# Problem 3080: Mark Elements on Array by Performing Queries

## Problem Information

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

## Problem Description

You are given a

0-indexed

array

nums

of size

n

consisting of positive integers.

You are also given a 2D array

queries

of size

m

where

queries[i] = [index

i

, k

i

]

.

Initially all elements of the array are

unmarked

.

You need to apply

m

queries on the array in order, where on the

i

th

query you do the following:

Mark the element at index

index

i

if it is not already marked.

Then mark

k

i

unmarked elements in the array with the

smallest

values. If multiple such elements exist, mark the ones with the smallest indices. And if less than

k

i

unmarked elements exist, then mark all of them.

Return

an array answer of size

m

where

answer[i]

is the

sum

of unmarked elements in the array after the

i

th

query

.

Example 1:

Input:

nums = [1,2,2,1,2,3,1], queries = [[1,2],[3,3],[4,2]]

Output:

[8,3,0]

Explanation:

We do the following queries on the array:

Mark the element at index

1

, and

2

of the smallest unmarked elements with the smallest indices if they exist, the marked elements now are

nums = [

1

,

2

,2,

1

,2,3,1]

. The sum of unmarked elements is

2 + 2 + 3 + 1 = 8

.

Mark the element at index

3

, since it is already marked we skip it. Then we mark

3

of the smallest unmarked elements with the smallest indices, the marked elements now are

nums = [

1

,

2

,

2

,

1

,

2

,3,

1

]

. The sum of unmarked elements is

3

.

Mark the element at index

4

, since it is already marked we skip it. Then we mark

2

of the smallest unmarked elements with the smallest indices if they exist, the marked elements now are

nums = [

1

,

2

,

2

,

1

,

2

,

3

,

1

]

. The sum of unmarked elements is

0

.

Example 2:

Input:

nums = [1,4,2,3], queries = [[0,1]]

Output:

[7]

Explanation:

We do one query which is mark the element at index

0

and mark the smallest element among unmarked elements. The marked elements will be

nums = [

1

,4,

2

,3]

, and the sum of unmarked elements is

4 + 3 = 7

.

Constraints:

n == nums.length

m == queries.length

1 <= m <= n <= 10

5

1 <= nums[i] <= 10

5

queries[i].length == 2

0 <= index

i

, k

i

<= n - 1

## Code Snippets

### C++:

```
class Solution {
public:
vector<long long> unmarkedSumArray(vector<int>& nums, vector<vector<int>>&
queries) {

}
};
```

### Java:

```
class Solution {
public long[] unmarkedSumArray(int[] nums, int[][] queries) {

}
}
```

### Python3:

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

### Python:

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

### JavaScript:

```
/**
* @param {number[]} nums
* @param {number[][]} queries
```

```
* @return {number[]}
*/
var unmarkedSumArray = function(nums, queries) {

};
```

**TypeScript:**

```
function unmarkedSumArray(nums: number[], queries: number[][]): number[] {

};
```

**C#:**

```
public class Solution {
public long[] UnmarkedSumArray(int[] nums, int[][] queries) {

}
}
```

**C:**

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* unmarkedSumArray(int* nums, int numsSize, int** queries, int
queriesSize, int* queriesColSize, int* returnSize) {

}
```

**Go:**

```
func unmarkedSumArray(nums []int, queries [][]int) []int64 {

}
```

**Kotlin:**

```
class Solution {
fun unmarkedSumArray(nums: IntArray, queries: Array<IntArray>): LongArray {

}
```

```
}
```

**Swift:**

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

}
}
```

**Rust:**

```
impl Solution {
pub fn unmarked_sum_array(nums: Vec<i32>, queries: Vec<Vec<i32>>) -> Vec<i64>
{

}
}
```

**Ruby:**

```
# @param {Integer[]} nums
# @param {Integer[][]} queries
# @return {Integer[]}
def unmarked_sum_array(nums, queries)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $queries
* @return Integer[]
*/
function unmarkedSumArray($nums, $queries) {

}
}
```

