PRIMA PARTE

1. **[3 punti]** Si definisca come problema computazionale il problema di trovare la chiave massima e il suo indice in una sequenza S di chiavi intere.

2. **[3 punti]** Si consideri il seguente frammento di pseudocodice estratto dall'algoritmo di pattern matching `findKMP(`T`,`P`)`.

```
i ← 0; j ← 0;
while (i < n) do
    if (P[j] = T[i]) then
        i ← i + 1; j ← j + 1;
        if (j = m) then return (i − m);
    else
        if (j > 0) then j ← f(j − 1);    // f ≡ failure function
        else i ← i + 1;
```

   Quale invariante vale alla fine di una generica iterazione del while?

3. **[3 punti]** L'array seguente rappresenta uno *heap* tramite level numbering:

(2,A)	(9,B)	(6,C)	(10,D)	(15,E)	(13,F)	(8,G)	(28,H)	(17,I)	(19,J)

   Si mostri l'array che rappresenta lo heap risultante dopo l'esecuzione dell'operazione `insert(4,K)`.

4. **[4 punti]** Con riferimento alla Mappa:

   (a) Dare la specifica precisa del metodo `put`.

   (b) Supponendo di implementare la Mappa tramite lista position-based, indicare la complessità di `put` giustificando la risposta.

5. **[3 punti]** Sia $G = (V, E)$ un grafo con n vertici ed m archi. Descrivere brevemente come determinare se G è connesso in tempo $O(n + m)$.

SECONDA PARTE

1. **[5 punti]** Sia $T(n)$ la complessità di un algoritmo e si supponga di sapere che:

   - $T(1) = 2$;
   - $T(n) = T(\lceil n/4 \rceil) + T(\lceil n/3 \rceil) + n$, per ogni $n > 1$.

   Dimostrare che $T(n) \geq 2n$ per ogni $n \geq 1$.

2. **[6 punti]** Sia T un albero binario proprio dove ogni nodo v memorizza un flag v`.active` che vale 1 (v *attivo*) oppure 0 (v *non attivo*). Per ogni sottoalbero T_v, radicato in v, sia a_v il numero di nodi attivi in T_v e na_v il numero di nodi non attivi in T_v (tali valori non sono memorizzati nell'albero). L'albero T si dice *valido* se *per ogni sottoalbero* T_v si ha che $a_v - na_v \geq -1$. Si vuole progettare un algoritmo *ricorsivo* `Check` per determinare se T è valido. Sia `Check`(T, v) la generica invocazione su un nodo $v \in T$.

   (a) Descrivere tramite pseudocodice `Check`(T, v), specificando con attenzione cosa deve restituire.

   (b) Analizzare la complessità di `Check`$(T, T.\text{root}())$ in funzione del numero n di nodi in T.

3. **[5 punti]** Siano $S_1 = S_1[0]S_1[1]\ldots S_1[n_1 - 1]$ e $S_2 = S_2[0]S_2[1]\ldots S_2[n_2 - 1]$ due sequenze ordinate di chiavi intere. In ogni sequenza le chiavi sono distinte, ma una stessa chiave può comparire sia in S_1 che in S_2. Progettare una modifica dell'algoritmo `Merge` che restituisca il numero di chiavi che compaiono sia in S_1 che in S_2.

   N.B. Per avere punteggio pieno l'algoritmo deve usare spazio aggiuntivo costante oltre alle due sequenze di input.

TEMPO COMPLESSIVO A DISPOSIZIONE: 2.5 ore