Problem 1626: Best Team With No Conflicts

Problem Information

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

Problem Description

You are the manager of a basketball team. For the upcoming tournament, you want to choose the team with the highest overall score. The score of the team is the

sum

of scores of all the players in the team.

However, the basketball team is not allowed to have

conflicts

. A

conflict

exists if a younger player has a

strictly higher

score than an older player. A conflict does

not

occur between players of the same age.

Given two lists,

scores

and

ages

, where each

scores[i]

and

ages[i]

represents the score and age of the

i

th

player, respectively, return

the highest overall score of all possible basketball teams

.

Example 1:

Input:

scores = [1,3,5,10,15], ages = [1,2,3,4,5]

Output:

34

Explanation:

You can choose all the players.

Example 2:

Input:

scores = [4,5,6,5], ages = [2,1,2,1]

Output:

16

Explanation:

It is best to choose the last 3 players. Notice that you are allowed to choose multiple people of the same age.

Example 3:

Input:

scores = [1,2,3,5], ages = [8,9,10,1]

Output:

6

Explanation:

It is best to choose the first 3 players.

Constraints:

1 <= scores.length, ages.length <= 1000

scores.length == ages.length

1 <= scores[i] <= 10

6

1 <= ages[i] <= 1000

Code Snippets

C++:

```
class Solution {
public:
int bestTeamScore(vector<int>& scores, vector<int>& ages) {

}
};
```

Java:

```
class Solution {
public int bestTeamScore(int[] scores, int[] ages) {

}
}
```

Python3:

```
class Solution:
def bestTeamScore(self, scores: List[int], ages: List[int]) -> int:
```

Python:

```
class Solution(object):
def bestTeamScore(self, scores, ages):
"""
:type scores: List[int]
:type ages: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} scores
* @param {number[]} ages
* @return {number}
*/
var bestTeamScore = function(scores, ages) {

};
```

TypeScript:

```
function bestTeamScore(scores: number[], ages: number[]): number {

};
```

C#:

```
public class Solution {
public int BestTeamScore(int[] scores, int[] ages) {

}
}
```

C:

```
int bestTeamScore(int* scores, int scoresSize, int* ages, int agesSize) {

}
```

Go:

```
func bestTeamScore(scores []int, ages []int) int {

}
```

Kotlin:

```
class Solution {
fun bestTeamScore(scores: IntArray, ages: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func bestTeamScore(_ scores: [Int], _ ages: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn best_team_score(scores: Vec<i32>, ages: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} scores
# @param {Integer[]} ages
# @return {Integer}
def best_team_score(scores, ages)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $scores
* @param Integer[] $ages
* @return Integer
*/
function bestTeamScore($scores, $ages) {

}
}
```

Dart:

```
class Solution {
int bestTeamScore(List<int> scores, List<int> ages) {
```

```
}
}
```

Scala:

```
object Solution {
def bestTeamScore(scores: Array[Int], ages: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec best_team_score(scores :: [integer], ages :: [integer]) :: integer
def best_team_score(scores, ages) do

end
end
```

Erlang:

```
-spec best_team_score(Scores :: [integer()], Ages :: [integer()]) ->
integer().
best_team_score(Scores, Ages) ->
.
```

Racket:

```
(define/contract (best-team-score scores ages)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Best Team With No Conflicts
```

```
* Difficulty: Medium
* 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 bestTeamScore(vector<int>& scores, vector<int>& ages) {

}
};
```

Java Solution:

```
/**
* Problem: Best Team With No Conflicts
* Difficulty: Medium
* 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 bestTeamScore(int[] scores, int[] ages) {

}
}
```

Python3 Solution:

```
"""
Problem: Best Team With No Conflicts
Difficulty: Medium
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 bestTeamScore(self, scores: List[int], ages: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def bestTeamScore(self, scores, ages):
"""
:type scores: List[int]
:type ages: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Best Team With No Conflicts
* Difficulty: Medium
* 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[]} scores
* @param {number[]} ages
* @return {number}
*/
var bestTeamScore = function(scores, ages) {

};
```

TypeScript Solution:

```
/**
* Problem: Best Team With No Conflicts
* Difficulty: Medium
* 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 bestTeamScore(scores: number[], ages: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Best Team With No Conflicts
* Difficulty: Medium
* 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 BestTeamScore(int[] scores, int[] ages) {

}
}
```

C Solution:

```
/*
* Problem: Best Team With No Conflicts
* Difficulty: Medium
* 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 bestTeamScore(int* scores, int scoresSize, int* ages, int agesSize) {

}
```

Go Solution:

```
// Problem: Best Team With No Conflicts
// Difficulty: Medium
// 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 bestTeamScore(scores []int, ages []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun bestTeamScore(scores: IntArray, ages: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func bestTeamScore(_ scores: [Int], _ ages: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Best Team With No Conflicts
// Difficulty: Medium
// 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 best_team_score(scores: Vec<i32>, ages: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} scores
# @param {Integer[]} ages
# @return {Integer}
def best_team_score(scores, ages)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $scores
* @param Integer[] $ages
* @return Integer
*/
function bestTeamScore($scores, $ages) {

}
}
```

Dart Solution:

```
class Solution {
int bestTeamScore(List<int> scores, List<int> ages) {

}
}
```

Scala Solution:

```
object Solution {
def bestTeamScore(scores: Array[Int], ages: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec best_team_score(scores :: [integer], ages :: [integer]) :: integer
def best_team_score(scores, ages) do

end
end
```

Erlang Solution:

```
-spec best_team_score(Scores :: [integer()], Ages :: [integer()]) ->
integer().
best_team_score(Scores, Ages) ->
.
```

Racket Solution:

```
(define/contract (best-team-score scores ages)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```