Problem 339: Nested List Weight Sum

Problem Information

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

Problem Description

You are given a nested list of integers

nestedList

. Each element is either an integer or a list whose elements may also be integers or other lists.

The

depth

of an integer is the number of lists that it is inside of. For example, the nested list

[1,[2,2],[[3],2],1]

has each integer's value set to its

depth

.

Return

the sum of each integer in

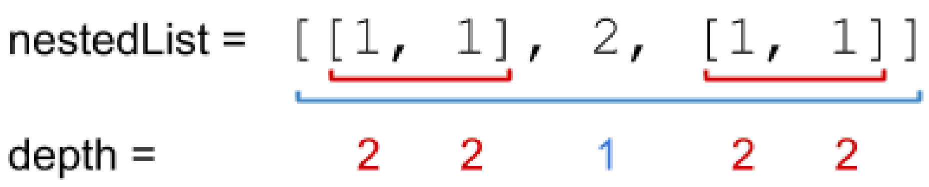
nestedList

multiplied by its

depth

.

Example 1:



Input:

nestedList = [[1,1],2,[1,1]]

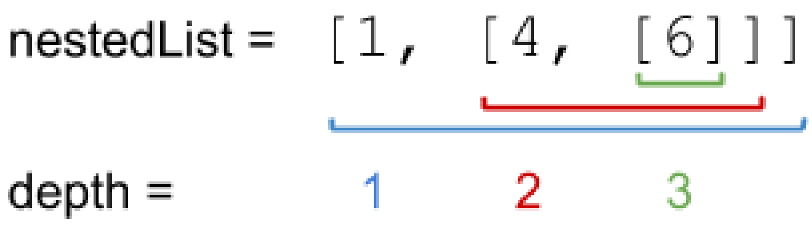
Output:

10

Explanation:

Four 1's at depth 2, one 2 at depth 1. 1*2 + 1*2 + 2*1 + 1*2 + 1*2 = 10.

Example 2:



Input:

nestedList = [1,[4,[6]]]

Output:

27

Explanation:

One 1 at depth 1, one 4 at depth 2, and one 6 at depth 3. 1*1 + 4*2 + 6*3 = 27.

Example 3:

Input:

nestedList = [0]

Output:

0

Constraints:

1 <= nestedList.length <= 50

The values of the integers in the nested list is in the range

[-100, 100]

.

The maximum

depth

of any integer is less than or equal to

50

.

Code Snippets

C++:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
* public:
* // Constructor initializes an empty nested list.
* NestedInteger();
*
* // Constructor initializes a single integer.
* NestedInteger(int value);
*
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* bool isInteger() const;
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* int getInteger() const;
*
* // Set this NestedInteger to hold a single integer.
* void setInteger(int value);
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* void add(const NestedInteger &ni);
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* const vector<NestedInteger> &getList() const;
* };
*/
class Solution {
public:
int depthSum(vector<NestedInteger>& nestedList) {

}
};
```

Java:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* public interface NestedInteger {
* // Constructor initializes an empty nested list.
* public NestedInteger();
*
* // Constructor initializes a single integer.
* public NestedInteger(int value);
*
* // @return true if this NestedInteger holds a single integer, rather than a
nested list.
* public boolean isInteger();
*
* // @return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* public Integer getInteger();
*
* // Set this NestedInteger to hold a single integer.
* public void setInteger(int value);
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* public void add(NestedInteger ni);
*
* // @return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* public List<NestedInteger> getList();
* }
*/
class Solution {
public int depthSum(List<NestedInteger> nestedList) {

}
}
```

Python3:

```
# """
# This is the interface that allows for creating nested lists.
# You should not implement it, or speculate about its implementation
# """
#class NestedInteger:
# def __init__(self, value=None):
# """
# If value is not specified, initializes an empty list.
# Otherwise initializes a single integer equal to value.
# """
#
# def isInteger(self):
# """
# @return True if this NestedInteger holds a single integer, rather than a
nested list.
# :rtype bool
# """
#
# def add(self, elem):
# """
# Set this NestedInteger to hold a nested list and adds a nested integer elem
to it.
# :rtype void
# """
#
# def setInteger(self, value):
# """
# Set this NestedInteger to hold a single integer equal to value.
# :rtype void
# """
#
# def getInteger(self):
# """
# @return the single integer that this NestedInteger holds, if it holds a
single integer
# The result is undefined if this NestedInteger holds a nested list
# :rtype int
# """
#
# def getList(self):
# """
# @return the nested list that this NestedInteger holds, if it holds a nested
list
```

```
# The result is undefined if this NestedInteger holds a single integer
# :rtype List[NestedInteger]
# """

class Solution:
def depthSum(self, nestedList: List[NestedInteger]) -> int:
```

