Consider the following problem. An organizer of a running race wishes to purchase (at least) n identical T-shirts (same size) for participants. At a wholesale, T-shirts are packaged at different prices depending on the number of T-shirts in the package. The organizer can buy any number of packages of each size (size here is not the T-shirt size, it is the number of T-shirts in a package), as long as the total number is at least n. The organizer wishes to find the minimum total price of such a set of packages. The input is given as n and an array Packages[1..m], where each Package[i] has a positive integer field Package[i].size and a positive real field Package[i].price, giving the number of T-shirts in the package and the price of the package, respectively.

A recursive algorithm to solve this problem is:

BestPrice[n : positiveinteger, Packages[1..m] : array of pairs (size: integer, price: real).]

1. MinPrice ← inf;

2. For d=1 to m do:

3. begin;

4. IF Packages[d].size ≥ n THEN TempPrice ← Packages[d].price

5. ELSE TempPrice ← Packages[d].price + BestPrice(n − Packages[d].size, P ackages);

6. IF TempPrice < MinPrice THEN MinPrice ← TempPrice;

7. end;

8. Return MinPrice.

**Part 1: Show the recursion tree of the above algorithm on the following input: n = 6, packages: buy 5 for $ 24, 3 for $10 or 2 for $8.**

Case 1: Buy package of 5 for $24+BestPrice(1,Package)

We already have 5 packages and for one more package below three are the possibilities

1. Buy a package of 5 for $24 🡪5 + 5 = 10 packages, total $48
2. Buy a package of 3 for $10 🡪5 + 3 = 8 packages, total $34
3. Buy a package of 2 for $8🡪 5 + 2 = 7 packages, total $32

Minimum = 6

Therefore, case-1 will return minimum packages of 7 (5+2) with best price $32.

Case 2: Buy package of 3 for $10+ BestPrice(3, Packages)

We already have 3 packages and for one or more packages below are the possibilities

1. Buy package of 5 for $24 🡪3 + 5 = 8 packages, total $34.
2. Buy package of 3 for $10 🡪3+ 3 = 6 packages, total $20.
3. Buy package of 2 for $8 🡪3+2 = 5 packages, total $18.(Below are the possibilities for remaining one package)
   1. Buy package of 5 for $24 🡪5+5 = 10 packages for $42
   2. Buy package of 3 for $10 🡪5 +3 = 8 packages for $28
   3. Buy package of 2 for $8 🡪 5+2 = 7 packages for $26

Minimum = 6

Therefore, case-2 will return minimum packages of 6 (3+3) with best price $20.

Case 3: Buy package of 2 for $8+ BestPrice(4, Packages)

We already have 2 packages and for one or more packages below are the possibilities

1. Buy package of 5 for $24 🡪 2 + 5 = 7 packages for $32
2. Buy package of 3 for $10 🡪 2 + 3 = 5 packages for $18 (Below are the possibilities for remaining one package)
   1. Buy package of 5 for $24 🡪5+5 = 10 packages for $42
   2. Buy package of 3 for $10 🡪5 +3 = 8 packages for $28
   3. Buy package of 2 for $8 🡪 5+2 = 7 packages for $26
3. Buy package of 2 for $10 🡪 2 + 2 = 4 packages for $16 (Below are the possibilities for remaining one package)
   1. Buy package of 5 for $24 🡪4+5 = 9 packages for $40
   2. Buy package of 3 for $10 🡪4 +3 = 7 packages for $26
   3. Buy package of 2 for $8 🡪 4+2 = 6 packages for $24

Minimum =6

Therefore, case-3 will return minimum packages of 6 (2+2+2) with best price $24

Finally, after overall calculation we could see that case 2 has best price for the packages that is 3+3 for $20.

**Part 2: Give a bound on the worst-case number of recursive calls the recursive algorithm could make in terms of n and m.**

* It is O(mn) because there are m recursive calls for each n.

**Part 3: Give a dynamic programming version of the recurrence.**

Packages[m] {Size; Price;} // An array of packages

OP[n]; // temporary array containing minimum price

minPrice=0;

tempPrice=0;

for i= 1 to n do

{

minPrice= 0;

for j= 1 to m do

{

if Pakages[j].size >=i

{

then tempPrice = Packages[j].price;

}

Else

{

tempPrice= Packages[j].price+ OP[n-Packages[j].size];

}

if tempPrice < minPrice then

{ minPrice = tempPrice; }

}

OP[i] = minPrice;

}

Return OP[n]

**Part 4: Give a time analysis of this dynamic programming algorithm, in terms of n and m.**

* We have 2 nested loops in the dynamic algorithm, one going from 1 to n and the other from 1 to m. This gives us an overall time of O(nm).

**Part 5: Show the array that your algorithm produces on the above example.**

We have array of Packages and their cost: 5 for $ 24, 3 for $10 or 2 for $8

Here number of minimum packages are 6 then n=6 and OP(n) store the best price at each step

1. When i=1, it is only one package, so the Min(8, 10, 24) therefore OP[1] = 8
2. When i=2, we have Minimum (24, 10, 8), therefore OP[2] = 8
3. When i= 3, Minimum (OP [2] +8, 10, 24) = Minimum (16, 10, 24) =10, therefore OP[3] = 10
4. When i= 4, Minimum (OP [3] +8, OP[3]+ 10,24) =Minimum(16,18,24)= 16, therefore OP[4] = 16
5. When i= 5, Minimum (OP [4]+ 10,24) =Minimum(18,24)= 18, therefore OP[5] = 18
6. When i= 6, Minimum (OP [5] + 8, OP [5] + 10, OP [5] + 24) =Minimum (24, 20, 32) = 20, therefore OP[6] = 20

