Problem 857: Minimum Cost to Hire K Workers

Problem Information

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

Problem Description

There are

n

workers. You are given two integer arrays

quality

and

wage

where

quality[i]

is the quality of the

i

th

worker and

wage[i]

is the minimum wage expectation for the

i

th

worker.

We want to hire exactly

k

workers to form a

paid group

. To hire a group of

k

workers, we must pay them according to the following rules:

Every worker in the paid group must be paid at least their minimum wage expectation.

In the group, each worker's pay must be directly proportional to their quality. This means if a worker’s quality is double that of another worker in the group, then they must be paid twice as much as the other worker.

Given the integer

k

, return

the least amount of money needed to form a paid group satisfying the above conditions

. Answers within

10

-5

of the actual answer will be accepted.

Example 1:

Input:

quality = [10,20,5], wage = [70,50,30], k = 2

Output:

105.00000

Explanation:

We pay 70 to 0

th

worker and 35 to 2

nd

worker.

Example 2:

Input:

quality = [3,1,10,10,1], wage = [4,8,2,2,7], k = 3

Output:

30.66667

Explanation:

We pay 4 to 0

th

worker, 13.33333 to 2

nd

and 3

rd

workers separately.

Constraints:

n == quality.length == wage.length

1 <= k <= n <= 10

4

1 <= quality[i], wage[i] <= 10

4

Code Snippets

C++:

```
class Solution {
public:
double mincostToHireWorkers(vector<int>& quality, vector<int>& wage, int k) {

}
};
```

Java:

```
class Solution {
public double mincostToHireWorkers(int[] quality, int[] wage, int k) {

}
}
```

Python3:

```
class Solution:
def mincostToHireWorkers(self, quality: List[int], wage: List[int], k: int)
-> float:
```

Python:

```
class Solution(object):
def mincostToHireWorkers(self, quality, wage, k):
"""
:type quality: List[int]
:type wage: List[int]
:type k: int
:rtype: float
"""
```

JavaScript:

```
/**
* @param {number[]} quality
* @param {number[]} wage
* @param {number} k
* @return {number}
*/
var mincostToHireWorkers = function(quality, wage, k) {

};
```

TypeScript:

```
function mincostToHireWorkers(quality: number[], wage: number[], k: number):
number {

};
```

C#:

```
public class Solution {
public double MincostToHireWorkers(int[] quality, int[] wage, int k) {

}
}
```

C:

```
double mincostToHireWorkers(int* quality, int qualitySize, int* wage, int
wageSize, int k) {

}
```

Go:

```
func mincostToHireWorkers(quality []int, wage []int, k int) float64 {

}
```

Kotlin:

```
class Solution {
fun mincostToHireWorkers(quality: IntArray, wage: IntArray, k: Int): Double {

}
}
```

Swift:

```
class Solution {
func mincostToHireWorkers(_ quality: [Int], _ wage: [Int], _ k: Int) ->
Double {

}
}
```

Rust:

```
impl Solution {
pub fn mincost_to_hire_workers(quality: Vec<i32>, wage: Vec<i32>, k: i32) ->
f64 {

}
```

```
}
```

Ruby:

```
# @param {Integer[]} quality
# @param {Integer[]} wage
# @param {Integer} k
# @return {Float}
def mincost_to_hire_workers(quality, wage, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $quality
* @param Integer[] $wage
* @param Integer $k
* @return Float
*/
function mincostToHireWorkers($quality, $wage, $k) {

}
}
```

Dart:

```
class Solution {
double mincostToHireWorkers(List<int> quality, List<int> wage, int k) {

}
}
```

Scala:

```
object Solution {
def mincostToHireWorkers(quality: Array[Int], wage: Array[Int], k: Int):
Double = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec mincost_to_hire_workers(quality :: [integer], wage :: [integer], k ::
integer) :: float
def mincost_to_hire_workers(quality, wage, k) do

end
end
```

Erlang:

```
-spec mincost_to_hire_workers(Quality :: [integer()], Wage :: [integer()], K
:: integer()) -> float().
mincost_to_hire_workers(Quality, Wage, K) ->
.
```

Racket:

```
(define/contract (mincost-to-hire-workers quality wage k)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer? flonum?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Cost to Hire K Workers
* Difficulty: Hard
* Tags: array, greedy, sort, 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:
double mincostToHireWorkers(vector<int>& quality, vector<int>& wage, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Cost to Hire K Workers
* Difficulty: Hard
* Tags: array, greedy, sort, 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 double mincostToHireWorkers(int[] quality, int[] wage, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Cost to Hire K Workers
Difficulty: Hard
Tags: array, greedy, sort, 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 mincostToHireWorkers(self, quality: List[int], wage: List[int], k: int)
-> float:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def mincostToHireWorkers(self, quality, wage, k):
"""
:type quality: List[int]
:type wage: List[int]
:type k: int
:rtype: float
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Cost to Hire K Workers
* Difficulty: Hard
* Tags: array, greedy, sort, 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[]} quality
* @param {number[]} wage
* @param {number} k
* @return {number}
*/
var mincostToHireWorkers = function(quality, wage, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Cost to Hire K Workers
* Difficulty: Hard
* Tags: array, greedy, sort, 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 mincostToHireWorkers(quality: number[], wage: number[], k: number):
number {

};
```

C# Solution:

```
/*
* Problem: Minimum Cost to Hire K Workers
* Difficulty: Hard
* Tags: array, greedy, sort, 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 double MincostToHireWorkers(int[] quality, int[] wage, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Cost to Hire K Workers
* Difficulty: Hard
* Tags: array, greedy, sort, 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
*/

double mincostToHireWorkers(int* quality, int qualitySize, int* wage, int
wageSize, int k) {
```

```
}
```

Go Solution:

```
// Problem: Minimum Cost to Hire K Workers
// Difficulty: Hard
// Tags: array, greedy, sort, 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 mincostToHireWorkers(quality []int, wage []int, k int) float64 {

}
```

Kotlin Solution:

```
class Solution {
fun mincostToHireWorkers(quality: IntArray, wage: IntArray, k: Int): Double {

}
}
```

Swift Solution:

```
class Solution {
func mincostToHireWorkers(_ quality: [Int], _ wage: [Int], _ k: Int) ->
Double {

}
}
```

Rust Solution:

```
// Problem: Minimum Cost to Hire K Workers
// Difficulty: Hard
// Tags: array, greedy, sort, 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 mincost_to_hire_workers(quality: Vec<i32>, wage: Vec<i32>, k: i32) ->
f64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} quality
# @param {Integer[]} wage
# @param {Integer} k
# @return {Float}
def mincost_to_hire_workers(quality, wage, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $quality
* @param Integer[] $wage
* @param Integer $k
* @return Float
*/
function mincostToHireWorkers($quality, $wage, $k) {

}
}
```

Dart Solution:

```
class Solution {
double mincostToHireWorkers(List<int> quality, List<int> wage, int k) {

}
}
```

Scala Solution:

```
object Solution {
def mincostToHireWorkers(quality: Array[Int], wage: Array[Int], k: Int):
Double = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec mincost_to_hire_workers(quality :: [integer], wage :: [integer], k ::
integer) :: float
def mincost_to_hire_workers(quality, wage, k) do

end
end
```

Erlang Solution:

```
-spec mincost_to_hire_workers(Quality :: [integer()], Wage :: [integer()], K
:: integer()) -> float().
mincost_to_hire_workers(Quality, Wage, K) ->
.
```

Racket Solution:

```
(define/contract (mincost-to-hire-workers quality wage k)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer? flonum?)
)
```