Problem 1547: Minimum Cost to Cut a Stick

Problem Information

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

Problem Description

Given a wooden stick of length

n

units. The stick is labelled from

0

to

n

. For example, a stick of length

6

is labelled as follows:

Given an integer array

cuts

where

cuts[i]

denotes a position you should perform a cut at.

You should perform the cuts in order, you can change the order of the cuts as you wish.

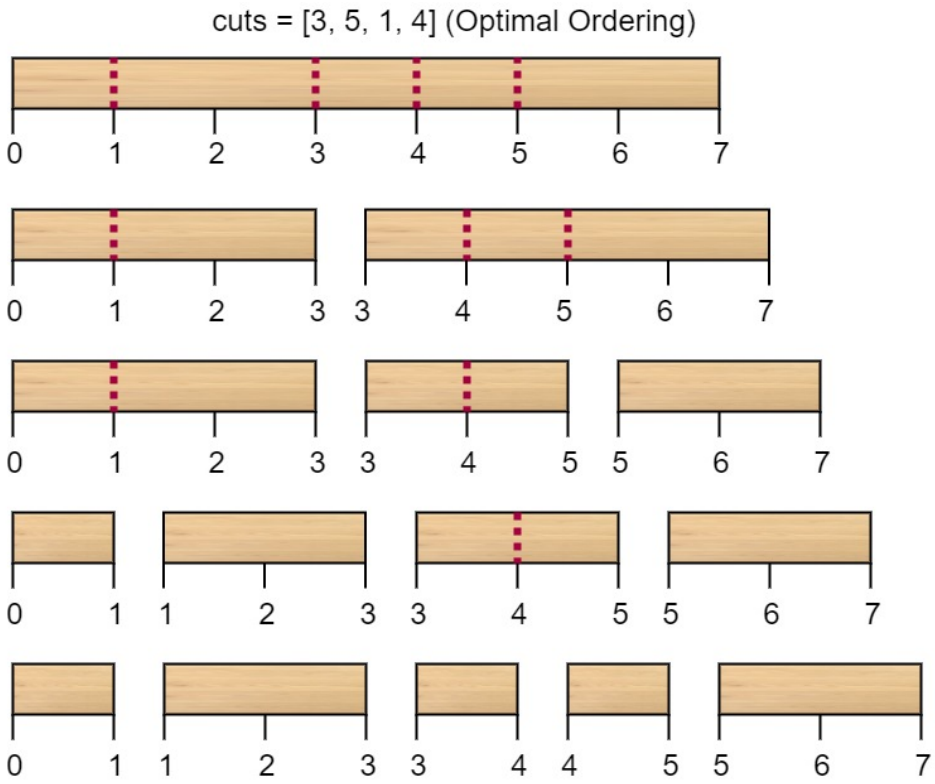
The cost of one cut is the length of the stick to be cut, the total cost is the sum of costs of all cuts. When you cut a stick, it will be split into two smaller sticks (i.e. the sum of their lengths is the length of the stick before the cut). Please refer to the first example for a better explanation.

Return

the minimum total cost

of the cuts.

Example 1:



Input:

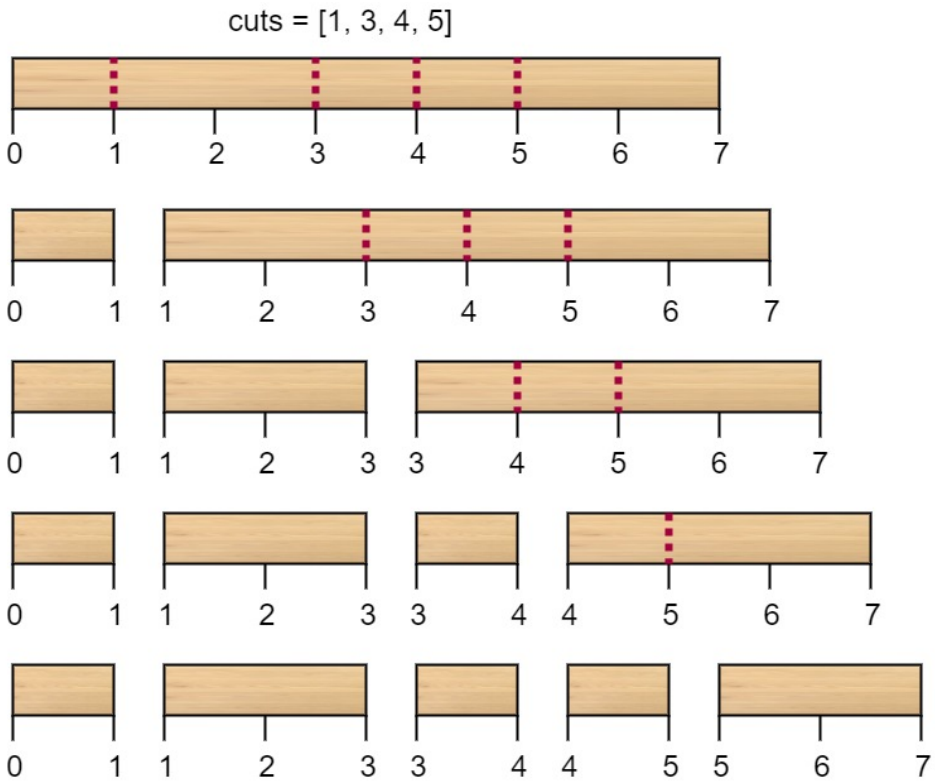
n = 7, cuts = [1,3,4,5]

