

Problem 2144: Minimum Cost of Buying Candies With Discount

Problem Information

Difficulty: **Easy**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

A shop is selling candies at a discount. For

every two

candies sold, the shop gives a

third

candy for

free

.

The customer can choose

any

candy to take away for free as long as the cost of the chosen candy is less than or equal to the

minimum

cost of the two candies bought.

For example, if there are

4

candies with costs

1

,

2

,

3

, and

4

, and the customer buys candies with costs

2

and

3

, they can take the candy with cost

1

for free, but not the candy with cost

4

. Given a

0-indexed

integer array

cost

, where

$\text{cost}[i]$

denotes the cost of the

i

th

candy, return

the

minimum cost

of buying

all

the candies

.

Example 1:

Input:

$\text{cost} = [1, 2, 3]$

Output:

Explanation:

We buy the candies with costs 2 and 3, and take the candy with cost 1 for free. The total cost of buying all candies is $2 + 3 = 5$. This is the

only

way we can buy the candies. Note that we cannot buy candies with costs 1 and 3, and then take the candy with cost 2 for free. The cost of the free candy has to be less than or equal to the minimum cost of the purchased candies.

Example 2:

Input:

cost = [6,5,7,9,2,2]

Output:

23

Explanation:

The way in which we can get the minimum cost is described below: - Buy candies with costs 9 and 7 - Take the candy with cost 6 for free - We buy candies with costs 5 and 2 - Take the last remaining candy with cost 2 for free Hence, the minimum cost to buy all candies is $9 + 7 + 5 + 2 = 23$.

Example 3:

Input:

cost = [5,5]

Output:

10

Explanation:

Since there are only 2 candies, we buy both of them. There is not a third candy we can take for free. Hence, the minimum cost to buy all candies is $5 + 5 = 10$.

Constraints:

$1 \leq \text{cost.length} \leq 100$

$1 \leq \text{cost}[i] \leq 100$

Code Snippets

C++:

```
class Solution {  
public:  
    int minimumCost(vector<int>& cost) {  
        }  
    };
```

Java:

```
class Solution {  
public int minimumCost(int[] cost) {  
    }  
}
```

Python3:

```
class Solution:  
    def minimumCost(self, cost: List[int]) -> int:
```

Python:

```
class Solution(object):  
    def minimumCost(self, cost):  
        """
```

```
:type cost: List[int]
:rtype: int
"""

```

JavaScript:

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

};


```

TypeScript:

```
function minimumCost(cost: number[]): number {

};


```

C#:

```
public class Solution {
    public int MinimumCost(int[] cost) {

    }
}
```

C:

```
int minimumCost(int* cost, int costSize) {

}
```

Go:

```
func minimumCost(cost []int) int {

}
```

Kotlin:

```
class Solution {  
    fun minimumCost(cost: IntArray): Int {  
        }  
        }  
    }
```

Swift:

```
class Solution {  
    func minimumCost(_ cost: [Int]) -> Int {  
        }  
        }  
    }
```

Rust:

```
impl Solution {  
    pub fn minimum_cost(cost: Vec<i32>) -> i32 {  
        }  
        }  
    }
```

Ruby:

```
# @param {Integer[]} cost  
# @return {Integer}  
def minimum_cost(cost)  
  
end
```

PHP:

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

Dart:

```
class Solution {  
    int minimumCost(List<int> cost) {  
  
    }  
}
```

Scala:

```
object Solution {  
    def minimumCost(cost: Array[Int]): Int = {  
  
    }  
}
```

Elixir:

```
defmodule Solution do  
    @spec minimum_cost(cost :: [integer]) :: integer  
    def minimum_cost(cost) do  
  
    end  
end
```

Erlang:

```
-spec minimum_cost([integer()]) -> integer().  
minimum_cost([_]) ->  
.
```

Racket:

```
(define/contract (minimum-cost cost)  
  (-> (listof exact-integer?) exact-integer?)  
)
```

Solutions

C++ Solution:

```

/*
 * Problem: Minimum Cost of Buying Candies With Discount
 * Difficulty: Easy
 * 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:
    int minimumCost(vector<int>& cost) {

    }
};

```

Java Solution:

```

/**
 * Problem: Minimum Cost of Buying Candies With Discount
 * Difficulty: Easy
 * 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 int minimumCost(int[] cost) {

}
}

```

Python3 Solution:

```

"""
Problem: Minimum Cost of Buying Candies With Discount
Difficulty: Easy
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 minimumCost(self, cost: List[int]) -> int:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

class Solution(object):
def minimumCost(self, cost):
"""
:type cost: List[int]
:rtype: int
"""

```

JavaScript Solution:

```

/**
 * Problem: Minimum Cost of Buying Candies With Discount
 * Difficulty: Easy
 * 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[]} cost
 * @return {number}
 */
var minimumCost = function(cost) {

};


```

TypeScript Solution:

```

/**
 * Problem: Minimum Cost of Buying Candies With Discount
 * Difficulty: Easy
 * 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 minimumCost(cost: number[]): number {
}

```

C# Solution:

```

/*
 * Problem: Minimum Cost of Buying Candies With Discount
 * Difficulty: Easy
 * 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 int MinimumCost(int[] cost) {
        return 0;
    }
}

```

C Solution:

```

/*
 * Problem: Minimum Cost of Buying Candies With Discount
 * Difficulty: Easy
 * 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
 */

```

```
*/  
  
int minimumCost(int* cost, int costSize) {  
  
}  

```

Go Solution:

```
// Problem: Minimum Cost of Buying Candies With Discount  
// Difficulty: Easy  
// 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 minimumCost(cost []int) int {  
  
}
```

Kotlin Solution:

```
class Solution {  
    fun minimumCost(cost: IntArray): Int {  
  
    }  
}
```

Swift Solution:

```
class Solution {  
    func minimumCost(_ cost: [Int]) -> Int {  
  
    }  
}
```

Rust Solution:

```
// Problem: Minimum Cost of Buying Candies With Discount  
// Difficulty: Easy  
// 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 minimum_cost(cost: Vec<i32>) -> i32 {
        }

    }
}

```

Ruby Solution:

```

# @param {Integer[]} cost
# @return {Integer}
def minimum_cost(cost)

end

```

PHP Solution:

```

class Solution {

    /**
     * @param Integer[] $cost
     * @return Integer
     */
    function minimumCost($cost) {

    }
}

```

Dart Solution:

```

class Solution {
    int minimumCost(List<int> cost) {
        }

    }
}

```

Scala Solution:

```
object Solution {  
    def minimumCost(cost: Array[Int]): Int = {  
        }  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec minimum_cost(cost :: [integer]) :: integer  
  def minimum_cost(cost) do  
  
  end  
end
```

Erlang Solution:

```
-spec minimum_cost([integer()]) -> integer().  
minimum_cost([Cost] ->  
  .
```

Racket Solution:

```
(define/contract (minimum-cost cost)  
  (-> (listof exact-integer?) exact-integer?)  
  )
```