Solution 1

```go
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

package main

import "sort"

type Disk []int
type Disks []Disk

func (disks Disks) Len() int           { return len(disks) }
func (disks Disks) Swap(i, j int)      { disks[i], disks[j] = disks[j], disks[i] }
func (disks Disks) Less(i, j int) bool { return disks[i][2] < disks[j][2] }

func DiskStacking(input [][]int) [][]int {
  disks := make(Disks, len(input))
  for i, disk := range input {
    disks[i] = disk
  }
  sort.Sort(disks)
  heights := make([]int, len(disks))
  sequences := make([]int, len(disks))
  for i := range disks {
    heights[i] = disks[i][2]
    sequences[i] = -1
  }
  for i := 1; i < len(disks); i++ {
    disk := disks[i]
    for j := 0; j < i; j++ {
      other := disks[j]
      // If the conditions of disk stacking are met
      if areValidDimensions(other, disk) {
        // If it's an increase in size
        if heights[i] <= disk[2]+heights[j] {
          heights[i] = disk[2] + heights[j]
          sequences[i] = j
        }
      }
    }
  }
  maxIndex := 0
  for i, height := range heights {
    if height >= heights[maxIndex] {
      maxIndex = i
    }
  }
  sequence := buildSequence(disks, sequences, maxIndex)
  return sequence
}

func areValidDimensions(o Disk, c Disk) bool {
  return o[0] < c[0] && o[1] < c[1] && o[2] < c[2]
}

func buildSequence(array []Disk, sequences []int, index int) [][]int {
  out := [][]int{}
  for index != -1 {
    out = append(out, array[index])
    index = sequences[index]
  }
  reverse(out)
  return out
}

func reverse(numbers [][]int) {
  for i, j := 0, len(numbers)-1; i < j; i, j = i+1, j-1 {
    numbers[i], numbers[j] = numbers[j], numbers[i]
  }
}

```