Problem 1235: Maximum Profit in Job Scheduling

Problem Information

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

Problem Description

We have

n

jobs, where every job is scheduled to be done from

startTime[i]

to

endTime[i]

, obtaining a profit of

profit[i]

.

You're given the

startTime

,

endTime

and

profit

arrays, return the maximum profit you can take such that there are no two jobs in the subset with overlapping time range.

If you choose a job that ends at time

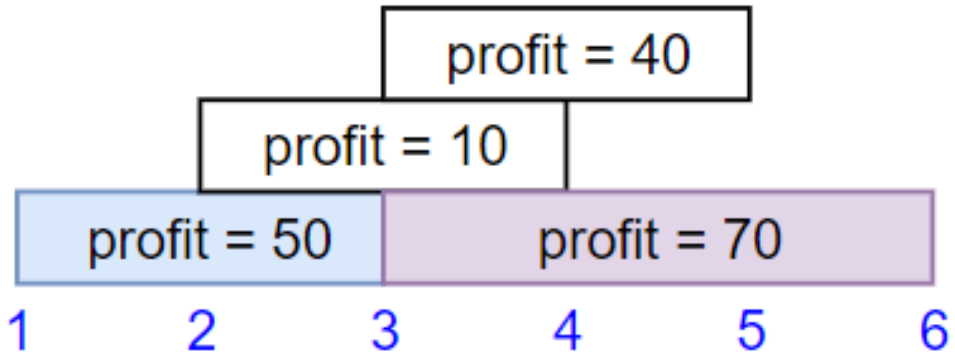
X

you will be able to start another job that starts at time

X

.

Example 1:



Input:

startTime = [1,2,3,3], endTime = [3,4,5,6], profit = [50,10,40,70]

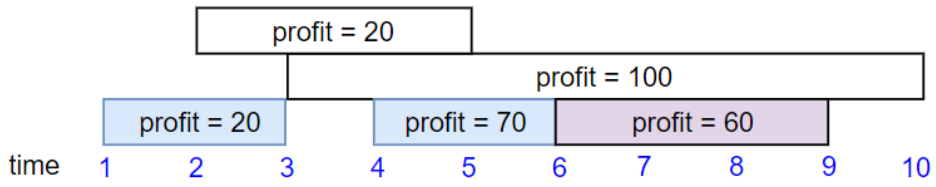
Output:

120

Explanation:

The subset chosen is the first and fourth job. Time range [1-3]+[3-6] , we get profit of 120 = 50 + 70.

Example 2:



Input:

startTime = [1,2,3,4,6], endTime = [3,5,10,6,9], profit = [20,20,100,70,60]

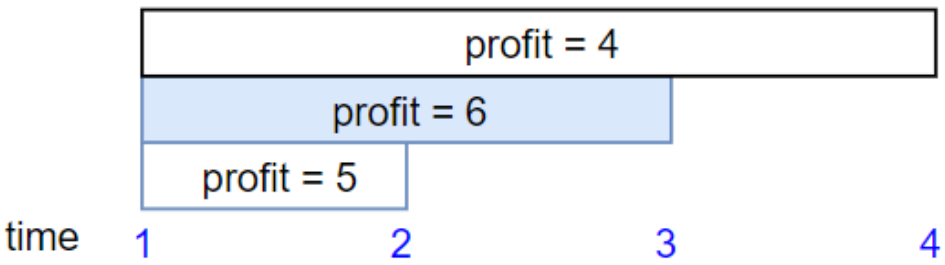
Output:

150

Explanation:

The subset chosen is the first, fourth and fifth job. Profit obtained 150 = 20 + 70 + 60.

Example 3:



Input:

startTime = [1,1,1], endTime = [2,3,4], profit = [5,6,4]

Output:

6

Constraints:

1 <= startTime.length == endTime.length == profit.length <= 5 * 10

4

1 <= startTime[i] < endTime[i] <= 10

9

1 <= profit[i] <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int jobScheduling(vector<int>& startTime, vector<int>& endTime, vector<int>&
profit) {

}
};
```

Java:

```
class Solution {
public int jobScheduling(int[] startTime, int[] endTime, int[] profit) {

}
}
```

Python3:

```
class Solution:
def jobScheduling(self, startTime: List[int], endTime: List[int], profit:
List[int]) -> int:
```

Python:

```
class Solution(object):
def jobScheduling(self, startTime, endTime, profit):
"""
:type startTime: List[int]
:type endTime: List[int]
:type profit: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} startTime
* @param {number[]} endTime
* @param {number[]} profit
* @return {number}
*/
var jobScheduling = function(startTime, endTime, profit) {

};
```

TypeScript:

```
function jobScheduling(startTime: number[], endTime: number[], profit:
number[]): number {

};
```

C#:

```
public class Solution {
public int JobScheduling(int[] startTime, int[] endTime, int[] profit) {

}
}
```

C:

```
int jobScheduling(int* startTime, int startTimeSize, int* endTime, int
endTimeSize, int* profit, int profitSize) {
```

