Problem 2418: Sort the People

Problem Information

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

Problem Description

You are given an array of strings

names

, and an array

heights

that consists of

distinct

positive integers. Both arrays are of length

n

.

For each index

i

,

names[i]

and

heights[i]

denote the name and height of the

i

th

person.

Return

names

sorted in

descending

order by the people's heights

.

Example 1:

Input:

names = ["Mary","John","Emma"], heights = [180,165,170]

Output:

["Mary","Emma","John"]

Explanation:

Mary is the tallest, followed by Emma and John.

Example 2:

Input:

names = ["Alice","Bob","Bob"], heights = [155,185,150]

Output:

["Bob","Alice","Bob"]

Explanation:

The first Bob is the tallest, followed by Alice and the second Bob.

Constraints:

n == names.length == heights.length

1 <= n <= 10

3

1 <= names[i].length <= 20

1 <= heights[i] <= 10

5

names[i]

consists of lower and upper case English letters.

All the values of

heights

are distinct.

Code Snippets

C++:

```
class Solution {
public:
vector<string> sortPeople(vector<string>& names, vector<int>& heights) {

}
};
```

Java:

```
class Solution {
public String[] sortPeople(String[] names, int[] heights) {

}
}
```

Python3:

```
class Solution:
def sortPeople(self, names: List[str], heights: List[int]) -> List[str]:
```

Python:

```
class Solution(object):
def sortPeople(self, names, heights):
"""
:type names: List[str]
:type heights: List[int]
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string[]} names
* @param {number[]} heights
* @return {string[]}
*/
var sortPeople = function(names, heights) {

};
```

TypeScript:

```
function sortPeople(names: string[], heights: number[]): string[] {

};
```

C#:

```
public class Solution {
public string[] SortPeople(string[] names, int[] heights) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** sortPeople(char** names, int namesSize, int* heights, int heightsSize,
int* returnSize) {

}
```

Go:

```
func sortPeople(names []string, heights []int) []string {

}
```

Kotlin:

```
class Solution {
fun sortPeople(names: Array<String>, heights: IntArray): Array<String> {

}
}
```

Swift:

```
class Solution {
func sortPeople(_ names: [String], _ heights: [Int]) -> [String] {
}
}
```

Rust:

```
impl Solution {
pub fn sort_people(names: Vec<String>, heights: Vec<i32>) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String[]} names
# @param {Integer[]} heights
# @return {String[]}
def sort_people(names, heights)

end
```

PHP:

```
class Solution {

/**
* @param String[] $names
* @param Integer[] $heights
* @return String[]
*/
function sortPeople($names, $heights) {

}
}
```

Dart:

```
class Solution {
List<String> sortPeople(List<String> names, List<int> heights) {

}
}
```

Scala:

```
object Solution {
def sortPeople(names: Array[String], heights: Array[Int]): Array[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec sort_people(names :: [String.t], heights :: [integer]) :: [String.t]
def sort_people(names, heights) do

end
end
```

Erlang:

```
-spec sort_people(Names :: [unicode:unicode_binary()], Heights ::
[integer()]) -> [unicode:unicode_binary()].
sort_people(Names, Heights) ->
.
```

Racket:

```
(define/contract (sort-people names heights)
(-> (listof string?) (listof exact-integer?) (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sort the People
* Difficulty: Easy
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

class Solution {
public:
vector<string> sortPeople(vector<string>& names, vector<int>& heights) {

}
};
```

Java Solution:

```
/**
* Problem: Sort the People
* Difficulty: Easy
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public String[] sortPeople(String[] names, int[] heights) {

}
}
```

Python3 Solution:

```
"""
Problem: Sort the People
Difficulty: Easy
Tags: array, string, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def sortPeople(self, names: List[str], heights: List[int]) -> List[str]:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def sortPeople(self, names, heights):
"""
:type names: List[str]
:type heights: List[int]
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Sort the People
* Difficulty: Easy
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string[]} names
* @param {number[]} heights
* @return {string[]}
*/
var sortPeople = function(names, heights) {

};
```

TypeScript Solution:

```
/**
* Problem: Sort the People
* Difficulty: Easy
* Tags: array, string, hash, sort
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function sortPeople(names: string[], heights: number[]): string[] {

};
```

C# Solution:

```
/*
* Problem: Sort the People
* Difficulty: Easy
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public string[] SortPeople(string[] names, int[] heights) {

}
}
```

C Solution:

```
/*
* Problem: Sort the People
* Difficulty: Easy
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
```

```
*/
char** sortPeople(char** names, int namesSize, int* heights, int heightsSize,
int* returnSize) {

}
```

Go Solution:

```
// Problem: Sort the People
// Difficulty: Easy
// Tags: array, string, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func sortPeople(names []string, heights []int) []string {

}
```

Kotlin Solution:

```
class Solution {
fun sortPeople(names: Array<String>, heights: IntArray): Array<String> {

}
}
```

Swift Solution:

```
class Solution {
func sortPeople(_ names: [String], _ heights: [Int]) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Sort the People
// Difficulty: Easy
// Tags: array, string, hash, sort
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn sort_people(names: Vec<String>, heights: Vec<i32>) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String[]} names
# @param {Integer[]} heights
# @return {String[]}
def sort_people(names, heights)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $names
* @param Integer[] $heights
* @return String[]
*/
function sortPeople($names, $heights) {

}
}
```

Dart Solution:

```
class Solution {
List<String> sortPeople(List<String> names, List<int> heights) {

}
}
```

Scala Solution:

```scala
object Solution {
def sortPeople(names: Array[String], heights: Array[Int]): Array[String] = {

}
}
```

Elixir Solution:

```elixir
defmodule Solution do
@spec sort_people(names :: [String.t], heights :: [integer]) :: [String.t]
def sort_people(names, heights) do

end
end
```

Erlang Solution:

```erlang
-spec sort_people(Names :: [unicode:unicode_binary()], Heights ::
[integer()]) -> [unicode:unicode_binary()].
sort_people(Names, Heights) ->
.
```

Racket Solution:

```racket
(define/contract (sort-people names heights)
(-> (listof string?) (listof exact-integer?) (listof string?))
)
```