# Problem 3300: Minimum Element After Replacement With Digit Sum

## Problem Information

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

## Problem Description

You are given an integer array

nums

.

You replace each element in

nums

with the

sum

of its digits.

Return the

minimum

element in

nums

after all replacements.

Example 1:

Input:

nums = [10,12,13,14]

Output:

1

Explanation:

nums

becomes

[1, 3, 4, 5]

after all replacements, with minimum element 1.

Example 2:

Input:

nums = [1,2,3,4]

Output:

1

Explanation:

nums

becomes

[1, 2, 3, 4]

after all replacements, with minimum element 1.

Example 3:

Input:

nums = [999,19,199]

Output:

10

Explanation:

nums

becomes

[27, 10, 19]

after all replacements, with minimum element 10.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 10

4

## Code Snippets

**C++:**

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

```
}
};
```

**Java:**

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

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

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

};
```

**C#:**

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

}
}
```

**C:**

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

}
```

**Go:**

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

}
```

**Kotlin:**

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

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

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

}
}
```

**Dart:**

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

}
}
```

**Scala:**

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

}
}
```

**Elixir:**

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

end
end
```

### Erlang:

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

### Racket:

```
(define/contract (min-element nums)
(-> (listof exact-integer?) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Minimum Element After Replacement With Digit Sum
* Difficulty: Easy
* Tags: array, math
*
* 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:
int minElement(vector<int>& nums) {

}
};
```

### Java Solution:

```
/**
* Problem: Minimum Element After Replacement With Digit Sum
* Difficulty: Easy
* Tags: array, math
*
* 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 int minElement(int[] nums) {

}
}
```

### Python3 Solution:

```
"""
Problem: Minimum Element After Replacement With Digit Sum
Difficulty: Easy
Tags: array, math

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

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Minimum Element After Replacement With Digit Sum
* Difficulty: Easy
```

```
* Tags: array, math
*
* 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[]} nums
* @return {number}
*/
var minElement = function(nums) {

};
```

### TypeScript Solution:

```
/**
* Problem: Minimum Element After Replacement With Digit Sum
* Difficulty: Easy
* Tags: array, math
*
* 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 minElement(nums: number[]): number {

};
```

### C# Solution:

```
/*
* Problem: Minimum Element After Replacement With Digit Sum
* Difficulty: Easy
* Tags: array, math
*
* 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 int MinElement(int[] nums) {

}
}
```

### C Solution:

```
/*
* Problem: Minimum Element After Replacement With Digit Sum
* Difficulty: Easy
* Tags: array, math
*
* 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
*/

int minElement(int* nums, int numsSize) {

}
```

### Go Solution:

```
// Problem: Minimum Element After Replacement With Digit Sum
// Difficulty: Easy
// Tags: array, math
//
// 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 minElement(nums []int) int {

}
```

### Kotlin Solution:

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

}
}
```

### Swift Solution:

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

}
}
```

### Rust Solution:

```
// Problem: Minimum Element After Replacement With Digit Sum
// Difficulty: Easy
// Tags: array, math
//
// 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 min_element(nums: Vec<i32>) -> i32 {

}
}
```

### Ruby Solution:

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

end
```

### PHP Solution:

```
class Solution {
```

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

}
}
```

**Dart Solution:**

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

}
}
```

**Scala Solution:**

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

}
}
```

**Elixir Solution:**

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

end
end
```

**Erlang Solution:**

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

**Racket Solution:**

```
(define/contract (min-element nums)
(-> (listof exact-integer?) exact-integer?)
)
```