Esercizio 1 Si consideri la seguente variante del problema di SELEZIONE DI ATTIVITA'visto a lezione. Si hanno n attività ciascuna delle quali è caratterizzata da un valore e dall'intervallo temporale in cui deve svolgersi. Bisogna trovare un sottoinsieme di attività *compatibili* che abbia valore massimo. Si ricorda che le attività sono compatibili se hanno intervalli temporali disgiunti. Il valore del sottoinsieme è dato dalla somma dei valori delle sue attività. Si assuma che l'i-esima attività $1 \leq i \leq n$ è data dalla terna (s_i, t_i, v_i) dove (s_i, t_i) è l'intervallo temporale in cui l'attività i deve svolgersi mentre v_i è il suo valore. Si assuma pure, per semplicità, che $t_1 \leq t_2, \ldots \leq t_n$.
Ad esempio: date le 5 attività $(1,4,4), (3,6,10), (5,10,9), (7,12,6), (11,13,2)$ la soluzione è $\{2,4\}$ che ha valore 16.

1. *(max 16 punti)* Descrivere in pesudo–codice un algoritmo che calcola il valore della soluzione. 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 $O(n)$, produca la soluzione.

Soluzione Esercizio 1

a) Usiamo il metodo della programmazione dinamica. Per ogni $i = 1, 2, \ldots, n$ sia $N(i)$ il valore della soluzione se si hanno le sole attività $\{1, 2, \ldots i\}$ e si vuole che la soluzione contenga l'attività i. Il calcolo degli $N(i)$ è facile:

Per ogni $i = 1, 2, \ldots, n$

$$N(i) = \begin{cases} v_i & \text{se non ci sono in } \{1,2,\ldots i-1\} \text{ attività compatibili con } i \\ N(j) + v_i & \text{dove } j \text{ è l'attività in } \{1,2,\ldots,i-1\} \text{ compatibile con } i \text{ che ha } N(j) \text{ massimo} \end{cases}$$

ovviamente il valore della soluzione del nostro problema è il massimo tra i valori $N(1), N(2), \ldots, N(n)$. Quindi l'algoritmo può essere così descritto:

ATTIVITA': INPUT tre vettori di n componenti S, T e V contenenti i tempi di inizio, tempi di fine e i valori delle n attività, inoltre si ha $T[1] \leq T[2] \leq \ldots \leq T[n]$.

```
N[1] ← V[1]
pmax ← 1
FOR i ← 2 TO n DO
    j ← 0
    vmax ← 0
    k ← 1
    WHILE T[k] < S[i] DO
        IF vmax < N[k] THEN
            vmax ← N[k]
            j ← k
        ENDIF
        k ← k + 1
    ENDWHILE
    IF j = 0 THEN N[i] ← V[i]
             ELSE N[i] ← N[j] + V[i]
    IF N[pmax] < N[i] THEN pmax ← i
ENDFOR
OUTPUT N[pmax].
```

La complessità dell'algoritmo è determinata dal 'FOR' e dal WHILE nidificati e quindi è limitata da $O(n^2)$.

b) Come al solito 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 un vettore P ad n componenti dove in $P[i]$ si mette i se $N[i] = V[i]$ mentre si mette j se $N[i] = N[j] + V[i]$. In questo modo per ricostruire la soluzione basterà scorrere all'indietro il vettore P. Quindi bastera' poi aggiungere all'algoritmo le istruzioni:

$SOL \leftarrow \emptyset$
WHILE $P[pmax] \neq pmax$ DO
    $SOL \leftarrow SOL \cup \{P[pmax]\}$
    $pmax \leftarrow P[pmax]$
ENDWHILE
$SOL \leftarrow SOL \cup \{pmax\}$
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 $\{1, 2, \ldots 2 \cdot n\}$ 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** $\mathrm{O}(n^2 D(n))$, 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], ..., SOL[n]
    ELSE
        IF i = 0 THEN x ← 1
                 ELSE x ← SOL[i] + 2
        WHILE n − (i + 1) ≤ ⌈(2n − x)/2⌉ DO
            SOL[i + 1] ← x
            ABC(n, SOL, i + 1)
            x ← 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 $\mathrm{O}(n^2 \cdot D(n))$.