Python:

```
# """
# This is the interface that allows for creating nested lists.
# You should not implement it, or speculate about its implementation
# """
#class NestedInteger(object):
# def __init__(self, value=None):
# """
# If value is not specified, initializes an empty list.
# Otherwise initializes a single integer equal to value.
# """
#
# def isInteger(self):
# """
# @return True if this NestedInteger holds a single integer, rather than a
nested list.
# :rtype bool
# """
#
# def add(self, elem):
# """
# Set this NestedInteger to hold a nested list and adds a nested integer elem
to it.
# :rtype void
# """
#
# def setInteger(self, value):
# """
# Set this NestedInteger to hold a single integer equal to value.
# :rtype void
# """
#
# def getInteger(self):
# """
# @return the single integer that this NestedInteger holds, if it holds a
single integer
# The result is undefined if this NestedInteger holds a nested list
# :rtype int
# """
#
# def getList(self):
# """
# @return the nested list that this NestedInteger holds, if it holds a nested
list
# The result is undefined if this NestedInteger holds a single integer
# :rtype List[NestedInteger]
# """

class Solution(object):
def depthSum(self, nestedList):
"""
:type nestedList: List[NestedInteger]
:rtype: int
"""
```

JavaScript:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* function NestedInteger() {
*
* Return true if this NestedInteger holds a single integer, rather than a
nested list.
* @return {boolean}
* this.isInteger = function() {
* ...
* };
*
* Return the single integer that this NestedInteger holds, if it holds a
single integer
* The result is undefined if this NestedInteger holds a nested list
* @return {integer}
* this.getInteger = function() {
* ...
* };
*
* Set this NestedInteger to hold a single integer equal to value.
* @return {void}
* this.setInteger = function(value) {
* ...
* };
*
* Set this NestedInteger to hold a nested list and adds a nested integer elem
to it.
* @return {void}
* this.add = function(elem) {
* ...
* };
*
* Return the nested list that this NestedInteger holds, if it holds a nested
list
* The result is undefined if this NestedInteger holds a single integer
* @return {NestedInteger[]}
* this.getList = function() {
* ...
* };
* };
*/
/**
* @param {NestedInteger[]} nestedList
* @return {number}
*/
var depthSum = function(nestedList) {

};
```

TypeScript:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
* If value is provided, then it holds a single integer
* Otherwise it holds an empty nested list
* constructor(value?: number) {
* ...
* };
*
* Return true if this NestedInteger holds a single integer, rather than a
nested list.
* isInteger(): boolean {
* ...
* };
*
* Return the single integer that this NestedInteger holds, if it holds a
single integer
* The result is undefined if this NestedInteger holds a nested list
* getInteger(): number | null {
* ...
* };
*
* Set this NestedInteger to hold a single integer equal to value.
* setInteger(value: number) {
* ...
* };
*
* Set this NestedInteger to hold a nested list and adds a nested integer elem
to it.
* add(elem: NestedInteger) {
* ...
* };
*
* Return the nested list that this NestedInteger holds
* The result is undefined if this NestedInteger holds a single integer
* getList(): NestedInteger[] {
* ...
* };
* };
*/

function depthSum(nestedList: NestedInteger[]): number {

};
```

C#:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* interface NestedInteger {
*
* // Constructor initializes an empty nested list.
* public NestedInteger();
*
* // Constructor initializes a single integer.
* public NestedInteger(int value);
*
* // @return true if this NestedInteger holds a single integer, rather than a
nested list.
* bool IsInteger();
*
* // @return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* int GetInteger();
*
* // Set this NestedInteger to hold a single integer.
* public void SetInteger(int value);
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* public void Add(NestedInteger ni);
*
* // @return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* IList<NestedInteger> GetList();
* }
*/
public class Solution {
public int DepthSum(IList<NestedInteger> nestedList) {

}
}
```

C:

