Problem 1167: Minimum Cost to Connect Sticks

Problem Information

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

Problem Description

You have some number of sticks with positive integer lengths. These lengths are given as an array

sticks

, where

sticks[i]

is the length of the

i

th

stick.

You can connect any two sticks of lengths

x

and

y

into one stick by paying a cost of

x + y

. You must connect all the sticks until there is only one stick remaining.

Return

the minimum cost of connecting all the given sticks into one stick in this way

.

Example 1:

Input:

sticks = [2,4,3]

Output:

14

Explanation:

You start with sticks = [2,4,3]. 1. Combine sticks 2 and 3 for a cost of 2 + 3 = 5. Now you have sticks = [5,4]. 2. Combine sticks 5 and 4 for a cost of 5 + 4 = 9. Now you have sticks = [9]. There is only one stick left, so you are done. The total cost is 5 + 9 = 14.

Example 2:

Input:

sticks = [1,8,3,5]

Output:

30

Explanation:

You start with sticks = [1,8,3,5]. 1. Combine sticks 1 and 3 for a cost of 1 + 3 = 4. Now you have sticks = [4,8,5]. 2. Combine sticks 4 and 5 for a cost of 4 + 5 = 9. Now you have sticks = [9,8]. 3. Combine sticks 9 and 8 for a cost of 9 + 8 = 17. Now you have sticks = [17]. There is only one stick left, so you are done. The total cost is 4 + 9 + 17 = 30.

Example 3:

Input:

sticks = [5]

Output:

0

Explanation:

There is only one stick, so you don't need to do anything. The total cost is 0.

Constraints:

1 <= sticks.length <= 10

4

1 <= sticks[i] <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int connectSticks(vector<int>& sticks) {

}
};
```

Java:

```
class Solution {
public int connectSticks(int[] sticks) {

}
}
```

Python3:

```
class Solution:
def connectSticks(self, sticks: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function connectSticks(sticks: number[]): number {

};
```

C#:

```
public class Solution {
public int ConnectSticks(int[] sticks) {
```

```
}
}
```

C:

```
int connectSticks(int* sticks, int sticksSize) {

}
```

Go:

```
func connectSticks(sticks []int) int {

}
```

Kotlin:

```
class Solution {
fun connectSticks(sticks: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func connectSticks(_ sticks: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn connect_sticks(sticks: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} sticks
# @return {Integer}
def connect_sticks(sticks)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $sticks
* @return Integer
*/
function connectSticks($sticks) {

}
}
```

Dart:

```
class Solution {
int connectSticks(List<int> sticks) {

}
}
```

Scala:

```
object Solution {
def connectSticks(sticks: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec connect_sticks(sticks :: [integer]) :: integer
def connect_sticks(sticks) do

end
end
```

Erlang:

```
-spec connect_sticks(Sticks :: [integer()]) -> integer().
connect_sticks(Sticks) ->
.
```

Racket:

```
(define/contract (connect-sticks sticks)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Cost to Connect Sticks
* Difficulty: Medium
* Tags: array, greedy, queue, heap
*
* 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 connectSticks(vector<int>& sticks) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Cost to Connect Sticks
* Difficulty: Medium
* Tags: array, greedy, queue, heap
*
* 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 connectSticks(int[] sticks) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Cost to Connect Sticks
Difficulty: Medium
Tags: array, greedy, queue, heap

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Cost to Connect Sticks
* Difficulty: Medium
```

```
* Tags: array, greedy, queue, heap
*
* 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[]} sticks
* @return {number}
*/
var connectSticks = function(sticks) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Cost to Connect Sticks
* Difficulty: Medium
* Tags: array, greedy, queue, heap
*
* 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 connectSticks(sticks: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Cost to Connect Sticks
* Difficulty: Medium
* Tags: array, greedy, queue, heap
*
* 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 ConnectSticks(int[] sticks) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Cost to Connect Sticks
* Difficulty: Medium
* Tags: array, greedy, queue, heap
*
* 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 connectSticks(int* sticks, int sticksSize) {

}
```

Go Solution:

```
// Problem: Minimum Cost to Connect Sticks
// Difficulty: Medium
// Tags: array, greedy, queue, heap
//
// 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 connectSticks(sticks []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun connectSticks(sticks: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func connectSticks(_ sticks: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Cost to Connect Sticks
// Difficulty: Medium
// Tags: array, greedy, queue, heap
//
// 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 connect_sticks(sticks: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} sticks
# @return {Integer}
def connect_sticks(sticks)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $sticks
* @return Integer
*/
function connectSticks($sticks) {

}
}
```

Dart Solution:

```
class Solution {
int connectSticks(List<int> sticks) {

}
}
```

Scala Solution:

```
object Solution {
def connectSticks(sticks: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec connect_sticks(sticks :: [integer]) :: integer
def connect_sticks(sticks) do

end
end
```

Erlang Solution:

```
-spec connect_sticks(Sticks :: [integer()]) -> integer().
connect_sticks(Sticks) ->
.
```

Racket Solution:

```
(define/contract (connect-sticks sticks)
(-> (listof exact-integer?) exact-integer?)
)
```