```
}
```

Go:

```
func jobScheduling(startTime []int, endTime []int, profit []int) int {

}
```

Kotlin:

```
class Solution {
fun jobScheduling(startTime: IntArray, endTime: IntArray, profit: IntArray):
Int {

}
}
```

Swift:

```
class Solution {
func jobScheduling(_ startTime: [Int], _ endTime: [Int], _ profit: [Int]) ->
Int {

}
}
```

Rust:

```
impl Solution {
pub fn job_scheduling(start_time: Vec<i32>, end_time: Vec<i32>, profit:
Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} start_time
# @param {Integer[]} end_time
# @param {Integer[]} profit
# @return {Integer}
def job_scheduling(start_time, end_time, profit)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[] $startTime
* @param Integer[] $endTime
* @param Integer[] $profit
* @return Integer
*/
function jobScheduling($startTime, $endTime, $profit) {

}
}
```

Dart:

```
class Solution {
int jobScheduling(List<int> startTime, List<int> endTime, List<int> profit) {

}
}
```

Scala:

```
object Solution {
def jobScheduling(startTime: Array[Int], endTime: Array[Int], profit:
Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec job_scheduling(start_time :: [integer], end_time :: [integer], profit
:: [integer]) :: integer
def job_scheduling(start_time, end_time, profit) do
```

```
end
end
```

Erlang:

```
-spec job_scheduling(StartTime :: [integer()], EndTime :: [integer()], Profit
:: [integer()]) -> integer().
job_scheduling(StartTime, EndTime, Profit) ->
.
```

Racket:

```
(define/contract (job-scheduling startTime endTime profit)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?)
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Profit in Job Scheduling
* Difficulty: Hard
* Tags: array, dp, sort, search
*
* 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 jobScheduling(vector<int>& startTime, vector<int>& endTime, vector<int>&
profit) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Profit in Job Scheduling
* Difficulty: Hard
* Tags: array, dp, sort, search
*
* 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 jobScheduling(int[] startTime, int[] endTime, int[] profit) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Profit in Job Scheduling
Difficulty: Hard
Tags: array, dp, sort, search

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 jobScheduling(self, startTime: List[int], endTime: List[int], profit:
List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def jobScheduling(self, startTime, endTime, profit):
"""
:type startTime: List[int]
:type endTime: List[int]
:type profit: List[int]
```

```
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Profit in Job Scheduling
* Difficulty: Hard
* Tags: array, dp, sort, search
*
* 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[]} startTime
* @param {number[]} endTime
* @param {number[]} profit
* @return {number}
*/
var jobScheduling = function(startTime, endTime, profit) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Profit in Job Scheduling
* Difficulty: Hard
* Tags: array, dp, sort, search
*
* 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 jobScheduling(startTime: number[], endTime: number[], profit:
number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Profit in Job Scheduling
* Difficulty: Hard
* Tags: array, dp, sort, search
*
* 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 JobScheduling(int[] startTime, int[] endTime, int[] profit) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Profit in Job Scheduling
* Difficulty: Hard
* Tags: array, dp, sort, search
*
* 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 jobScheduling(int* startTime, int startTimeSize, int* endTime, int
endTimeSize, int* profit, int profitSize) {

}
```

Go Solution:

```
// Problem: Maximum Profit in Job Scheduling
// Difficulty: Hard
// Tags: array, dp, sort, search
//
// 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 jobScheduling(startTime []int, endTime []int, profit []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun jobScheduling(startTime: IntArray, endTime: IntArray, profit: IntArray):
Int {

}
}
```

Swift Solution:

```
class Solution {
func jobScheduling(_ startTime: [Int], _ endTime: [Int], _ profit: [Int]) ->
Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Profit in Job Scheduling
// Difficulty: Hard
// Tags: array, dp, sort, search
//
// 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 job_scheduling(start_time: Vec<i32>, end_time: Vec<i32>, profit:
Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} start_time
# @param {Integer[]} end_time
# @param {Integer[]} profit
# @return {Integer}
def job_scheduling(start_time, end_time, profit)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $startTime
* @param Integer[] $endTime
* @param Integer[] $profit
* @return Integer
*/
function jobScheduling($startTime, $endTime, $profit) {

}
}
```

Dart Solution:

```
class Solution {
int jobScheduling(List<int> startTime, List<int> endTime, List<int> profit) {

}
}
```

Scala Solution:

```
object Solution {
def jobScheduling(startTime: Array[Int], endTime: Array[Int], profit:
Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec job_scheduling(start_time :: [integer], end_time :: [integer], profit
:: [integer]) :: integer
def job_scheduling(start_time, end_time, profit) do

end
end
```

Erlang Solution:

```
-spec job_scheduling(StartTime :: [integer()], EndTime :: [integer()], Profit
:: [integer()]) -> integer().
job_scheduling(StartTime, EndTime, Profit) ->
.
```

Racket Solution:

```
(define/contract (job-scheduling startTime endTime profit)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?)
exact-integer?)
)
```