

Problem 3457: Eat Pizzas!

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given an integer array

pizzas

of size

n

, where

$\text{pizzas}[i]$

represents the weight of the

i

th

pizza. Every day, you eat

exactly

4 pizzas. Due to your incredible metabolism, when you eat pizzas of weights

w

,

X

,

Y

, and

Z

, where

$W \leq X \leq Y \leq Z$

, you gain the weight of only 1 pizza!

On

odd-numbered

days

(1-indexed)

, you gain a weight of

Z

.

On

even-numbered

days, you gain a weight of

Y

Find the maximum total weight you can gain by eating all pizzas optimally.

Note

: It is guaranteed that

n

is a multiple of 4, and each pizza can be eaten only once.

Example 1:

Input:

pizzas = [1,2,3,4,5,6,7,8]

Output:

14

Explanation:

On day 1, you eat pizzas at indices

[1, 2, 4, 7] = [2, 3, 5, 8]

. You gain a weight of 8.

On day 2, you eat pizzas at indices

$$[0, 3, 5, 6] = [1, 4, 6, 7]$$

. You gain a weight of 6.

The total weight gained after eating all the pizzas is

$$8 + 6 = 14$$

Example 2:

Input:

$$\text{pizzas} = [2, 1, 1, 1, 1, 1, 1, 1]$$

Output:

3

Explanation:

On day 1, you eat pizzas at indices

$$[4, 5, 6, 0] = [1, 1, 1, 2]$$

. You gain a weight of 2.

On day 2, you eat pizzas at indices

$$[1, 2, 3, 7] = [1, 1, 1, 1]$$

. You gain a weight of 1.

The total weight gained after eating all the pizzas is

$$2 + 1 = 3.$$

Constraints:

$4 \leq n == \text{pizzas.length} \leq 2 * 10$

5

$1 \leq \text{pizzas}[i] \leq 10$

5

n

is a multiple of 4.

Code Snippets

C++:

```
class Solution {
public:
    long long maxWeight(vector<int>& pizzas) {
        }
    };
}
```

Java:

```
class Solution {
public long maxWeight(int[] pizzas) {
        }
    }
}
```

Python3:

```
class Solution:
    def maxWeight(self, pizzas: List[int]) -> int:
```

Python:

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

JavaScript:

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

TypeScript:

```
function maxWeight(pizzas: number[]): number {
}
```

C#:

```
public class Solution {
    public long MaxWeight(int[] pizzas) {
}
```

C:

```
long long maxWeight(int* pizzas, int pizzasSize) {
}
```

Go:

```
func maxWeight(pizzas []int) int64 {
}
```

Kotlin:

```
class Solution {  
    fun maxWeight(pizzas: IntArray): Long {  
  
    }  
}
```

Swift:

```
class Solution {  
    func maxWeight(_ pizzas: [Int]) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn max_weight(pizzas: Vec<i32>) -> i64 {  
  
    }  
}
```

Ruby:

```
# @param {Integer[]} pizzas  
# @return {Integer}  
def max_weight(pizzas)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $pizzas  
     * @return Integer  
     */  
    function maxWeight($pizzas) {  
  
    }
```

```
}
```

Dart:

```
class Solution {  
    int maxWeight(List<int> pizzas) {  
  
    }  
}
```

Scala:

```
object Solution {  
    def maxWeight(pizzas: Array[Int]): Long = {  
  
    }  
}
```

Elixir:

```
defmodule Solution do  
  @spec max_weight(list :: [integer]) :: integer  
  def max_weight(list) do  
  
  end  
end
```

Erlang:

```
-spec max_weight(list :: [integer()]) -> integer().  
max_weight(list) ->  
.
```

Racket:

```
(define/contract (max-weight list)  
  (-> (listof exact-integer?) exact-integer?)  
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Eat Pizzas!
 * Difficulty: Medium
 * Tags: array, greedy, sort
 *
 * 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 maxWeight(vector<int>& pizzas) {
}
```

Java Solution:

```
/**
 * Problem: Eat Pizzas!
 * Difficulty: Medium
 * Tags: array, greedy, sort
 *
 * 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 maxWeight(int[] pizzas) {
}
```

Python3 Solution:

```
"""
Problem: Eat Pizzas!
Difficulty: Medium
Tags: array, greedy, sort
```

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

Python Solution:

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

```

JavaScript Solution:

```
/**
 * Problem: Eat Pizzas!
 * Difficulty: Medium
 * Tags: array, greedy, sort
 *
 * 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[]} pizzas
 * @return {number}
 */
var maxWeight = function(pizzas) {

};
```

TypeScript Solution:

```

/**
 * Problem: Eat Pizzas!
 * Difficulty: Medium
 * Tags: array, greedy, sort
 *
 * 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 maxWeight(pizzas: number[]): number {
}

```

C# Solution:

```

/*
 * Problem: Eat Pizzas!
 * Difficulty: Medium
 * Tags: array, greedy, sort
 *
 * 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 MaxWeight(int[] pizzas) {
}
}

```

C Solution:

```

/*
 * Problem: Eat Pizzas!
 * Difficulty: Medium
 * Tags: array, greedy, sort
 *
 * 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 maxWeight(int* pizzas, int pizzasSize) {  
  
}  

```

Go Solution:

```
// Problem: Eat Pizzas!  
// Difficulty: Medium  
// Tags: array, greedy, sort  
//  
// 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 maxWeight(pizzas []int) int64 {  
  
}
```

Kotlin Solution:

```
class Solution {  
    fun maxWeight(pizzas: IntArray): Long {  
  
    }  
}
```

Swift Solution:

```
class Solution {  
    func maxWeight(_ pizzas: [Int]) -> Int {  
  
    }  
}
```

Rust Solution:

```
// Problem: Eat Pizzas!  
// Difficulty: Medium  
// Tags: array, greedy, sort
```

```

// 
// 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_weight(pizzas: Vec<i32>) -> i64 {

}
}

```

Ruby Solution:

```

# @param {Integer[]} pizzas
# @return {Integer}
def max_weight(pizzas)

end

```

PHP Solution:

```

class Solution {

/**
 * @param Integer[] $pizzas
 * @return Integer
 */
function maxWeight($pizzas) {

}
}

```

Dart Solution:

```

class Solution {
int maxWeight(List<int> pizzas) {

}
}

```

Scala Solution:

```
object Solution {  
    def maxWeight(pizzas: Array[Int]): Long = {  
        }  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec max_weight(list :: [integer]) :: integer  
  def max_weight(list) do  
  
  end  
end
```

Erlang Solution:

```
-spec max_weight(list :: [integer()]) -> integer().  
max_weight(list) ->  
.
```

Racket Solution:

```
(define/contract (max-weight pizzas)  
  (-> (listof exact-integer?) exact-integer?)  
)
```