Esercizio 1 Date due stringhe di n caratteri $X = x_1x_2\dots x_n$ e $Y = y_1y_2\dots y_n$ ed una terza stringa di $2\cdot n$ caratteri $Z = z_1z_2\dots z_{2n}$, la stringa Z è il *mescolamento* di X e Y se Z si può ottenere intrecciando i caratteri di X e quelli di Y in modo da preservare comunque l'ordinamento dei caratteri di ciascuna delle due stringhe.
Ad esempio: $Z = a\mathbf{bbc}a\mathbf{c}ba$ è il mescolamento di $X = \mathbf{bbcc}$ e $Y = aaba$.
Un mescolamento può essere specificato tramite un vettore bianario di $2\cdot n$ componenti, dove nell'i-esima troviamo x o y a seconda che il carattere z_i nel mescolamento corrisponda ad un carattere di X o ad un carattere di Y.
Ad esempio: per $X = \mathbf{bbcc}$, $Y = aaba$ e $Z = a\mathbf{bbc}a\mathbf{c}ba$, il mescolamento è determinato da $SOL = yxxxyxyy$.

1. *(max 16 punti)* Descrivere in pesudo–codice un algoritmo che, date due stringhe di n caratteri X, Y ed una stringa di $2\cdot n$ caratteri Z, determina se Z è il mescolamento di X e Y. L'algoritmo deve essere **ben motivato** ed avere complessità $O(n^2)$.

2. *(max 7 punti)* Modificare l'algoritmo proposto in modo che, con costo addittivo $O(n)$, in caso Z sia il mescolamento di X e Y, venga prodotto in output un mescolamento.

Soluzione Esercizio 1

a) Usiamo il metodo della programmazione dinamica.

Per ogni $0 \le i \le n$ e $0 \le j \le n$ sia $N(i,j)$ pari a *vero* o *falso* a seconda che $Z = z_1z_2\dots z_{i+j}$ sia o meno il mescolamento di $X = x_1x_2\dots x_i$ e $Y = y_1y_2\dots y_j$. Il calcolo degli $N(i,j)$ è facile:

Per ogni $i,j \ge 1$

$$N(i,j) = \begin{cases} vero & \text{se } N(i,j-1) = vero \text{ e } z_{i+j} = y_j \text{ oppure } N(i-1,j) = vero \text{ e } z_{i+j} = x_i \\ falso & \text{altrimenti} \end{cases}$$

come condizioni iniziali si ha $N(0,0) = vero$ e

$$N(0,j) = \begin{cases} vero & \text{se } N(0,j-1) = vero \text{ e } z_j = y_j \\ falso & \text{altrimenti} \end{cases} \qquad N(i,0) = \begin{cases} vero & \text{se } N(i-1,0) = vero \text{ e } z_i = x_i \\ falso & \text{altrimenti} \end{cases}$$

Ovviamente la soluzione al nostro problema è data da $N(n,n)$. Quindi l'algoritmo può essere così descritto:

```
MESCOLAMENTO:  INPUT tre vettori X, Y di n componenti e Z di 2·n componenti.
    N[0,0] ← vero
    FOR i ← 1 TO n DO
        N[i,0] ← falso
        N[0,i] ← falso
        IF N[i−1,0] = vero AND Z[i] = X[i] THEN N[i,0] ← vero
        IF N[0,i−1] = vero AND Z[i] = Y[i] THEN N[0,i] ← vero
    ENDFOR
    FOR i ← 1 TO n DO
        FOR j ← 1 TO n DO
            N[i,j] ← falso
            IF (N[i,j−1] = vero AND Z[i+j] = Y[j]) OR (N[i−1,j] = vero AND Z[i+j] = X[i]) THEN N[i,j] ← vero
        ENDFOR
    ENDFOR
OUTPUT N[n,n].
```

La complessità dell'algoritmo è determinata dai due 'FOR' nidificati ed è quindi $O(n^2)$.

