Esercizio 1

Dati n interi positivi $x_1, x_2, \ldots x_n$ e due interi $M \geq 0$ e $t \geq 0$, vogliamo sapere se è possibile ottenere M come somma di al più t degli n numeri.

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

- per $M = 0$ la risposta è SI (grazie al fatto che 0 addendi danno somma 0)
- per $M = 12$ la risposta è SI (grazie al fatto che $2 + 10 = 12$ o anche $5 + 7 = 12$)
- per $M = 14$ la risposta è SI (grazie al fatto che $2 + 5 + 7 = 14$)
- per $M = 6$ la risposta è NO

1. *(max 10 punti)* Proporre un algoritmo che risolve il problema in tempo $O(M \cdot n)$.
2. *(max 10 punti)* Modificare l'algoritmo proposto al punto 1 in modo che se la risposta è SI in output vengono restituiti gli al più t numeri la cui somma risulta M. La modifica deve avere costo additivo $O(n)$.

Soluzione Esercizio 1 Si mantenga una matrice T di dimensione $M \times n$ tale che

$$T[i,j] = \begin{cases} \text{numero minimo di addendi per ottenere il valore } i \text{ dai primi } j \text{ numeri} \\ +\infty \text{ se non è possibile ottenere il valore } i \text{ sommando elementi dei primi } j \text{ numeri} \end{cases}$$

Per risolvere il problema originario basterà controllare che $T[M, n] \leq t$.
Ovviamente vale:

- $T[0, j] = 0$ per $j \geq 0$ (il valore 0 può ottenersi con 0 addendi)
- $T[i, 0] = +\infty$ per $i \geq 1$ (disponendo di 0 numeri non posso ottenere alcuna somma positiva)

Per calcolare i valori di T per cui $1 \leq i \leq M$, $1 \leq j \leq n$ si può utilizzare la seguente regola:

$$T[i,j] = \begin{cases} T[i, j-1] & \text{se } i < x_j \\ \max\{T[i, j-1], T[i - x_j, j-1] + 1\} & \text{altrimenti} \end{cases}$$

E' facile calcolare i valori della matrice T spendendo tempo $O(1)$ 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 t e i numeri x_1, ... x_n
  OUTPUT un valore booleano
    FOR j = 0 TO n DO T[0, j] ← 0
    FOR i = 1 TO M T[i, 0] ← +∞
    FOR i = 1 TO M DO
        FOR j = 1 TO n DO
            T[i, j] ← T[i, j − 1]
                IF x_j ≤ i AND T[i−x_j, j−1]+1 < T[i, j−1] THEN T[i, j] ← T[i−x_j, j−1]+1
        ENDFOR
    ENDFOR
    IF T[M, n] > t THEN OUTPUT NO
        ELSE OUTPUT SI
```

Dati i valori della matrice T, gli al più t interi che danno come somma M possono essere facilmente individuati in tempo $O(n)$ procedendo dall'ultima verso la prima cella della matrice.

ELEMENTI:
  INPUT gli interi M e t, i numeri $x_1, \ldots x_n$ e la matrice T
  OUTPUT la sequenza di al più t numeri la cui somma vale M
    $k \leftarrow 0$
    $i \leftarrow M$
    $j \leftarrow n$
    WHILE $T[i, j] > 0$ DO
        IF $x_j \leq i$ AND $T[i - x_j, j - 1] + 1 = T[i, j]$ THEN
            $k \leftarrow k + 1$
            $SOL[k] \leftarrow x_j$
            $i \leftarrow i - x_j$
        ENDIF
        $j \leftarrow j - 1$
    ENDWHILE
    OUTPUT $SOL[1], SOL[2], \ldots SOL[k]$

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 nella permutazione le cifre dispari precedono le cifre pari. . Ad esempio per $n = 4$ la procedura deve stampare (non necessariamente in quest'ordine): 1 3 2 4, 1 3 4 2, 3 1 2 4, 3 1 4 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 legare la complessità 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 inserire nelle posizioni che vanno da 1 a $\lceil \frac{n}{2} \rceil$ solo numeri dispari non ancora inseriti mentre nella posizioni che vanno da $\lceil \frac{n}{2} \rceil + 1$ ad n vanno inseiriti solo numeri pari non ancora inseriti.
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, il vettore booleano P che indica quali degli n numeri sono presenti nella permutazione parziale. La prima chiamata sarà ABC(n, k 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
        j ← 1
        IF i > ⌈n/2⌉ THEN j ← 2
        WHILE j ≤ n DO
            IF P[j] = 0 THEN
                SOL[i] ← j
                P[j] ← 1
                ABC(n, i + 1, SOL, P)
                P[j] ← 0
            ENDIF
            j ← j + 2
        ENDWHILE
    ENDIF
```