Esercizio. È dato un insieme $S = \{s_1, s_2, ..., s_n\}$ di n numeri naturali ed un numero naturale X. Vogliamo capire se esiste un sottoinsiemi di valori $V \subseteq S$ tale che la sommatoria dei valori in V sia uguale a X, ovvero $\sum_{s \in V} s = X$.

Soluzione. Questo problema può essere risolto utilizzando programmazione dinamica considerando i problemi $P(i, j)$, con $i \in \{1, \dots, n\}$ e $j \in \{0, \dots, X\}$, che consistono nel verificare se esiste un sottoinsieme dell'insieme $\{s_1, s_2, ..., s_i\}$ con sommatoria j. Si noti che il problema iniziale corrisponde con $P(n, X)$. Tali problemi possono essere risolti procedendo induttivamente rispetto a i:

$$P(1, j) = \begin{cases} \mathit{true} & \text{se } j = 0 \vee j = s_1 \\ \mathit{false} & \text{altrimenti} \end{cases}$$

e per $i > 1$:

$$P(i, j) = \begin{cases} P(i-1, j) & \text{se } s_i > j \\ P(i-1, j) \vee P(i-1, j - s_i) & \text{altrimenti} \end{cases}$$

Il seguente algoritmo risolve il problema iniziale risolvendo tutti i problemi $P(i, j)$, salvando le soluzioni già trovate nella tabella di booleani $B[1..n, 0..X]$ e restituendo alla fine il valore $B[n, X]$.

Algorithm 1: SubsetSum(*Natural* $S[1..n]$, *Natural* X) → *Boolean*

```
Boolean B[1..n, 0..X]
for j = 0 to X do
    if (j == 0 or j == S[1]) then
        B[1, j] = true
    else
        B[1, j] = false
for i = 2 to n do
    for j = 0 to X do
        if S[i] > j then
            B[i, j] = B[i − 1, j]
        else
            B[i, j] = B[i − 1, j] or B[i − 1, j − S[i]]
return B[n, X]
```

Il costo computazionale di tale algoritmo risulta essere $T(n, X) = \Theta(n \times X)$ in quanto prevede di riempire tutte le celle della tabella B e ogni riempimento richiede tempo costante.

Esercizio. Si consideri il problema del resto nel caso generale, ovvero non assumendo di avere a disposizione monete di un sistema canonico. Più precisamente si deve stampare la combinazione con il numero minimo di monete, se esiste, per raggiungere il resto R considerando come possibili valori delle monete quelli indicati nell'array di naturali $T[1..n]$.

Soluzione. Ci occupiamo innanzitutto di come calcolare il minimo numero di monete. Si può utilizzare programmazione dinamica considerando i problemi $P(i, j)$, con $i \in \{1, \dots, n\}$ e $j \in \{0, \dots, R\}$, che consistono nel calcolare il numero minimo di monete per ottenere il resto j usando solamente monete aventi valore in $T[1..i]$. Per comodità si utilizza $P(i, j) = \infty$ per indicare che non esiste una combinazione di monete di valore in $T[1..i]$ per ottenere il resto j. Tali problemi possono essere risolti procedendo induttivamente rispetto a i e rispetto a j:

$$P(1, j) = \begin{cases} 0 & \text{se } j = 0 \\ j \;/\; T[1] & \text{se } (j \bmod T[1] = 0) \\ \infty & \text{altrimenti} \end{cases}$$

e per $i > 1$:

$$P(i,j) \;=\; \begin{cases} P(i-1,j) & \text{se } T[i] > j \\ min\{P(i-1,j), 1+P(i,j-T[i])\} & \text{altrimenti} \end{cases}$$

Il seguente algoritmo inizialmente risolve tutti i problemi $P(i,j)$, salvando le soluzioni già trovate nella tabella $M[1..n, 0..R]$. Ogni volta che si riempie una cella $M[i,j]$ si capisce se la soluzione al problema $P(i,j)$ utilizza oppure no almeno una moneta di taglio $T[i]$; si salva tale informazione in una tabella di booleani $B[1..n, 0..R]$ che viene poi utilizzata per stampare la combinazione ottimale di monete, se esiste.

Algorithm 2: RESTO(*Natural R*, *Natural T*[1..n])

```
Natural M[1..n, 0..R]
Boolean B[1..n, 0..R]
// Riempimento tabella per programmazione dinamica M e della corrispondente tabella B
M[1,0] = 0 ;  B[1,0] = false
for j = 1 to X do
    if R mod T[1] == 0 then
        M[1,j] = R div T[1] ;  B[1,j] = true
    else
        M[1,j] = ∞ ;  B[1,j] = false
for i = 2 to n do
    for j = 0 to R do
        if T[i] > j then
            M[i,j] = M[i-1,j] ;  B[i,j] = false
        else
            if M[i-1,j] < 1 + M[i,j-T[i]] then
                M[i,j] = M[i-1,j] ;  B[i,j] = false
            else
                M[i,j] = 1 + M[i,j-T[i]] ;  B[i,j] = true
// Stampa della soluzione utilizzando la matrice di booleani B
if M[n,R] == ∞ then
    PRINT("Resto impossibile")
else
    Natural residuo = R, tot
    for i = n downto 1 do
        // Calcolo del numero di monete di taglio T[i] da utilizzare
        tot = 0
        while B[i, residuo] do
            residuo = residuo - T[i]
            tot = tot + 1
        PRINT(tot + " monete di taglio " + T[i])
```

Il costo computazionale è dominato dalla parte di scrittura della tabella di programmazione dinamica M e della corrisponde tabella di booleani B, in quanto la parte di stampa semplicemente legge solo alcune celle della tabella di booleani B. Quindi, il costo risulta essere $T(n,R) = \Theta(n \times R)$ in quanto il riempimento di ogni cella della tabella M richiede tempo costante.