Problem 3259: Maximum Energy Boost From Two Drinks

Problem Information

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

Problem Description

You are given two integer arrays

energyDrinkA

and

energyDrinkB

of the same length

n

by a futuristic sports scientist. These arrays represent the energy boosts per hour provided by two different energy drinks, A and B, respectively.

You want to

maximize

your total energy boost by drinking one energy drink

per hour

. However, if you want to switch from consuming one energy drink to the other, you need to wait for one hour

to cleanse your system (meaning you won't get any energy boost in that hour).

Return the

maximum

total energy boost you can gain in the next

n

hours.

Note

that you can start consuming

either

of the two energy drinks.

Example 1:

Input:

energyDrinkA

= [1,3,1],

energyDrinkB

= [3,1,1]

Output:

5

Explanation:

To gain an energy boost of 5, drink only the energy drink A (or only B).

Example 2:

Input:

energyDrinkA

= [4,1,1],

energyDrinkB

= [1,1,3]

Output:

7

Explanation:

To gain an energy boost of 7:

Drink the energy drink A for the first hour.

Switch to the energy drink B and we lose the energy boost of the second hour.

Gain the energy boost of the drink B in the third hour.

Constraints:

n == energyDrinkA.length == energyDrinkB.length

3 <= n <= 10

5

1 <= energyDrinkA[i], energyDrinkB[i] <= 10

Code Snippets

C++:

```
class Solution {
public:
long long maxEnergyBoost(vector<int>& energyDrinkA, vector<int>&
energyDrinkB) {

}
};
```

Java:

```
class Solution {
public long maxEnergyBoost(int[] energyDrinkA, int[] energyDrinkB) {

}
}
```

Python3:

```
class Solution:
def maxEnergyBoost(self, energyDrinkA: List[int], energyDrinkB: List[int]) ->
int:
```

Python:

```
class Solution(object):
def maxEnergyBoost(self, energyDrinkA, energyDrinkB):
"""
:type energyDrinkA: List[int]
:type energyDrinkB: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} energyDrinkA
* @param {number[]} energyDrinkB
* @return {number}
*/
var maxEnergyBoost = function(energyDrinkA, energyDrinkB) {

};
```

TypeScript:

```
function maxEnergyBoost(energyDrinkA: number[], energyDrinkB: number[]):
number {

};
```

C#:

```
public class Solution {
public long MaxEnergyBoost(int[] energyDrinkA, int[] energyDrinkB) {

}
}
```

C:

```
long long maxEnergyBoost(int* energyDrinkA, int energyDrinkASize, int*
energyDrinkB, int energyDrinkBSize) {

}
```

Go:

```
func maxEnergyBoost(energyDrinkA []int, energyDrinkB []int) int64 {

}
```

Kotlin:

```
class Solution {
fun maxEnergyBoost(energyDrinkA: IntArray, energyDrinkB: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func maxEnergyBoost(_ energyDrinkA: [Int], _ energyDrinkB: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_energy_boost(energy_drink_a: Vec<i32>, energy_drink_b: Vec<i32>)
-> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} energy_drink_a
# @param {Integer[]} energy_drink_b
# @return {Integer}
def max_energy_boost(energy_drink_a, energy_drink_b)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $energyDrinkA
* @param Integer[] $energyDrinkB
* @return Integer
*/
function maxEnergyBoost($energyDrinkA, $energyDrinkB) {

}
}
```

Dart:

```
class Solution {
int maxEnergyBoost(List<int> energyDrinkA, List<int> energyDrinkB) {

}
}
```

Scala:

```
object Solution {
def maxEnergyBoost(energyDrinkA: Array[Int], energyDrinkB: Array[Int]): Long
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_energy_boost(energy_drink_a :: [integer], energy_drink_b ::
[integer]) :: integer
def max_energy_boost(energy_drink_a, energy_drink_b) do

end
end
```