```
/**
* *********************************************************************
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* *********************************************************************
*
* // Initializes an empty nested list and return a reference to the nested
integer.
* struct NestedInteger *NestedIntegerInit();
*
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* bool NestedIntegerIsInteger(struct NestedInteger *);
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* int NestedIntegerGetInteger(struct NestedInteger *);
*
* // Set this NestedInteger to hold a single integer.
* void NestedIntegerSetInteger(struct NestedInteger *ni, int value);
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
elem to it.
* void NestedIntegerAdd(struct NestedInteger *ni, struct NestedInteger
*elem);
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* struct NestedInteger **NestedIntegerGetList(struct NestedInteger *);
*
* // Return the nested list's size that this NestedInteger holds, if it holds
a nested list
* // The result is undefined if this NestedInteger holds a single integer
* int NestedIntegerGetListSize(struct NestedInteger *);
* };
*/
int depthSum(struct NestedInteger** nestedList, int nestedListSize) {

}
```

Go:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* type NestedInteger struct {
* }
*
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* func (n NestedInteger) IsInteger() bool {}
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* // So before calling this method, you should have a check
* func (n NestedInteger) GetInteger() int {}
*
* // Set this NestedInteger to hold a single integer.
* func (n *NestedInteger) SetInteger(value int) {}
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* func (n *NestedInteger) Add(elem NestedInteger) {}
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* // You can access NestedInteger's List element directly if you want to
modify it
* func (n NestedInteger) GetList() []*NestedInteger {}
*/
func depthSum(nestedList []*NestedInteger) int {

}
```

Kotlin:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
* // Constructor initializes an empty nested list.
* constructor()
*
* // Constructor initializes a single integer.
* constructor(value: Int)
*
* // @return true if this NestedInteger holds a single integer, rather than a
nested list.
* fun isInteger(): Boolean
*
* // @return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* fun getInteger(): Int?
*
* // Set this NestedInteger to hold a single integer.
* fun setInteger(value: Int): Unit
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* fun add(ni: NestedInteger): Unit
*
* // @return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* fun getList(): List<NestedInteger>?
* }
*/
class Solution {
fun depthSum(nestedList: List<NestedInteger>): Int {

}
}
```

Swift:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* public func isInteger() -> Bool
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* public func getInteger() -> Int
*
* // Set this NestedInteger to hold a single integer.
* public func setInteger(value: Int)
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* public func add(elem: NestedInteger)
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* public func getList() -> [NestedInteger]
* }
*/
class Solution {
func depthSum(_ nestedList: [NestedInteger]) -> Int {

}
}
```

Rust:

```
// #[derive(Debug, PartialEq, Eq)]
// pub enum NestedInteger {
// Int(i32),
// List(Vec<NestedInteger>)
// }
impl Solution {
pub fn depth_sum(nested_list: Vec<NestedInteger>) -> i32 {

}
}
```

Ruby:

```
# This is the interface that allows for creating nested lists.
# You should not implement it, or speculate about its implementation
#
#class NestedInteger
# def is_integer()
# """
# Return true if this NestedInteger holds a single integer, rather than a
nested list.
# @return {Boolean}
# """
#
# def get_integer()
# """
# Return the single integer that this NestedInteger holds, if it holds a
single integer
# The result is undefined if this NestedInteger holds a nested list
# @return {Integer}
# """
#
# def set_integer(value)
# """
# Set this NestedInteger to hold a single integer equal to value.
# @return {Void}
# """
#
# def add(elem)
# """
# Set this NestedInteger to hold a nested list and adds a nested integer elem
to it.
# @return {Void}
# """
#
# def get_list()
# """
# Return the nested list that this NestedInteger holds, if it holds a nested
list
# The result is undefined if this NestedInteger holds a single integer
# @return {NestedInteger[]}
# """

# @param {NestedInteger[]} nested_list
# @return {Integer}
def depth_sum(nested_list)

end
```

PHP:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {

* // if value is not specified, initializes an empty list.
* // Otherwise initializes a single integer equal to value.
* function __construct($value = null)

* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* function isInteger() : bool
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* function getInteger()
*
* // Set this NestedInteger to hold a single integer.
* function setInteger($i) : void
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* function add($ni) : void
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* function getList() : array
* }
*/
class Solution {

/**
* @param NestedInteger[] $nestedList
* @return Integer
*/
function depthSum($nestedList) {