b) Come è tipico degli algoritmi basata sulla tecnica della programmazione dinamica, per ricostruire la soluzione a partire dal suo valore basta tenere traccia durante l'esecuzione dell'algoritmo delle decisioni che hanno portato a determinare il valore della soluzione. In questo caso si può utilizzare una matrice P $(n+1) \times (n+1)$ dove, per ogni $i, j \geq 1$

$$P[i,j] = \begin{cases} x & \text{se } N[i-1,j] = \mathit{vero} \text{ AND } Z[i+j] = X[i] \\ y & \text{se } N[i,j-1] = \mathit{vero} \text{ AND } Z[i+j] = Y[i] \\ \text{indefinito} & \text{altrimenti} \end{cases}$$

mentre

$$P[0,j] = \begin{cases} y & \text{se } N[0,j-1] = \mathit{vero} \text{ e } Z[j] = Y[j] \\ \mathit{indefinito} & \text{altrimenti} \end{cases}$$

$$P[i,0] = \begin{cases} x & \text{se } N[i-1,0] = \mathit{vero} \text{ e } Z[i] = X[i] \\ \mathit{indefinito} & \text{altrimenti} \end{cases}$$

In questo modo per ottenere il mescolamento basterà scorrere all'indietro la matrice P, aggiungendo all'algoritmo le istruzioni:

```
    k ← 2 · n
    i ← n
    j ← n
    WHILE k > 0 DO
        SOL[k] ← P[i, j]
        IF P[i, j] = x THEN i ← i − 1
        ELSE j ← j − 1
        k ← k − 1
    ENDWHILE
OUTPUT SOL.
```

ed il costo addittivo dell'ultimo WHILE dell'algoritmo è $O(n)$.

Esercizio 2 *(max 7 punti)* Scrivere in pseudo–codice una procedura che preso in input un intero positivo n, stampi tutti i sottoinsiemi di $\{1, 2, \dots 2 \cdot n\}$ che contengono esattamente n interi e non contengono interi consecutivi. Ad esempio se $n = 3$ allora la procedura deve stampare (non necessariamente in quest'ordine): $\{1, 3, 5\}$, $\{1, 3, 6\}$, $\{1, 4, 6\}$, $\{2, 4, 6\}$. La complessità della procedura **deve essere** $O(n^2 D(n))$, dove $D(n)$ è il numero di sequenze da stampare.

Soluzione Esercizio 2 Un algoritmo che risolve il problema e che si basa sulla tecnica del backtracking è il seguente. L'algoritmo ABC, essendo ricorsivo, prende in input l'intero n, il vettore SOL in cui viene memorizzata la sequenza da stampare, il numero i di elementi inseriti nella sequenza fino a questo momento, Quindi la prima chiamata sarà ABC(n, SOL, 0).

ABC: INPUT l'intero n, il vettore SOL e l' intero i
  IF $i = n$ THEN stampa la sequenza $SOL[1], SOL[2], \dots, SOL[n]$
  ELSE
    IF $i = 0$ THEN $x \leftarrow 1$
        ELSE $x \leftarrow SOL[i] + 2$
    WHILE $n - (i + 1) \leq \left\lceil \frac{2n-x}{2} \right\rceil$ DO
      $SOL[i + 1] \leftarrow x$
      ABC(n, SOL, $i + 1$)
      $x \leftarrow x + 1$
    ENDWHILE
  ENDIF

Nelle soluzioni parziali gli elementi sono inseriti in ordine crescente. Onde evitare la generazione di insiemi inutili prima di inserire come $i + 1$-esimo elemento l'intero x si controlla che questo non sia consecutivo all'ultimo elemento inserito e che ci siano interi non adiacenti a valore maggiore di x sufficienti a completare la soluzione.
Per quanto detto una chiamata ricorsiva sarà effettuata se e solo se la soluzione parziale fino a quel momento costruita può estendersi ad una soluzione da stampare. Il costo di un nodo dell'albero di decisione è $O(n)$ e l'altezza dell'albero è n. La complessità risultante sarà $O(n^2 \cdot D(n))$.