Output:

16

Explanation:

Using cuts order = [1, 3, 4, 5] as in the input leads to the following scenario:



The first cut is done to a rod of length 7 so the cost is 7. The second cut is done to a rod of length 6 (i.e. the second part of the first cut), the third is done to a rod of length 4 and the last cut is to a rod of length 3. The total cost is 7 + 6 + 4 + 3 = 20. Rearranging the cuts to be [3, 5, 1, 4] for example will lead to a scenario with total cost = 16 (as shown in the example photo 7 + 4 + 3 + 2 = 16).

Example 2:

Input:

n = 9, cuts = [5,6,1,4,2]

Output:

22

Explanation:

If you try the given cuts ordering the cost will be 25. There are much ordering with total cost <= 25, for example, the order [4, 6, 5, 2, 1] has total cost = 22 which is the minimum possible.

Constraints:

2 <= n <= 10

6

1 <= cuts.length <= min(n - 1, 100)

1 <= cuts[i] <= n - 1

All the integers in

cuts

array are

distinct

.

Code Snippets

C++:

```
class Solution {
public:
int minCost(int n, vector<int>& cuts) {

}
```

```
};
```

Java:

```
class Solution {
public int minCost(int n, int[] cuts) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minCost(n: number, cuts: number[]): number {

};
```

C#:

```
public class Solution {
public int MinCost(int n, int[] cuts) {

}
}
```

C:

```
int minCost(int n, int* cuts, int cutsSize) {

}
```

Go:

```
func minCost(n int, cuts []int) int {

}
```

Kotlin:

```
class Solution {
fun minCost(n: Int, cuts: IntArray): Int {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn min_cost(n: i32, cuts: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[]} cuts
# @return {Integer}
def min_cost(n, cuts)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $cuts
* @return Integer
*/
function minCost($n, $cuts) {

}
}
```

Dart:

```
class Solution {
int minCost(int n, List<int> cuts) {

}
}
```

Scala:

```
object Solution {
def minCost(n: Int, cuts: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_cost(n :: integer, cuts :: [integer]) :: integer
```

```
def min_cost(n, cuts) do

end
end
```

Erlang:

```
-spec min_cost(N :: integer(), Cuts :: [integer()]) -> integer().
min_cost(N, Cuts) ->
.
```

Racket:

```
(define/contract (min-cost n cuts)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Cost to Cut a Stick
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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:
int minCost(int n, vector<int>& cuts) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Cost to Cut a Stick
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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 int minCost(int n, int[] cuts) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Cost to Cut a Stick
Difficulty: Hard
Tags: array, dp, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Cost to Cut a Stick
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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} n
* @param {number[]} cuts
* @return {number}
*/
var minCost = function(n, cuts) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Cost to Cut a Stick
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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 minCost(n: number, cuts: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Cost to Cut a Stick
* Difficulty: Hard
```

```
* Tags: array, dp, sort
*
* 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 int MinCost(int n, int[] cuts) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Cost to Cut a Stick
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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
*/

int minCost(int n, int* cuts, int cutsSize) {

}
```

Go Solution:

```
// Problem: Minimum Cost to Cut a Stick
// Difficulty: Hard
// Tags: array, dp, sort
//
// 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 minCost(n int, cuts []int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun minCost(n: Int, cuts: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Cost to Cut a Stick
// Difficulty: Hard
// Tags: array, dp, sort
//
// 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 min_cost(n: i32, cuts: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[]} cuts
# @return {Integer}
def min_cost(n, cuts)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $cuts
* @return Integer
*/
function minCost($n, $cuts) {

}
}
```

Dart Solution:

```
class Solution {
int minCost(int n, List<int> cuts) {

}
}
```

Scala Solution:

```
object Solution {
def minCost(n: Int, cuts: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_cost(n :: integer, cuts :: [integer]) :: integer
def min_cost(n, cuts) do

end
end
```

Erlang Solution:

```
-spec min_cost(N :: integer(), Cuts :: [integer()]) -> integer().
min_cost(N, Cuts) ->
.
```

Racket Solution:

```
(define/contract (min-cost n cuts)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```