}
}
```

Dart:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
* // If [integer] is an int, constructor initializes a single integer.
* // Otherwise it initializes an empty nested list.
* NestedInteger([int? integer]);
*
* // Returns true if this NestedInteger holds a single integer, rather than a
nested list.
* bool isInteger();
*
* // Returns the single integer that this NestedInteger holds, if it holds a
single integer.
* // The result is undefined if this NestedInteger holds a nested list
* int getInteger();
*
* // Sets this NestedInteger to hold a single integer.
* void setInteger(int value);
*
* // Sets this NestedInteger to hold a nested list and adds a nested integer
to it.
* void add(NestedInteger ni);
*
* // Returns the nested list that this NestedInteger holds, if it holds a
nested list.
* // The result is undefined if this NestedInteger holds a single integer
* List<NestedInteger> getList();
* }
*/
class Solution {
int depthSum(List<NestedInteger> nestedList) {

}
}
```

Scala:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* trait NestedInteger {
*
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* def isInteger: Boolean
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer.
* def getInteger: Int
*
* // Set this NestedInteger to hold a single integer.
* def setInteger(i: Int): Unit
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list.
* def getList: Array[NestedInteger]
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* def add(ni: NestedInteger): Unit
* }
*/
object Solution {
def depthSum(nestedList: List[NestedInteger]): Int = {

}
}
```

Elixir:

```
# # This is the interface that allows for creating nested lists.
# # You should not implement it, or speculate about its implementation
#
# # Create an empty nested list.
# :nested_integer.new() :: :nested_integer.nested_integer
#
# # Create a single integer.
# :nested_integer.new(val :: integer) :: :nested_integer.nested_integer
#
# # Return true if argument nested_integer holds a single integer, rather
than a nested list.
```

```
# :nested_integer.is_integer(nested_integer ::
:nested_integer.nested_integer) :: boolean
#
# # Return the single integer that nested_integer holds, if it holds a single
integer
# # The result is undefined if it holds a nested list.
# :nested_integer.get_integer(nested_integer ::
:nested_integer.nested_integer) :: integer
#
# # Return a copy of argument nested_integer with it set to hold a single
integer val.
# :nested_integer.set_integer(nested_integer ::
:nested_integer.nested_integer, val :: integer) ::
:nested_integer.nested_integer
#
# # Return a copy of argument nested_integer with it set to hold a nested
list and adds a nested_integer elem to it.
# :nested_integer.add(nested_integer :: :nested_integer.nested_integer, elem
:: :nested_integer.nested_integer) :: :nested_integer.nested_integer
#
# # Return the nested list that nested_integer holds, if it holds a nested
list.
# # The result is undefined if it holds a single integer.
# :nested_integer.get_list(nested_integer :: :nested_integer.nested_integer)
:: :array.array(:nested_integer.nested_integer)

defmodule Solution do
@spec depth_sum(nested_list :: [:nested_integer.nested_integer]) :: integer
def depth_sum(nested_list) do

end
end
```

Erlang:

```
%% % This is the interface that allows for creating nested lists.
%% % You should not implement it, or speculate about its implementation
%%
%% % Create an empty nested list.
%% nested_integer:new() -> nested_integer().
%%
%% % Create a single integer.
%% nested_integer:new(Val :: integer()) -> nested_integer().
%%
%% % Return true if argument NestedInteger holds a single integer, rather
than a nested list.
%% nested_integer:is_integer(NestedInteger :: nested_integer()) -> boolean().
%%
%% % Return the single integer that NestedInteger holds, if it holds a single
integer.
%% % The result is undefined if it holds a nested list.
%% nested_integer:get_integer(NestedInteger :: nested_integer()) ->
integer().
%%
%% % Return a copy of argument NestedInteger with it set to hold a single
integer Val.
%% nested_integer:set_integer(NestedInteger :: nested_integer(), Val ::
integer()) -> nested_integer().
%%
%% % Return a copy of argument NestedInteger with it set to hold a nested
list and adds a nested_integer Elem to it.
%% nested_integer:add(NestedInteger :: nested_integer(), Elem ::
nested_integer()) -> nested_integer().
%%
%% % Return the nested list that NestedInteger holds, if it holds a nested
list.
%% % The result is undefined if it holds a single integer.
%% nested_integer:get_list(NestedInteger :: nested_integer()) ->
array:array(nested_integer()).

-spec depth_sum(NestedList :: [nested_integer:nested_integer()]) ->
integer().
depth_sum(NestedList) ->
.
```

Racket:

```
;; This is the interface that allows for creating nested lists.
;; You should not implement it, or speculate about its implementation

#|

(define nested-integer%
(class object%
...

; Return true if this nested-integer% holds a single integer, rather than a
nested list.
; -> boolean?
(define/public (is-integer)
...)

