Problem 2141: Maximum Running Time of N Computers

Problem Information

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

Problem Description

You have

n

computers. You are given the integer

n

and a

0-indexed

integer array

batteries

where the

i

th

battery can

run

a computer for

batteries[i]

minutes. You are interested in running

all

n

computers

simultaneously

using the given batteries.

Initially, you can insert

at most one battery

into each computer. After that and at any integer time moment, you can remove a battery from a computer and insert another battery

any number of times

. The inserted battery can be a totally new battery or a battery from another computer. You may assume that the removing and inserting processes take no time.

Note that the batteries cannot be recharged.

Return

the

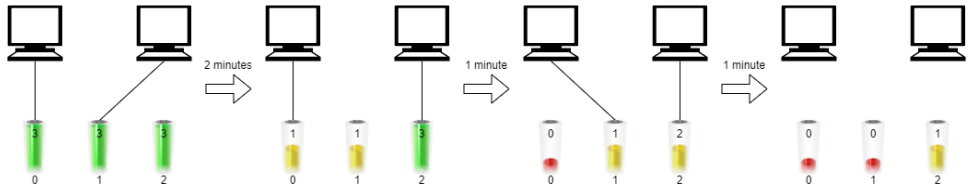
maximum

number of minutes you can run all the

n

computers simultaneously.

Example 1:



Input:

n = 2, batteries = [3,3,3]

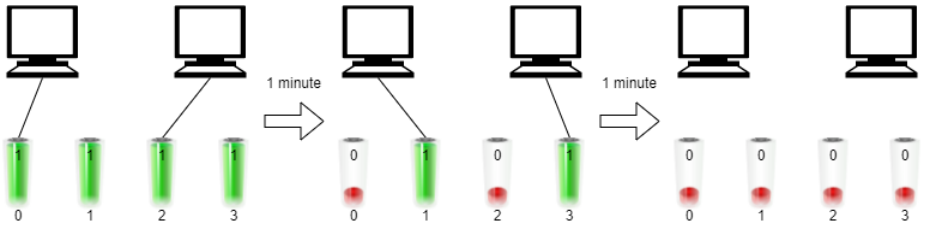
Output:

4

Explanation:

Initially, insert battery 0 into the first computer and battery 1 into the second computer. After two minutes, remove battery 1 from the second computer and insert battery 2 instead. Note that battery 1 can still run for one minute. At the end of the third minute, battery 0 is drained, and you need to remove it from the first computer and insert battery 1 instead. By the end of the fourth minute, battery 1 is also drained, and the first computer is no longer running. We can run the two computers simultaneously for at most 4 minutes, so we return 4.

Example 2:



Input:

n = 2, batteries = [1,1,1,1]

Output:

2

Explanation:

Initially, insert battery 0 into the first computer and battery 2 into the second computer. After one minute, battery 0 and battery 2 are drained so you need to remove them and insert battery 1 into the first computer and battery 3 into the second computer. After another minute, battery 1 and battery 3 are also drained so the first and second computers are no longer running. We can run the two computers simultaneously for at most 2 minutes, so we return 2.

Constraints:

1 <= n <= batteries.length <= 10

5

1 <= batteries[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
long long maxRunTime(int n, vector<int>& batteries) {

}
};
```

Java:

```
class Solution {
public long maxRunTime(int n, int[] batteries) {

}
}
```

Python3:

```
class Solution:
def maxRunTime(self, n: int, batteries: List[int]) -> int:
```

Python:

```
class Solution(object):
def maxRunTime(self, n, batteries):
"""
:type n: int
:type batteries: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[]} batteries
* @return {number}
*/
var maxRunTime = function(n, batteries) {

};
```

TypeScript:

```
function maxRunTime(n: number, batteries: number[]): number {

};
```

C#:

```
public class Solution {
public long MaxRunTime(int n, int[] batteries) {

}
}
```

C:

```
long long maxRunTime(int n, int* batteries, int batteriesSize) {

}
```

Go:

```
func maxRunTime(n int, batteries []int) int64 {

}
```

Kotlin:

```
class Solution {
fun maxRunTime(n: Int, batteries: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func maxRunTime(_ n: Int, _ batteries: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_run_time(n: i32, batteries: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[]} batteries
# @return {Integer}
def max_run_time(n, batteries)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $batteries
* @return Integer
*/
function maxRunTime($n, $batteries) {

}
}
```

Dart:

```
class Solution {
int maxRunTime(int n, List<int> batteries) {

}
}
```

Scala:

```
object Solution {
def maxRunTime(n: Int, batteries: Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_run_time(n :: integer, batteries :: [integer]) :: integer
def max_run_time(n, batteries) do

end
end
```

Erlang:

```
-spec max_run_time(N :: integer(), Batteries :: [integer()]) -> integer().
max_run_time(N, Batteries) ->
```

```
.
```

Racket:

```
(define/contract (max-run-time n batteries)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Running Time of N Computers
* Difficulty: Hard
* Tags: array, greedy, sort, search
*
* 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 maxRunTime(int n, vector<int>& batteries) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Running Time of N Computers
* Difficulty: Hard
* Tags: array, greedy, sort, search
*
* 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 maxRunTime(int n, int[] batteries) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Running Time of N Computers
Difficulty: Hard
Tags: array, greedy, sort, search

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 maxRunTime(self, n: int, batteries: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxRunTime(self, n, batteries):
"""
:type n: int
:type batteries: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Running Time of N Computers
* Difficulty: Hard
* Tags: array, greedy, sort, search
*
```

```
* 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} n
* @param {number[]} batteries
* @return {number}
*/
var maxRunTime = function(n, batteries) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Running Time of N Computers
* Difficulty: Hard
* Tags: array, greedy, sort, search
*
* 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 maxRunTime(n: number, batteries: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Running Time of N Computers
* Difficulty: Hard
* Tags: array, greedy, sort, search
*
* 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 MaxRunTime(int n, int[] batteries) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Running Time of N Computers
* Difficulty: Hard
* Tags: array, greedy, sort, search
*
* 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 maxRunTime(int n, int* batteries, int batteriesSize) {

}
```

Go Solution:

```
// Problem: Maximum Running Time of N Computers
// Difficulty: Hard
// Tags: array, greedy, sort, search
//
// 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 maxRunTime(n int, batteries []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxRunTime(n: Int, batteries: IntArray): Long {
```

```
}
}
```

Swift Solution:

```
class Solution {
func maxRunTime(_ n: Int, _ batteries: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Running Time of N Computers
// Difficulty: Hard
// Tags: array, greedy, sort, search
//
// 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_run_time(n: i32, batteries: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[]} batteries
# @return {Integer}
def max_run_time(n, batteries)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer $n
* @param Integer[] $batteries
* @return Integer
*/
function maxRunTime($n, $batteries) {

}
}
```

Dart Solution:

```
class Solution {
int maxRunTime(int n, List<int> batteries) {

}
}
```

Scala Solution:

```
object Solution {
def maxRunTime(n: Int, batteries: Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_run_time(n :: integer, batteries :: [integer]) :: integer
def max_run_time(n, batteries) do

end
end
```

Erlang Solution:

```
-spec max_run_time(N :: integer(), Batteries :: [integer()]) -> integer().
max_run_time(N, Batteries) ->
.
```

Racket Solution:

```
(define/contract (max-run-time n batteries)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```