Esercizio 1

Dati n interi positivi $x_1, x_2, \ldots x_n$ e due interi M e k vogliamo sapere se è possibile ottenere M come somma di al più k addendi scelti tra gli n numeri disponibili utilizzando eventualmente anche più volte gli stessi numeri.

Esempio. dati i numeri $2, 5, 7, 8, 10$ e $k = 3$:

- per $M = 26$ la risposta è SI ($8 + 8 + 10 = 26$)
- per $M = 12$ la risposta è SI ($5 + 7 = 12$ e $5 + 5 + 2 = 12$ e $2 + 10 = 12$ e $2 + 2 + 8 = 12$)
- per $M = 3$ la risposta è NO.

1. *(max 10 punti)* Proporre un algoritmo che risolve il problema in tempo $O(Mn)$.
2. *(max 10 punti)* Modificare l'algoritmo proposto al punto 1 in modo tale che se la risposta è SI in output vengono restituiti gli al più k addendi la cui somma è pari ad M (nell'istanza dell'esempio quando $M = 12$ deve restituire $(5, 7)$ o $(2, 5, 5)$ o $(2, 10)$ o $(2, 2, 8)$. La modifica deve avere costo additivo $O(M + n)$.

Soluzione Esercizio 1 Si mantenga un vettore T di dimensione M tale che

$$T[i] = \begin{cases} \text{il numero minimo di addendi necessari per ottenere } i \text{ utilizzando numeri in } \{x_1, x_2, \ldots x_n\} \\ +\infty \text{ se non è possibile ottenere } i \text{ dalla somma di numeri in } \{x_1, x_2, \ldots x_n\} \end{cases}$$

Per risolvere il problema originario basta controllare che $T[M] \leq k$.
Per calcolare i valori di $T[i]$ per cui $1 \leq i \leq M$ si può utilizzare la seguente regola:

$$T[i] = \begin{cases} +\infty & \text{se } i < \min\{x_1, \ldots x_n\} \\ 1 & \text{se } i \in \min\{x_1, \ldots x_n\} \\ 1 + \min_{1 \leq j \leq n, x_j < i}\{T[i - x_j]\} & \text{altrimenti} \end{cases}$$

E' facile calcolare i valori del vettore T spendendo tempo $O(n)$ su ogni cella e quindi ottenendo un tempo di esecuzione $O(M \cdot n)$. Lo pseudo-codice è il seguente:

```
SOMMA:
  INPUT gli interi M e k e i numeri x_1, ... x_n
  OUTPUT un valore booleano
    FOR i = 1 TO M DO
    T[i] ← +∞
        FOR j = 1 TO n DO
            IF x_j = i THEN T[i] ← 1
            IF x_j < i AND T[i] > T[i − x_j] + 1 THEN T[i] ← T[i − x_j] + 1
        ENDFOR
    ENDFOR
    IF T[M] ≤ k OUTPUT SI
    ELSE OUTPUT NO
```

Dati i valori del vettore T, gli addendi della somma che produce il valore M possono essere facilmente individuati in tempo $O(M + n)$ come segue.

ADDENDI:
  INPUT gli interi M e k, i numeri $x_1, \dots x_n$ e la matrice T
  OUTPUT la sequenza di al più k addendi che produce M
    $t \leftarrow 0$
    $j \leftarrow 1$
    WHILE $M > 0$ DO
        WHILE $(x_j > M)$ OR $(x_j \neq M$ AND $T[M - x_j] + 1 \neq T[M])$ DO $j \leftarrow j + 1$
        $t \leftarrow t + 1$
        $SOL[t] \leftarrow x_j$
        $M \leftarrow M - x_j$
    ENDWHILE
    OUTPUT $SOL[1], SOL[2], \dots SOL[t]$

Esercizio 2 *(max 10 punti)* Scrivere in pseudo-codice una procedura che preso in input un intero n stampi tutte le permutazioni dei numeri $\{1, 2, \ldots, n\}$ con la proprietà che nelle posizioni pari della permutazione risultano posizionati numeri pari. . Ad esempio per $n = 4$ la procedura deve stampare (non necessariamente in quest'ordine): 1 2 3 4, 1 4 3 2, 3 2 1 4, 3 4 1 2.
La complessità della procedura **deve essere** $O(n^2 \cdot D(n))$, dove $D(n)$ è il numero di permutazioni da stampare.

Soluzione Esercizio 2 Un algoritmo che risolve il problema e che si basa sulla tecnica del backtracking è il seguente. Per assicurarsi che la complessità sia proporzionale al numero $D(n)$ di permutazioni da stampare bisogna assicurarsi che una chiamata ricorsiva sarà effettuata se e solo se la soluzione parziale fino a quel momento costruita può estendersi ad una soluzione da stampare. Per ottenere questa proprietà basta assicurarsi che nell'aggiungere l'i-esimo numero alla permutazione questo numero non compaia nella permutazione parziale gia costruita e che sia un numero pari se i lo è, dispari altrimenti.
L'algoritmo ABC, essendo ricorsivo, prende in input l'intero n, il numero i che indica la posizione in cui inserire l'elemento, il vettore SOL in cui viene memorizzata la permutazione da stampare e il vettore booleano P che indica quali degli n numeri sono presenti nella permutazione parziale. La prima chiamata sarà ABC(n, 1, SOL, P) con il vettore P contenente tutti zeri. Lo pseudo-codice dell'algoritmo è il seguente:

```
ABC:  INPUT gli interi n e i, i vettori SOL e P
    IF i > n THEN stampa la sequenza SOL[1], SOL[2], ..., SOL[n]
    ELSE
        FOR j = 1 TO n DO
            IF (P[j] = 0) AND (i mod 2 = j mod 2) THEN
                SOL[i] ← j
                P[j] ← 1
                ABC(n, i + 1, SOL, P)
                P[j] ← 0
            ENDIF
        ENDFOR
    ENDIF
```