; Return the single integer that this nested-integer% holds, if it holds a
single integer,
; the result is undefined if this nested-integer% holds a nested list.
; -> integer?
(define/public (get-integer)
...)

; Set this nested-integer% to hold a single integer equal to value.
; -> integer? void?
(define/public (set-integer i)
...)

; Set this nested-integer% to hold a nested list and adds a nested integer
elem to it.
; -> (is-a?/c nested-integer%) void?
(define/public (add ni)
...)

; Return the nested list that this nested-integer% holds,
; the result is undefined if this nested-integer% holds a single integer.
; -> gvector?
(define/public (get-list)
...)))

|#

(define/contract (depth-sum nestedList)
(-> (listof (is-a?/c nested-integer%)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Nested List Weight Sum
* Difficulty: Medium
* Tags: search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
* public:
* // Constructor initializes an empty nested list.
* NestedInteger();
*
* // Constructor initializes a single integer.
* NestedInteger(int value);
*
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* bool isInteger() const;
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* int getInteger() const;
*
* // Set this NestedInteger to hold a single integer.
* void setInteger(int value);
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* void add(const NestedInteger &ni);
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* const vector<NestedInteger> &getList() const;
* };
*/
class Solution {
public:
int depthSum(vector<NestedInteger>& nestedList) {

}
};
```

Java Solution:

```
/**
* Problem: Nested List Weight Sum
* Difficulty: Medium
* Tags: search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* public interface NestedInteger {
* // Constructor initializes an empty nested list.
* public NestedInteger();
*
* // Constructor initializes a single integer.
* public NestedInteger(int value);
*
* // @return true if this NestedInteger holds a single integer, rather than a
nested list.
* public boolean isInteger();
*
* // @return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* public Integer getInteger();
*
* // Set this NestedInteger to hold a single integer.
* public void setInteger(int value);
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* public void add(NestedInteger ni);
*
* // @return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* public List<NestedInteger> getList();
* }
*/
class Solution {
public int depthSum(List<NestedInteger> nestedList) {

}
}
```

Python3 Solution:

```
"""
Problem: Nested List Weight Sum
Difficulty: Medium
Tags: search

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

# """
# This is the interface that allows for creating nested lists.
# You should not implement it, or speculate about its implementation
# """
#class NestedInteger:
# def __init__(self, value=None):
# """
# If value is not specified, initializes an empty list.
# Otherwise initializes a single integer equal to value.
# """
#
```

```
# def isInteger(self):
# """
# @return True if this NestedInteger holds a single integer, rather than a
nested list.
# :rtype bool
# """
#
# def add(self, elem):
# """
# Set this NestedInteger to hold a nested list and adds a nested integer elem
to it.
# :rtype void
# """
#
# def setInteger(self, value):
# """
# Set this NestedInteger to hold a single integer equal to value.
# :rtype void
# """
#
# def getInteger(self):
# """
# @return the single integer that this NestedInteger holds, if it holds a
single integer
# The result is undefined if this NestedInteger holds a nested list
# :rtype int
# """
#
# def getList(self):
# """
# @return the nested list that this NestedInteger holds, if it holds a nested
list
# The result is undefined if this NestedInteger holds a single integer
# :rtype List[NestedInteger]
# """

class Solution:
def depthSum(self, nestedList: List[NestedInteger]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
# """
# This is the interface that allows for creating nested lists.
# You should not implement it, or speculate about its implementation
# """
#class NestedInteger(object):
# def __init__(self, value=None):
# """
# If value is not specified, initializes an empty list.
# Otherwise initializes a single integer equal to value.
# """
#
# def isInteger(self):
# """
# @return True if this NestedInteger holds a single integer, rather than a
nested list.
# :rtype bool
# """
#
# def add(self, elem):
# """
# Set this NestedInteger to hold a nested list and adds a nested integer elem
to it.
# :rtype void
# """
#
# def setInteger(self, value):
# """
# Set this NestedInteger to hold a single integer equal to value.
# :rtype void
# """
#
# def getInteger(self):
# """
# @return the single integer that this NestedInteger holds, if it holds a
single integer
# The result is undefined if this NestedInteger holds a nested list
# :rtype int
# """
#
# def getList(self):
# """
# @return the nested list that this NestedInteger holds, if it holds a nested
list
# The result is undefined if this NestedInteger holds a single integer
# :rtype List[NestedInteger]
# """

