Assignment -4

3A>

Since the milk bank has limited number of packages we have to dispense the exact amount of requested milk using minimum no of packages. Solving this problem involves following steps

COMPLETE THIS

1. Create an 2 dimensional array R of size to store the number of packets of size k available when the requested amount of milk is i (i.e. R[i][k] represents that)
2. Initialize the first row of the array to hold the available packets of size k initially.
3. Create a one dimensional count array to store the count of the packets dispatched for j ounce milk request where 1<= j <=m. Initialize this array to ∞
4. Create a 2D array of size ((m+1) * 2) where m is the amount of milk to be dispatched. Each row stores the size of the packet dispatched in the first column and the number of packets of that size dispatched in second column.
5. For each size of the package, we check whether the requested amount j where 1<= j<=m is greater than or equal to the package size 0<=k<n , check for the availability of packet of size k in to dispatch p[k] amount of milk by using R[j-P[k]][k] > 0. Also we check if count of packets dispatched for j-P[k] ounces of milk is > 0.
6. If the above step conditions are satisfied then the count is updated to the count of packets dispatched for j-P[k] of milk request + 1. The count count[j] of packet k available will be reduced by 1 for the j-P[k] amount of milk. Store the packet size and its corresponding count in packetSize array.
7. Else The count of packet k available will be set to count of available packets for j-p[k] amount of milk
8. If step 4 is not satisfied then number of available of size k will be reset to initially made available count.
9. Once the above loops are completed for all m ounces of packet dispatch and for all packet sizes 0<=k <n-1, we have count[m] i.e. count of minimum number of packets dispatched for “m” ounces of requested milk and packetSize which has the mapping of the size of the packets dispatched and their corresponding count
10. Return count[m] and packetSize array

Assumption : Given P[] array is increasing order of the packet size . i.e. P = {1, 5, 10, 40, 50} and C[] consists of their count in the same order at the corresponding index positions

Given: an array P which consists of all possible sizes of the vial packages that are used to store milk, number of available of size p[k] initially, m is the amount of milk that needs to be dispensed and n is the size of P array
Returns : an integer which represents the minimum number of packages that needs to be dispensed if it can be dispensed else -1 is returned in that place and a 2D array which stores the packet size and the count of packets of that size dispatched (first column represents the package size and second column represents the count of packets of that size dispatched)

```
COUNT_PACKAGED_MILK(P[],C[], m, n)
{

1       Let R[][] be an array of size (m+1) * n
2       Let count[]  be an array of size m+1
3       count[0] =0

4        for i = 0 to n
5           R[0][i] = C[i]

6        packetSize[m+1][2]    // 2D array store the size of packet and the
         count of packet of that size dispensed

7       for j = 1 to m
8         set count[j]  = ∞
9         for k= 0 to n-1
10              if( j >=P[k] and R[j-P[k]][k] > 0  and count[j-P[k]] != ∞)
11                    if count[j] > count[j-P[k]]+1
12                       count[j] =  count[j-P[k]]+1
13                       R[j][k] = R[j-P[k]][k] -1
14                       packetSize[j][0] = P[k]
15                       packetSize[j][1] = count[j]
16                    else
17                       R[j][k] = R[j-P[k]][k]
```

```
18              else if ( j < P[k])
19                      R[j][k] = R[0][k]

20      if count[m] != ∞
21              return (count[m], packetSize)
22      else
23              return (-1, packetSize)
}
```

3 B> Correctness :

This problem exhibits optimal sub-structure in the following manner. Consider m ounces be the amount of milk that needs to be dispatched. Let p1,p2,p3,p4,p5..,pn be the packet sizes. Now we split this at different at different packet sizes boundary. Suppose Left half is used to dispatch p ounces then right half dispatches (m-p) ounces.

