CS5800: Algorithms Spring 2018
Assignment 4.3

Saptaparna Das

February 23, 2018

(a) Algorithm:

```
1. Set solution to 0.
2. Pick largest vial size, less than or equal to m.
3. Till vial size is less than m and still vials of that size are in stock, repeat
   the following steps:
        (a) Subtract vial size from m
        (b) Decreament stock size of that vial by 1
        (c) Increament solution by 1
4. If m becomes 0, return solution. Else, repeat step 2 and 3 for new vial.
5. If m doesn't become zero, return -1 to indicate m can not be formed
```

This above algorithm takes an greedy approach by picking largest possible vial everytime. However, there might be scenarios where the algorithm fails. E.g. if m=65 and we have vials of size 5, 20 and 50 avaliable in stock. In this case, greedy will choose 50 first and will never reach 65 if there is only 1 vial available of size 5. But there can be a solution to this (20,20,20,5). So, we can use dynamic programming. Please find below modified algorithm.

```
Let vials[] contain all the available vials and count[] contains their respective quantities.
m is the toal amount of milk.
1. Create empty array dp and track of size m+1
2. Initiatialize dp array witn integer max value and track array with -1.
3. Set dp[0] to 0 and possibleSum to 0.
4. Repeat the following steps for each vial:
        a. Calculate possibleSum as product of vial size and available quantity
        b. Set i to current vial size and do the following steps
                i. if i is greater than or equals current vial size:
                        set 1+ dp[i-current vial size] to dp[i]
                        set track[i] to  index of current vial
        c. if i <= possibleSum and i <= m, increament i by 1 and repeat step i
5. Print actual solution using track array
6. Return -1 if dp[m] = integer max value
   Else return dp[m]

To print actual vials used to make m ounce:

1. If track[m]=-1, print "No solution possible"
2. Set start to track length -1
3. Till start != 0, repeat the following steps:
        a. set j to track[start]
        b. print vials[j]
        c. start = start-vials[j]
```

(b) Proof of correctness:

To prove the above algorithm works correctly, we need to prove following property:

Optimal substructure: To make a total of m ounce of milk, let A_m be the minimum number of vials needed which includes a vial a_k. If we remove a_k from solution, we are left with subproblem to find minimum numbr of vials to make m-a_k ounce. So, A_m can be expressed as $A_(m-a_k) + a_k$. A_m must include optimal solution for $A_(m-a_k)$.
If it's not true then there exists an optimal solution for $A_(m-a_k)$ say $A'_(m-a_k)$. So $A'_(m-a_k) < A_(m-a_k)$. We can add a_k to this new optimal set, since the total amount to make is less by a_k. Hence, we can write, $A'_(m-a_k) + a_k < A_(m-a_k)+a_k$. i.e. $A'_m < A_m$ which is contradiction to original assumption.

So, this problem exhibits optimal substructure property i.e. optimal solution contains optimal solutions of subproblems. We can also see that in every step make a chioce and we are left with one subproblem.So, the two problems are computed indepensently. To get the recurrance relation for this problem, say C(m) is the optimal solution for the problem m ounce milk, which contains vial sizes v1,v2,..,vn. So, in every step we can choose a vial or leave. Suppose last vial is of size v. So, then we have solution to subproblem C(m-v) given v. Since we don't know which vial to choose, we check all possible vials to get minimum number of vials. Hence, C(m) can be written as
$C(m) = \min_{1 \leq i \leq n} C(m - v_i) + 1.$
C(m)=0 when m=0
But the problem here is for all vials C(m) recursively calls itself repeatedly with same parameter values. This exhibits overlapping subproblem property and a need to save results of smaller problems, so that each subproblem is calculated exactly once. This is called bottom-up approach. The idea here is, to calculate a subproblem, all its smaller subproblems are calculated first and saved. Hence, the dp array is created in the algorithm to store values of smaller subproblem. When the array is filled, we get our answer in last element of the array.

Complexity analysis: The algorithm to compute minimum number of vials, iterates through all the vials and checks for all values of sum less than or equals to m. So, total time taken should be O(mv) where v is the number of different vials available.
The auxiliary space taken by the algorithm is proportional to m i.e. the dp array which is used to store minimum number of vials needed for each number less than or equal to m. So, space complexity is O(m).

(c) To know if dispensing the amount is possible, in my algortihm I have calculated total possible amount of milk that can be constructed with each different vial by taking product of vial size and number of vials available of that size. So, along with comparing the current amount with total amount, I am comparing with the max possible amount for that vial also. If the ampunt is not possible last element of the track array won't be populated. I am checking this value at the end to see if m ounce is even possible to dispense.