Erlang:

```
-spec max_energy_boost(EnergyDrinkA :: [integer()], EnergyDrinkB ::
[integer()]) -> integer().
max_energy_boost(EnergyDrinkA, EnergyDrinkB) ->
.
```

Racket:

```
(define/contract (max-energy-boost energyDrinkA energyDrinkB)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Energy Boost From Two Drinks
* Difficulty: Medium
* Tags: array, dp
*
* 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:
long long maxEnergyBoost(vector<int>& energyDrinkA, vector<int>&
energyDrinkB) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Energy Boost From Two Drinks
* Difficulty: Medium
* Tags: array, dp
*
* 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 long maxEnergyBoost(int[] energyDrinkA, int[] energyDrinkB) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Energy Boost From Two Drinks
Difficulty: Medium
```

```
Tags: array, dp

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 maxEnergyBoost(self, energyDrinkA: List[int], energyDrinkB: List[int]) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxEnergyBoost(self, energyDrinkA, energyDrinkB):
"""
:type energyDrinkA: List[int]
:type energyDrinkB: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Energy Boost From Two Drinks
* Difficulty: Medium
* Tags: array, dp
*
* 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[]} energyDrinkA
* @param {number[]} energyDrinkB
* @return {number}
*/
var maxEnergyBoost = function(energyDrinkA, energyDrinkB) {
};
```

TypeScript Solution:

```
/**
* Problem: Maximum Energy Boost From Two Drinks
* Difficulty: Medium
* Tags: array, dp
*
* 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 maxEnergyBoost(energyDrinkA: number[], energyDrinkB: number[]):
number {

};
```

C# Solution:

```
/*
* Problem: Maximum Energy Boost From Two Drinks
* Difficulty: Medium
* Tags: array, dp
*
* 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 long MaxEnergyBoost(int[] energyDrinkA, int[] energyDrinkB) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Energy Boost From Two Drinks
* Difficulty: Medium
* Tags: array, dp
*
* 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
*/

long long maxEnergyBoost(int* energyDrinkA, int energyDrinkASize, int*
energyDrinkB, int energyDrinkBSize) {

}
```

Go Solution:

```
// Problem: Maximum Energy Boost From Two Drinks
// Difficulty: Medium
// Tags: array, dp
//
// 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 maxEnergyBoost(energyDrinkA []int, energyDrinkB []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxEnergyBoost(energyDrinkA: IntArray, energyDrinkB: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func maxEnergyBoost(_ energyDrinkA: [Int], _ energyDrinkB: [Int]) -> Int {
}
}
```

Rust Solution:

```
// Problem: Maximum Energy Boost From Two Drinks
// Difficulty: Medium
// Tags: array, dp
//
// 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 max_energy_boost(energy_drink_a: Vec<i32>, energy_drink_b: Vec<i32>)
-> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} energy_drink_a
# @param {Integer[]} energy_drink_b
# @return {Integer}
def max_energy_boost(energy_drink_a, energy_drink_b)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $energyDrinkA
* @param Integer[] $energyDrinkB
* @return Integer
*/
function maxEnergyBoost($energyDrinkA, $energyDrinkB) {

}
}
```

Dart Solution:

```
class Solution {
int maxEnergyBoost(List<int> energyDrinkA, List<int> energyDrinkB) {

}
}
```

Scala Solution:

```
object Solution {
def maxEnergyBoost(energyDrinkA: Array[Int], energyDrinkB: Array[Int]): Long
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_energy_boost(energy_drink_a :: [integer], energy_drink_b ::
[integer]) :: integer
def max_energy_boost(energy_drink_a, energy_drink_b) do

end
end
```

Erlang Solution:

```
-spec max_energy_boost(EnergyDrinkA :: [integer()], EnergyDrinkB ::
[integer()]) -> integer().
max_energy_boost(EnergyDrinkA, EnergyDrinkB) ->
.
```

Racket Solution:

```
(define/contract (max-energy-boost energyDrinkA energyDrinkB)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```