Solution 1

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

class Program {
    // O(nc) time | O(nc) space
    func knapsackProblem(_ items: [[Int]], _ capacity: Int) -> (Int, [Int]) {
        var knapsackValues = [[Int]]()

        for _ in 0 ..< items.count + 1 {
            let row = Array(repeating: 0, count: capacity + 1)
            knapsackValues.append(row)
        }

        for currentItemIndex in 1 ..< items.count + 1 {
            let currentValue = items[currentItemIndex - 1][0]
            let currentWeight = items[currentItemIndex - 1][1]

            for currentCapacity in 0 ..< capacity + 1 {
                if currentWeight <= currentCapacity {
                    knapsackValues[currentItemIndex][currentCapacity] = max(knapsackValues[currentItemIndex - 1][currentCapacity], knapsackValues[currentItemIndex - 1][currentCapacity - cur
                } else {
                    knapsackValues[currentItemIndex][currentCapacity] = knapsackValues[currentItemIndex - 1][currentCapacity]
                }
            }
        }

        return (knapsackValues[items.count][capacity], getKnapsackItems(items, knapsackValues))
    }

    func getKnapsackItems(_ items: [[Int]], _ knapsackValues: [[Int]]) -> [Int] {
        var sequence = [Int]()

        var currentItemIndex = knapsackValues.count - 1
        var currentCapacity = knapsackValues[0].count - 1

        while currentItemIndex > 0 {
            if knapsackValues[currentItemIndex][currentCapacity] == knapsackValues[currentItemIndex - 1][currentCapacity] {
                currentItemIndex -= 1
            } else {
                sequence.insert(currentItemIndex - 1, at: 0)

                currentCapacity -= items[currentItemIndex - 1][1]
                currentItemIndex -= 1
            }

            if currentCapacity == 0 {
                break
            }
        }

        return sequence
    }
}

```