Esercizio 1 (9 punti) Realizzare una procedura `triplet(A)` che dato un array `A[1,n]` di interi verifica se esistono tre indici, non necessariamente distinti, `i`, `j` e `k` tali che `A[i]+A[j]=A[k]`. Fornire lo pseudocodice, motivare la correttezza della soluzione e valutarne la complessità.

// Assumendo l'array sia ordinato

$triplet(A)$

$n = a.length$

$while(i\ AND\ j\ AND\ k \leq n\ \&\&\ A[i] + A[j]\ != k)$

    // Controllare tutti gli indici

    I + j <= k

        I++;

        j++;

    // Altri casi per tutti gli indici

Return (i,j,k)

Correttezza:

- I+j<=k

Per ogni "i" e "j" → i + j <=k

Vale sempre (invariante)

Mi fermo una volta trovati gli indici

Relazione di ricorrenza:

```
LCS[i][j] = {
   0                    se i = 0 o j = 0
   LCS[i-1][j-1] + 1        se X[i] = Y[j]
   max(LCS[i-1][j], LCS[i][j-1])  altrimenti
}
```

Algoritmo bottom-up:

```
funzione LCS(X, Y)
   m := lunghezza(X)
   n := lunghezza(Y)
   LCS := matrice[0..m][0..n]

   // Inizializzazione
   per i da 0 a m
      LCS[i][0] := 0
   per j da 0 a n
      LCS[0][j] := 0

   // Riempimento della matrice
   per i da 1 a m
      per j da 1 a n
         se X[i] = Y[j] allora
            LCS[i][j] := LCS[i-1][j-1] + 1
         altrimenti
            LCS[i][j] := max(LCS[i-1][j], LCS[i][j-1])

   restituisci LCS[m][n]

fine funzione
```

Esercizio (11 punti) Si consideri il problema della selezione delle attività compatibili. Si ha un insieme S di n attività, dove ogni attività ai ha un tempo di inizio si e un tempo di fine fi. Due attività ai e aj sono compatibili se i loro intervalli di tempo non si sovrappongono, ovvero se fi ≤ sj o fj ≤ si. L'obiettivo è selezionare il sottoinsieme di attività compatibili di cardinalità massima.

(a) Qual è la complessità dell'algoritmo esaustivo che esamina tutti i possibili sottoinsiemi di attività?

(b) Assumendo di conoscere un algoritmo che determina se due attività sono compatibili in tempo O(1), come si può modificare l'algoritmo del punto precedente per renderlo più efficiente?

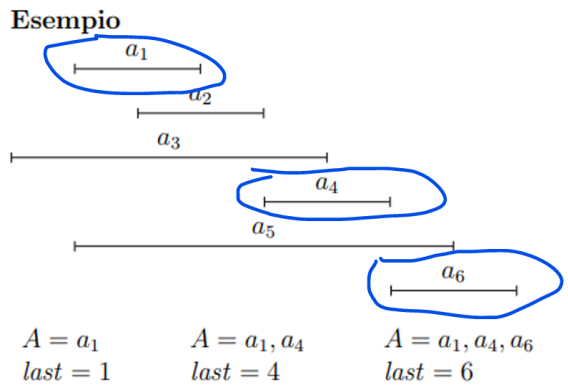
(c) Progettare un algoritmo greedy efficiente per risolvere il problema. Sono richiesti:

- La strategia greedy utilizzata
- Lo pseudocodice dell'algoritmo
- La dimostrazione della correttezza (proprietà di scelta greedy e sottostruttura ottima)
- L'analisi della complessità

Dato l'algoritmo in avanti:

Versione iterativa:

```
GREEDY-SEL(S, f)
1  n = S.length
2  A = {a_1}
3  last = 1 // indice dell'ultima attività selezionata
4  for m = 2 to n
5       if s_m ≥ f_last
6            A = A ∪ {a_m}
7            last = m
8  return A
```



Greedy-Sel-Reverse(S, f)

1 n = S.length

2 A = {a[n]}

3 first = n // indice della prima attività selezionata (partendo dalla fine)

4 for m = n-1 down to 1

5 if f[m] <= s[first]

6 A = A ∪ {a[m]}

7 first = m

8 return A