**Dart:**

```
class Solution {
List<int> unmarkedSumArray(List<int> nums, List<List<int>> queries) {

}
}
```

**Scala:**

```
object Solution {
def unmarkedSumArray(nums: Array[Int], queries: Array[Array[Int]]):
Array[Long] = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec unmarked_sum_array(nums :: [integer], queries :: [[integer]]) ::
[integer]
def unmarked_sum_array(nums, queries) do

end
end
```

**Erlang:**

```
-spec unmarked_sum_array(Nums :: [integer()], Queries :: [[integer()]]) ->
[integer()].
unmarked_sum_array(Nums, Queries) ->
.
```

**Racket:**

```
(define/contract (unmarked-sum-array nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Mark Elements on Array by Performing Queries
* Difficulty: Medium
* Tags: array, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<long long> unmarkedSumArray(vector<int>& nums, vector<vector<int>>&
queries) {

}
};
```

### Java Solution:

```
/**
* Problem: Mark Elements on Array by Performing Queries
* Difficulty: Medium
* Tags: array, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public long[] unmarkedSumArray(int[] nums, int[][] queries) {

}
}
```

### Python3 Solution:

```
"""
Problem: Mark Elements on Array by Performing Queries
Difficulty: Medium
Tags: array, hash, sort, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def unmarkedSumArray(self, nums: List[int], queries: List[List[int]]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Mark Elements on Array by Performing Queries
* Difficulty: Medium
* Tags: array, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} nums
* @param {number[][]} queries
* @return {number[]}
```

```
*/
var unmarkedSumArray = function(nums, queries) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Mark Elements on Array by Performing Queries
* Difficulty: Medium
* Tags: array, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function unmarkedSumArray(nums: number[], queries: number[][]): number[] {

};
```

**C# Solution:**

```
/*
* Problem: Mark Elements on Array by Performing Queries
* Difficulty: Medium
* Tags: array, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public long[] UnmarkedSumArray(int[] nums, int[][] queries) {

}
}
```

**C Solution:**

```
/*
* Problem: Mark Elements on Array by Performing Queries
* Difficulty: Medium
* Tags: array, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* unmarkedSumArray(int* nums, int numsSize, int** queries, int
queriesSize, int* queriesColSize, int* returnSize) {

}
```

**Go Solution:**

```
// Problem: Mark Elements on Array by Performing Queries
// Difficulty: Medium
// Tags: array, hash, sort, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func unmarkedSumArray(nums []int, queries [][]int) []int64 {

}
```

**Kotlin Solution:**

```
class Solution {
fun unmarkedSumArray(nums: IntArray, queries: Array<IntArray>): LongArray {

}
}
```

**Swift Solution:**

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

}
}
```

### Rust Solution:

```
// Problem: Mark Elements on Array by Performing Queries
// Difficulty: Medium
// Tags: array, hash, sort, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn unmarked_sum_array(nums: Vec<i32>, queries: Vec<Vec<i32>>) -> Vec<i64>
{

}
}
```

### Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[][]} queries
# @return {Integer[]}
def unmarked_sum_array(nums, queries)

end
```

### PHP Solution:

```
class Solution {

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

```
}
}
```

### Dart Solution:

```
class Solution {
List<int> unmarkedSumArray(List<int> nums, List<List<int>> queries) {

}
}
```

### Scala Solution:

```
object Solution {
def unmarkedSumArray(nums: Array[Int], queries: Array[Array[Int]]):
Array[Long] = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec unmarked_sum_array(nums :: [integer], queries :: [[integer]]) ::
[integer]
def unmarked_sum_array(nums, queries) do

end
end
```

### Erlang Solution:

```
-spec unmarked_sum_array(Nums :: [integer()], Queries :: [[integer()]]) ->
[integer()].
unmarked_sum_array(Nums, Queries) ->
.
```

### Racket Solution:

```
(define/contract (unmarked-sum-array nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```