def solve(toConsider, avail):
    global numCalls
    numCalls += 1
    if toConsider == [] or avail == 0:
        result = (0, ())
    elif toConsider[0].getWeight() > avail:
        result = solve(toConsider[1:], avail)
    else:
        nextItem = toConsider[0]
        #Explore left branch
        withVal, withToTake = solve(toConsider[1:], avail - nextItem.getWeight())
        withVal += nextItem.getValue()
        #Explore right branch
        withoutVal, withoutToTake = solve(toConsider[1:], avail)
        #Choose better branch
        if withVal > withoutVal:
            result = (withVal, withToTake + (nextItem,))
        else:
            result = (withoutVal, withoutToTake)
    return result

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>d</td>
<td>9</td>
<td>5</td>
</tr>
</tbody>
</table>
def fastSolve(toConsider, avail, memo = None):
    global numCalls
    numCalls += 1
    if memo == None:
        #Initialize for first invocation
        memo = {}  
    if (len(toConsider), avail) in memo:
        #Use solution found earlier
        result = memo[(len(toConsider), avail)]
        return result
    elif toConsider == [] or avail == 0:
        result = (0, ())
    elif toConsider[0].getWeight() > avail:
        #Lop off first item in toConsider and solve
        result = fastSolve(toConsider[1:], avail, memo)
    else:
        item = toConsider[0]
        #Consider taking first item
        withVal, withToTake = fastSolve(toConsider[1:],
                               avail - item.getWeight(), memo)
        withVal += item.getValue()
        #Consider not taking first item
        withoutVal, withoutToTake = fastSolve(toConsider[1:],
                               avail, memo)
        #Choose better alternative
        if withVal > withoutVal:
            result = (withVal, withToTake + (item,))
        else:
            result = (withoutVal, withoutToTake)
        #Update memo
        memo[(len(toConsider), avail)] = result
    return result

import time, sys
sys.setrecursionlimit(20000)
def test(maxVal = 10, maxWeight = 10, runSlowly = False):
    random.seed(0)
    global numCalls
    capacity = 8*maxWeight
    print '#items, #num taken, Value, Solver, #calls, time'
    for numItems in (4,8,16,32,64,128,256,512,1024):
        Items = buildManyItems(numItems, maxVal, maxWeight)
        if runSlowly:
            tests = (fastSolve, solve)
        else:
            tests = (fastSolve,)
        for func in tests:
            numCalls = 0
            startTime = time.time()
            val, toTake = func(Items, capacity)
            elapsed = time.time() - startTime
            funcName = func.__name__
            print numItems, len(toTake), val, funcName, numCalls, elapsed