Informatica Generale - **II esonero**

docente: Ivano Salvo, 10/11/2023

Problema 1: Dati due vettori u e v ciascuno contenente elementi distinti, rispettivamente di lunghezza m ed n, si devono caricare in un vettore z (già allocato di lunghezza $p = m + n$) tutti gli elementi di u che *non* compaiono in v e tutti gli elementi di v che *non* compaiono in u. La funzione deve tornare come risultato il numero di elementi effettivamente ricopiati in z.

Esempio: Dato $u = \{5, 2, 3\}$ e $v = \{3, 7, 1, 2\}$, z può essere caricato con {1, 5, 7, ♯, ♯, ♯, ♯} (non è rilevante l'ordine con cui gli elementi appaiono in z) e viene ritornato il valore 3 (♯ indica un valore qualsiasi).

Scrivere lo pseudo-codice di *diffSimm*(u, v, z) in ciascuno dei seguenti casi:

1. u e v qualsiasi, senza modificarli e senza allocare memoria $\Theta(m + n)$;

2. per due vettori ascendenti. Discutere la complessità e dire se, potendo modificare i vettori, al punto **1** sarebbe conveniente prima ordinarli e poi applicare questo algoritmo invece che usare quello che non li modifica;

3. per due vettori contenenti valori in un intervallo noto $[min, max]$ di interi sufficientemente "piccolo" $(max - min \in \mathcal{O}(m + n))$.

Problema 2: Per vettori "sufficientemente piccoli", gli ordinamenti naïf risultano più efficienti di quelli avanzati. Nella seguente versione di *quickSort* si sospendono le chiamate ricorsive quando l'intervallo da ordinare è più piccolo di un dato k e si chiama *insertSort* per ordinare gli elementi rimasti fuori posto:

```
def quickSort(v, k):
   qckSrtAux(v, 0, len(v), k)
   insertSort(v)

def qckSrtAux(v, inf, sup, k):
   if sup - inf > k:
      m = partiziona(v, inf, sup)
      qkcSrtAux(v, inf, m, k)
      qkcSrtAux(v, m, sup, k)
```

1. La funzione ordina correttamente il vettore v? In caso affermativo, argomentare brevemente la correttezza, oppure spiegate cosa non funziona.

2. qual è la complessità asintotica *attesa* di questa versione di *quickSort*? Sarebbe uguale se l'ultima chiamata fosse *selectSort*(v)?

3. Discutere cosa accade se applichiamo la stessa idea a *mergeSort*. Quale potrebbe essere un'idea migliore per sfruttare l'efficienza degli ordinamenti naïf su vettori "piccoli" dentro *mergeSort*?