PRIMA PARTE

1. **[3 punti]** Definire brevemente il concetto di *Abstract Data Type* (ADT) caratteristico dell'approccio object-oriented, e spiegare in che modo Java permette di specificare un ADT.

2. **[3 punti]** Si consideri l'algoritmo di pattern matching di Knuth Morris e Pratt (KMPMatch) e si supponga che in un'iterazione del ciclo **while** si trovi $T[i] \neq P[j]$ con $j > 0$. Dire quale confronto farà l'algoritmo nella successiva iterazione, spiegando perché è corretto fare così.

3. **[3 punti]** Una funzione hash è caratterizzata da due componenti: un hash code e una compression function. Descrivere il ruolo di ciascun componente e indicare un buon hash code per stringhe di caratteri.

4. **[4 punti]** Sia S una sequenza di n chiavi intere da ordinare. Dire quale tra gli algoritmi di ordinamento che conoscete usereste nei seguenti casi, specificando la complessità risultante: (i) le chiavi da ordinare appartengono all'intervallo $[1, n^2]$; (ii) esistono 3 chiavi, tolte le quali S risulta già ordinata. Motivare le risposte.

5. **[3 punti]** Sia $G = (V, E)$ un grafo non diretto e connesso. Spiegare brevemente come si può determinare il cammino più breve tra due vertici $s, t \in V$.

SECONDA PARTE

1. **[4 punti]** Sia $G(n)$ una funzione definita come segue:
   - $G(0) = 1$ e $G(1) = 2$;
   - $G(n-1) + 2G(n-2)$ per ogn $n > 1$.

   (a) Trovare i valori di $G(n)$, per $n = 2, 3, 4, 5$.

   (b) Dal punto precedente, intuire una formula chiusa per $G(n)$, per $n \geq 0$, e provarla per induzione.

2. **[6 punti]** Sia T un albero binario di ricerca contenente n entry con chiavi reali distinte. Descrivere un algoritmo *non ricorsivo* che determina la più piccola chiave positiva presente in T, e analizzarne la complessità. Se in T non esistono chiavi positive, l'algoritmo deve restituire `'no positive key'`.

3. **[6 punti]** Sia S una sequenza di n chiavi intere. Non si fanno ipotesi sul range delle chiavi, che può essere arbitrariamente grande, ma si sa che in S ci sono solo $k = O(\log n)$ chiavi distinte. Una chiave si dice *frequente* se compare almeno 10 volte in S. Progettare un algoritmo di complessità $o(n \log n)$ che determini quante sono le chiavi frequenti in S. Far vedere che la complessità è $o(n \log n)$.

TEMPO COMPLESSIVO A DISPOSIZIONE: 2.5 ore