class Solution(object):
def depthSum(self, nestedList):
"""
:type nestedList: List[NestedInteger]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Nested List Weight Sum
* Difficulty: Medium
* Tags: search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* function NestedInteger() {
*
* Return true if this NestedInteger holds a single integer, rather than a
nested list.
* @return {boolean}
* this.isInteger = function() {
* ...
* };
*
* Return the single integer that this NestedInteger holds, if it holds a
single integer
* The result is undefined if this NestedInteger holds a nested list
* @return {integer}
* this.getInteger = function() {
* ...
* };
*
* Set this NestedInteger to hold a single integer equal to value.
* @return {void}
* this.setInteger = function(value) {
* ...
* };
*
* Set this NestedInteger to hold a nested list and adds a nested integer elem
to it.
* @return {void}
* this.add = function(elem) {
* ...
* };
*
* Return the nested list that this NestedInteger holds, if it holds a nested
list
* The result is undefined if this NestedInteger holds a single integer
* @return {NestedInteger[]}
* this.getList = function() {
* ...
* };
* };
*/
/**
* @param {NestedInteger[]} nestedList
* @return {number}
*/
var depthSum = function(nestedList) {

};
```

TypeScript Solution:

```
/**
* Problem: Nested List Weight Sum
* Difficulty: Medium
* Tags: search
*
```

```
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
* If value is provided, then it holds a single integer
* Otherwise it holds an empty nested list
* constructor(value?: number) {
* ...
* };
*
* Return true if this NestedInteger holds a single integer, rather than a
nested list.
* isInteger(): boolean {
* ...
* };
*
* Return the single integer that this NestedInteger holds, if it holds a
single integer
* The result is undefined if this NestedInteger holds a nested list
* getInteger(): number | null {
* ...
* };
*
* Set this NestedInteger to hold a single integer equal to value.
* setInteger(value: number) {
* ...
* };
*
* Set this NestedInteger to hold a nested list and adds a nested integer elem
to it.
* add(elem: NestedInteger) {
* ...
* };
*
* Return the nested list that this NestedInteger holds
* The result is undefined if this NestedInteger holds a single integer
* getList(): NestedInteger[] {
```

```
* ...
* };
* };
*/

function depthSum(nestedList: NestedInteger[]): number {

};
```

C# Solution:

```
/*
* Problem: Nested List Weight Sum
* Difficulty: Medium
* Tags: search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* interface NestedInteger {
*
* // Constructor initializes an empty nested list.
* public NestedInteger();
*
* // Constructor initializes a single integer.
* public NestedInteger(int value);
*
* // @return true if this NestedInteger holds a single integer, rather than a
nested list.
* bool IsInteger();
*
* // @return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* int GetInteger();
*
* // Set this NestedInteger to hold a single integer.
* public void SetInteger(int value);
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* public void Add(NestedInteger ni);
*
* // @return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* IList<NestedInteger> GetList();
* }
*/
public class Solution {
public int DepthSum(IList<NestedInteger> nestedList) {

}
}
```

C Solution:

```
/*
* Problem: Nested List Weight Sum
* Difficulty: Medium
* Tags: search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* *********************************************************************
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* *********************************************************************
*
* // Initializes an empty nested list and return a reference to the nested
integer.
* struct NestedInteger *NestedIntegerInit();
*
```

```
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* bool NestedIntegerIsInteger(struct NestedInteger *);
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* int NestedIntegerGetInteger(struct NestedInteger *);
*
* // Set this NestedInteger to hold a single integer.
* void NestedIntegerSetInteger(struct NestedInteger *ni, int value);
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
elem to it.
* void NestedIntegerAdd(struct NestedInteger *ni, struct NestedInteger
*elem);
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* struct NestedInteger **NestedIntegerGetList(struct NestedInteger *);
*
* // Return the nested list's size that this NestedInteger holds, if it holds
a nested list
* // The result is undefined if this NestedInteger holds a single integer
* int NestedIntegerGetListSize(struct NestedInteger *);
* };
*/
int depthSum(struct NestedInteger** nestedList, int nestedListSize) {

}
```

Go Solution:

```
// Problem: Nested List Weight Sum
// Difficulty: Medium
// Tags: search
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach
```

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* type NestedInteger struct {
* }
*
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* func (n NestedInteger) IsInteger() bool {}
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* // So before calling this method, you should have a check
* func (n NestedInteger) GetInteger() int {}
*
* // Set this NestedInteger to hold a single integer.
* func (n *NestedInteger) SetInteger(value int) {}
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* func (n *NestedInteger) Add(elem NestedInteger) {}
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* // You can access NestedInteger's List element directly if you want to
modify it
* func (n NestedInteger) GetList() []*NestedInteger {}
*/
func depthSum(nestedList []*NestedInteger) int {

}
```

Kotlin Solution:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
```

```
* // Constructor initializes an empty nested list.
* constructor()
*
* // Constructor initializes a single integer.
* constructor(value: Int)
*
* // @return true if this NestedInteger holds a single integer, rather than a
nested list.
* fun isInteger(): Boolean
*
* // @return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* fun getInteger(): Int?
*
* // Set this NestedInteger to hold a single integer.
* fun setInteger(value: Int): Unit
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* fun add(ni: NestedInteger): Unit
*
* // @return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* fun getList(): List<NestedInteger>?
* }
*/
class Solution {
fun depthSum(nestedList: List<NestedInteger>): Int {

}
}
```

Swift Solution:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* public func isInteger() -> Bool
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* public func getInteger() -> Int
*
* // Set this NestedInteger to hold a single integer.
* public func setInteger(value: Int)
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* public func add(elem: NestedInteger)
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* public func getList() -> [NestedInteger]
* }
*/
class Solution {
func depthSum(_ nestedList: [NestedInteger]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Nested List Weight Sum
// Difficulty: Medium
// Tags: search
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

// #[derive(Debug, PartialEq, Eq)]
// pub enum NestedInteger {
// Int(i32),
// List(Vec<NestedInteger>)
// }
```

```
impl Solution {
pub fn depth_sum(nested_list: Vec<NestedInteger>) -> i32 {

}
}
```

Ruby Solution:

```
# This is the interface that allows for creating nested lists.
# You should not implement it, or speculate about its implementation
#
#class NestedInteger
# def is_integer()
# """
# Return true if this NestedInteger holds a single integer, rather than a
nested list.
# @return {Boolean}
# """
#
# def get_integer()
# """
# Return the single integer that this NestedInteger holds, if it holds a
single integer
# The result is undefined if this NestedInteger holds a nested list
# @return {Integer}
# """
#
# def set_integer(value)
# """
# Set this NestedInteger to hold a single integer equal to value.
# @return {Void}
# """
#
# def add(elem)
# """
# Set this NestedInteger to hold a nested list and adds a nested integer elem
to it.
# @return {Void}
# """
#
# def get_list()
# """
# Return the nested list that this NestedInteger holds, if it holds a nested
list
# The result is undefined if this NestedInteger holds a single integer
# @return {NestedInteger[]}
# """

# @param {NestedInteger[]} nested_list
# @return {Integer}
def depth_sum(nested_list)

end
```

PHP Solution:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {

* // if value is not specified, initializes an empty list.
* // Otherwise initializes a single integer equal to value.
* function __construct($value = null)

* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* function isInteger() : bool
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* function getInteger()
*
* // Set this NestedInteger to hold a single integer.
* function setInteger($i) : void
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* function add($ni) : void
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* function getList() : array
* }
*/
class Solution {

/**
* @param NestedInteger[] $nestedList
* @return Integer
*/
function depthSum($nestedList) {

}
}
```

Dart Solution:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
* // If [integer] is an int, constructor initializes a single integer.
* // Otherwise it initializes an empty nested list.
* NestedInteger([int? integer]);
*
* // Returns true if this NestedInteger holds a single integer, rather than a
nested list.
* bool isInteger();
*
* // Returns the single integer that this NestedInteger holds, if it holds a
single integer.
* // The result is undefined if this NestedInteger holds a nested list
* int getInteger();
*
* // Sets this NestedInteger to hold a single integer.
* void setInteger(int value);
*
* // Sets this NestedInteger to hold a nested list and adds a nested integer
to it.
* void add(NestedInteger ni);
*
* // Returns the nested list that this NestedInteger holds, if it holds a
nested list.
* // The result is undefined if this NestedInteger holds a single integer
* List<NestedInteger> getList();
* }
*/
class Solution {
int depthSum(List<NestedInteger> nestedList) {

}
}
```

Scala Solution:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* trait NestedInteger {
*
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* def isInteger: Boolean
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer.
* def getInteger: Int
*
* // Set this NestedInteger to hold a single integer.
* def setInteger(i: Int): Unit
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list.
* def getList: Array[NestedInteger]
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* def add(ni: NestedInteger): Unit
* }
*/
object Solution {
def depthSum(nestedList: List[NestedInteger]): Int = {
}
}
```

Elixir Solution:

