earn(Nums :: [integer()]) -> integer().
delete_and_earn(Nums) ->
.
```

Racket Solution:

```
(define/contract (delete-and-earn nums)
(-> (listof exact-integer?) exact-integer?)
)
```