# Problem 406: Queue Reconstruction by Height

## Problem Information

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

## Problem Description

You are given an array of people,

people

, which are the attributes of some people in a queue (not necessarily in order). Each

people[i] = [h

i

, k

i

]

represents the

i

th

person of height

h

i

with

exactly

k

i

other people in front who have a height greater than or equal to

h

i

.

Reconstruct and return

the queue that is represented by the input array

people

. The returned queue should be formatted as an array

queue

, where

queue[j] = [h

j

, k

j

]

is the attributes of the

j

th

person in the queue (

queue[0]

is the person at the front of the queue).

Example 1:

Input:

people = [[7,0],[4,4],[7,1],[5,0],[6,1],[5,2]]

Output:

[[5,0],[7,0],[5,2],[6,1],[4,4],[7,1]]

Explanation:

Person 0 has height 5 with no other people taller or the same height in front. Person 1 has height 7 with no other people taller or the same height in front. Person 2 has height 5 with two persons taller or the same height in front, which is person 0 and 1. Person 3 has height 6 with one person taller or the same height in front, which is person 1. Person 4 has height 4 with four people taller or the same height in front, which are people 0, 1, 2, and 3. Person 5 has height 7 with one person taller or the same height in front, which is person 1. Hence [[5,0],[7,0],[5,2],[6,1],[4,4],[7,1]] is the reconstructed queue.

Example 2:

Input:

people = [[6,0],[5,0],[4,0],[3,2],[2,2],[1,4]]

Output:

[[4,0],[5,0],[2,2],[3,2],[1,4],[6,0]]

Constraints:

1 <= people.length <= 2000

0 <= h

i

<= 10

6

0 <= k

i

< people.length

It is guaranteed that the queue can be reconstructed.

## Code Snippets

**C++:**

```
class Solution {
public:
vector<vector<int>> reconstructQueue(vector<vector<int>>& people) {

}
};
```

**Java:**

```
class Solution {
public int[][] reconstructQueue(int[][] people) {

}
}
```

### Python3:

```
class Solution:
def reconstructQueue(self, people: List[List[int]]) -> List[List[int]]:
```

### Python:

```
class Solution(object):
def reconstructQueue(self, people):
"""
:type people: List[List[int]]
:rtype: List[List[int]]
"""
```

### JavaScript:

```
/**
* @param {number[][]} people
* @return {number[][]}
*/
var reconstructQueue = function(people) {

};
```

### TypeScript:

```
function reconstructQueue(people: number[][]): number[][] {

};
```

### C#:

```
public class Solution {
public int[][] ReconstructQueue(int[][] people) {

}
}
```

### C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** reconstructQueue(int** people, int peopleSize, int* peopleColSize, int*
returnSize, int** returnColumnSizes) {

}
```

### Go:

```
func reconstructQueue(people [][]int) [][]int {

}
```

### Kotlin:

```
class Solution {
fun reconstructQueue(people: Array<IntArray>): Array<IntArray> {

}
}
```

### Swift:

```
class Solution {
func reconstructQueue(_ people: [[Int]]) -> [[Int]] {

}
}
```

### Rust:

```
impl Solution {
pub fn reconstruct_queue(people: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

### Ruby:

```
# @param {Integer[][]} people
# @return {Integer[][]}
def reconstruct_queue(people)

end
```

### PHP:

```
class Solution {

/**
* @param Integer[][] $people
* @return Integer[][]
*/
function reconstructQueue($people) {

}
}
```

### Dart:

```
class Solution {
List<List<int>> reconstructQueue(List<List<int>> people) {

}
}
```

### Scala:

```
object Solution {
def reconstructQueue(people: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

### Elixir:

```
defmodule Solution do
@spec reconstruct_queue(people :: [[integer]]) :: [[integer]]
def reconstruct_queue(people) do

end
end
```

### Erlang:

```
-spec reconstruct_queue(People :: [[integer()]]) -> [[integer()]].
reconstruct_queue(People) ->
.
```

### Racket:

```
(define/contract (reconstruct-queue people)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Queue Reconstruction by Height
* Difficulty: Medium
* Tags: array, tree, sort, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
vector<vector<int>> reconstructQueue(vector<vector<int>>& people) {

}
};
```

### Java Solution:

```
/**
* Problem: Queue Reconstruction by Height
* Difficulty: Medium
* Tags: array, tree, sort, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int[][] reconstructQueue(int[][] people) {

}
}
```

### Python3 Solution:

```
"""
Problem: Queue Reconstruction by Height
Difficulty: Medium
Tags: array, tree, sort, queue

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

### Python Solution:

```
class Solution(object):
def reconstructQueue(self, people):
"""
:type people: List[List[int]]
:rtype: List[List[int]]
"""
```

### JavaScript Solution:

```
/**
* Problem: Queue Reconstruction by Height
* Difficulty: Medium
* Tags: array, tree, sort, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number[][]} people
* @return {number[][]}
*/
var reconstructQueue = function(people) {

};
```

### TypeScript Solution:

```
/**
* Problem: Queue Reconstruction by Height
* Difficulty: Medium
* Tags: array, tree, sort, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function reconstructQueue(people: number[][]): number[][] {

};
```

### C# Solution:

```
/*
* Problem: Queue Reconstruction by Height
* Difficulty: Medium
* Tags: array, tree, sort, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```

*/

public class Solution {
public int[][] ReconstructQueue(int[][] people) {

}
}
```

### C Solution:

```
/*
* Problem: Queue Reconstruction by Height
* Difficulty: Medium
* Tags: array, tree, sort, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** reconstructQueue(int** people, int peopleSize, int* peopleColSize, int*
returnSize, int** returnColumnSizes) {

}
```

### Go Solution:

```
// Problem: Queue Reconstruction by Height
// Difficulty: Medium
// Tags: array, tree, sort, queue
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height
```

```
func reconstructQueue(people [][]int) [][]int {

}
```

### Kotlin Solution:

```
class Solution {
fun reconstructQueue(people: Array<IntArray>): Array<IntArray> {

}
}
```

### Swift Solution:

```
class Solution {
func reconstructQueue(_ people: [[Int]]) -> [[Int]] {

}
}
```

### Rust Solution:

```
// Problem: Queue Reconstruction by Height
// Difficulty: Medium
// Tags: array, tree, sort, queue
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn reconstruct_queue(people: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

### Ruby Solution:

```
# @param {Integer[][]} people
# @return {Integer[][]}
def reconstruct_queue(people)
```

```
end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[][] $people
* @return Integer[][]
*/
function reconstructQueue($people) {

}
}
```

### Dart Solution:

```
class Solution {
List<List<int>> reconstructQueue(List<List<int>> people) {

}
}
```

### Scala Solution:

```
object Solution {
def reconstructQueue(people: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec reconstruct_queue(people :: [[integer]]) :: [[integer]]
def reconstruct_queue(people) do

end
end
```

## Erlang Solution:

```
-spec reconstruct_queue(People :: [[integer()]]) -> [[integer()]].
reconstruct_queue(People) ->
.
```

## Racket Solution:

```
(define/contract (reconstruct-queue people)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```