```
# # This is the interface that allows for creating nested lists.
# # You should not implement it, or speculate about its implementation
#
# # Create an empty nested list.
# :nested_integer.new() :: :nested_integer.nested_integer
#
# # Create a single integer.
# :nested_integer.new(val :: integer) :: :nested_integer.nested_integer
#
# # Return true if argument nested_integer holds a single integer, rather
than a nested list.
# :nested_integer.is_integer(nested_integer ::
:nested_integer.nested_integer) :: boolean
#
# # Return the single integer that nested_integer holds, if it holds a single
integer
# # The result is undefined if it holds a nested list.
# :nested_integer.get_integer(nested_integer ::
:nested_integer.nested_integer) :: integer
#
# # Return a copy of argument nested_integer with it set to hold a single
integer val.
# :nested_integer.set_integer(nested_integer ::
:nested_integer.nested_integer, val :: integer) ::
:nested_integer.nested_integer
#
# # Return a copy of argument nested_integer with it set to hold a nested
list and adds a nested_integer elem to it.
# :nested_integer.add(nested_integer :: :nested_integer.nested_integer, elem
:: :nested_integer.nested_integer) :: :nested_integer.nested_integer
#
# # Return the nested list that nested_integer holds, if it holds a nested
list.
# # The result is undefined if it holds a single integer.
# :nested_integer.get_list(nested_integer :: :nested_integer.nested_integer)
```

```
:: :array.array(:nested_integer.nested_integer)

defmodule Solution do
@spec depth_sum(nested_list :: [:nested_integer.nested_integer]) :: integer
def depth_sum(nested_list) do

end
end
```

Erlang Solution:

```
%% % This is the interface that allows for creating nested lists.
%% % You should not implement it, or speculate about its implementation
%%
%% % Create an empty nested list.
%% nested_integer:new() -> nested_integer().
%%
%% % Create a single integer.
%% nested_integer:new(Val :: integer()) -> nested_integer().
%%
%% % Return true if argument NestedInteger holds a single integer, rather
than a nested list.
%% nested_integer:is_integer(NestedInteger :: nested_integer()) -> boolean().
%%
%% % Return the single integer that NestedInteger holds, if it holds a single
integer.
%% % The result is undefined if it holds a nested list.
%% nested_integer:get_integer(NestedInteger :: nested_integer()) ->
integer().
%%
%% % Return a copy of argument NestedInteger with it set to hold a single
integer Val.
%% nested_integer:set_integer(NestedInteger :: nested_integer(), Val ::
integer()) -> nested_integer().
%%
%% % Return a copy of argument NestedInteger with it set to hold a nested
list and adds a nested_integer Elem to it.
%% nested_integer:add(NestedInteger :: nested_integer(), Elem ::
nested_integer()) -> nested_integer().
%%
%% % Return the nested list that NestedInteger holds, if it holds a nested
list.
```

```
%% % The result is undefined if it holds a single integer.
%% nested_integer:get_list(NestedInteger :: nested_integer()) ->
array:array(nested_integer()).

-spec depth_sum(NestedList :: [nested_integer:nested_integer()]) ->
integer().
depth_sum(NestedList) ->
.
```

Racket Solution:

```
;; This is the interface that allows for creating nested lists.
;; You should not implement it, or speculate about its implementation

#|

(define nested-integer%
(class object%
...

; Return true if this nested-integer% holds a single integer, rather than a
nested list.
; -> boolean?
(define/public (is-integer)
...)

; Return the single integer that this nested-integer% holds, if it holds a
single integer,
; the result is undefined if this nested-integer% holds a nested list.
; -> integer?
(define/public (get-integer)
...)

; Set this nested-integer% to hold a single integer equal to value.
; -> integer? void?
(define/public (set-integer i)
...)

; Set this nested-integer% to hold a nested list and adds a nested integer
elem to it.
; -> (is-a?/c nested-integer%) void?
(define/public (add ni)
...)

; Return the nested list that this nested-integer% holds,
; the result is undefined if this nested-integer% holds a single integer.
; -> gvector?
(define/public (get-list)
...)))

|#

(define/contract (depth-sum nestedList)
(-> (listof (is-a?/c nested-integer%)) exact-integer?)
)
```