Let C[n] be the minimum no of packets that will be dispatched for "m" ounces of milk request. In order to get an optimal solution there must be a packet of size si
S. T. si <=m which is dispatched to give an optimal solution . The packets dispatched for the remaining (m – si) ounces of milk should also give optimal solution.
This means the c[m] = 1+ c[m-pi] i.e. one packet dispatch and c[m-pi] represents the optimal count for the milk (m-pi)

Since we are not sure if the packet size can be dispatched or not because of packet constraint i.e if no of packets available is zero and size constraint i.e. if packet size > amount of milk to be dispatched. So some packet k will be dispatched whose availability count >0 and size pk <= m. If requested amount of milk is zero then the optimal solution is 0 packets dispatched.

$$c[m] = 0 \quad \text{if } m=0$$

$$c[m] = \min_{i\,:\,pi >= m} (1 + c[m-pi]) \quad \text{if } m > 0$$

In order to prove the correctness we need to consider the algorithm above. We can see that c[m] is set to ∞ at line number 7. Line 9 ensures that the c[m] is changes only if the pk <= m i.e. packet size is less than requested size and if packet available count is > 0 i.e. R[m-P[k]][k] > 0. Later line 10 it is set to a value c[m-p[k]] + 1 if only c[m] > c[m-p[k]] + 1. If assigning one more packet will result in a better solution than existing solution. All this ensures $\min_{i\,:\,pi >= m} (1 + c[m-pi])$ is achieved.

If c [m] is ∞ at line 17 that means we cannot dispatch exact m ounces of milk. So we return -1

Time Complexity :

n -> size of array which consists of packet sizes

m -> amount of milk to be dispatched

Line 4 involves a for loop which runs for a max of n time .
Run time of line 4 and 5= n * c1

Line 7 and 9 involves of two for loop. Outer for loop runs a max of m times and inner for loop runs for a maximum of n time .
Run time of all the statements 7 to 19 = m*n * c2

Run time for line 8 = n * c3

Let run time of all the other statement be some constant = C

Total run time = (n * c1) + (m* n * c2) + (n * c3) + C

Therefore TimeComplexity T(n) = O(mn), Ignoring all lower order terms

Space Complexity :

In this algorithm auxiliary space used involves one 2Dimensional array of size (m*n) is used to record the number of available packets and array of size n to store the count of packets dispatched. Another 2D array of size (m+1) *2 to store the size of packet and the count of packet of that size dispatched

Total space used = (m *n) + n + 2m + 2
Therefore, **Space complexity = O(mn)**

3 c> Given packet of sizes S={1,5,10,20,50} ounces . Let C={ c1,c2,c3, c4, c5}be the count of packets of size 1,5,10,20 and 50 respectively which is available in stock
Initially.

In order to check if we can dispense an exact requested amount “m” of blood, let’s consider an example

Let N be the packets dispensed so far.
If m = 30 S={50,20,10,5,1}, C = { 3,0,2,1,2} and N=0

1>Initially let’s consider a packet of size 50 ounces
s1 =50 , c1 =3 and m = 30

50 > 30 so we ignore packet of this size as it is greater than the required amount

1> Let’s consider the packet of size 20 ounces

s2 = 20, c2 = 0 and m =30

Here 20 < 30 but c2 = 0 so we move to the next since no packets of size 20 ounces is available.

2> Let’s consider the third packet of size 10 ounces

s3= 10, c2 = 2 and m =30

Here 10 < 30 and c3 = 2 >0

NOTE :
If c3 <= m then remaining amount to be dispatched will be
(m – (sn * cn))
else the remaining amount to be dispatched will be
(m – (sn * (m/sn)))

Remaining amount of milk to be dispatched(m) = 30 – (10 *c3)

m = 30 –(10*2) = 10

3> Now Let’s consider packets of size 5 ounces

S4 = 5, c4 = 1, m =10

Here 5 < 10 and c4 = 1>0

Remaining amount of milk be dispatched(m) = (10 – (5 *1))

m = 5

4> Now Let's consider packets of size 1 ounces

S5 = 1, c5 = 2, m = 5

Here 1 < 5 and c5 = 2
Remaining amount of milk to be dispatched(m) = (5 – (1 * 2))

=> m = 3

5> Now we are not left with any packets of size 3 or less than 3 to dispatch.

In this case we return "not possible" as it is impossible to exact amount of milk for the requested amount of milk with the available packets in stock.

This above explanation can be tested using the algorithm below :

Given :
m -> requested amount of milk
n -> length of array s
s[] -> array which holds all the available packet sizes
c[] ->array to hold the count of packets of the sizes s[i] at index position i

Returns : true if exact amount can be dispatched else returns false

```
isDispatched(s[],c[],m, n)
{
        for ( i = 0 to n)
        {
                if ( m % s[i] == 0)
                {
                        if (  c[i]  > (m / s[i])
                                m = m – ( m / s[i]))
                        else
                                m = m – (c[i] *s[i])
                }
        }
        return (m == 0) ? true : false
```

}