PRIMA PARTE

1. **[3 punti]** Si consideri l'algoritmo `computeFailKMP` per il calcolo della failure function f di un pattern P. In una generica iterazione del while, si confronta $P[j]$ e $P[i]$, con $0 \leq j < i < |P|$, e i valori $f(0), f(1), \ldots, f(i-1)$ sono stati già calcolati. Se $P[j] = P[i]$ cosa fa l'algoritmo in quella iterazione e perché è corretto fare così?

2. **[3 punti]** Sia T un albero binario con $n \geq 3$ nodi e radice r. Sia u il figlio sinistro di r e v il figlio destro di r.

   (a) Qual è il nodo successore di u nella visita in postorder di T?

   (b) Qual è il nodo successore di v nella visita in postorder di T?

3. **[3 punti]** Si consideri una coda con priorità `P`, implementata tramite *heap*, inizialmente vuota. Si supponga di eseguire le seguenti operazioni nell'ordine dato: `P.insert(5,·)`, `P.insert(3,·)`, `P.insert(6,·)`, `P.insert(4,·)`, `P.insert(7,·)`, `P.insert(2,·)`, `P.removeMin()`. Disegnare lo heap risultante (come albero, riportando solo le chiavi) dopo tutte le operazioni

4. **[4 punti]** Dimostrare che un Multi-Way Search Tree contenente n entry ha $n+1$ foglie.

5. **[3 punti]** Il seguente algoritmo ordina una sequenza `S` di n interi.

   ```
   S1 ← interi pari di S; S2 ← interi dispari di S;
   Ordina S1 con MergeSort;
   Ordina S2 con QuickSort deterministico;
   Merge(S1,S2;S);
   ```

   Osservando che la separazione di pari e dispari può essere fatta in tempo $\Theta(n)$, determinare la complessità al caso pessimo dell'algoritmo esprimendola con $\Theta(\cdot)$.

SECONDA PARTE

1. **[5 punti]** Sia $S = S[0] \ldots S[n-1]$ una sequenza di $n \geq 1$ interi distinti. Il seguente algoritmo determina la lunghezza della più lunga sottosequenza *crescente* di S.

```
curr ← 1; length ← 1;
for i ← 1 to n-1 do {
  if (S[i-1] < S[i]) then {
    curr ← curr+1;
    if (curr > length) then length ← curr;
  }
  else curr ← 1;
}
return length;
```

   Trovare un opportuno invariante per il ciclo **for**.

2. **[6 punti]** Sia T un albero binario di ricerca le cui entry rappresentano studenti di un'università. Ogni studente è associato a una entry (k, x), dove k è la matricola, e x indica se lo studente è straniero ($x = 1$) o italiano ($x = 0$). Per ogni nodo $v \in T$ esiste un intero v.`numStr` che riporta il numero di studenti stranieri in T_v (sottoalbero con radice v). Progettare un algoritmo *ricorsivo* `MinMatStraniero`(T, v) che dato un nodo $v \in T$ restituisce *la più piccola matricola di uno studente straniero in* T_v, e analizzarne la complessità. Se non ci sono studenti stranieri in T_v l'algoritmo restituisce `null`.

3. **[5 punti]** Sia $G = (V, E)$ un grafo *non connesso* con n vertici ed m archi. Progettare un algoritmo che conti le coppie di vertici $u, v \in V$ tali che u e v sono raggiungibili uno dall'altro tramite cammini, analizzandone la complessità. Per avere punteggio pieno la complessità deve essere $O(n + m)$. (Si ricordi che da un insieme di K oggetti si possono formare $K(K-1)/2$ coppie distinte.)

TEMPO COMPLESSIVO A DISPOSIZIONE: 2.5 ore