Algoritmi e Strutture Dati

Esercizio 1.[11 punti]
Risolvere in ordine di grandezza la seguente equazione ricorsiva

$$T(n) = \begin{cases} 1 & \text{if } n \leq 1, \\ T(n/2) + n + 2^n & \text{if } n > 1. \end{cases}$$

Traccia della soluzione dell'esercizio 1.
$T(n)$ è banalmente $\Omega(2^n)$.
Per induzione è possibile anche dimostrare che $T(n)$ è $O(2^n)$
$T(n) = T(n/2) + n + 2^n \leq k2^{n/2} + n + 2^n$
Esiste k tale per cui $k2^{n/2} + n + 2^n \leq k2^n$?
Risolvendo la disequazione otteniamo che ciò è vero per $k \geq \frac{n+2^n}{2^n - 2^{n/2}}$
Visto che

$$\lim_{n->\to\infty} \frac{n + 2^n}{2^n - 2^{n/2}} = 1$$

ciò significa che per tutti gli n maggiori di un opportuno n_0 la quantità $\frac{n+2^n}{2^n-2^{n/2}}$ sarà minore di 2. A questo punto basta prendere $k = 2$ e il gioco è fatto !

Esercizio 2.[11 punti]
Dato un albero binario, trovare un algoritmo in pseudocodice per calcolare, la sua larghezza. La larghezza di un albero è il numero massimo di nodi allo stesso livello o, equivalentemente, alla stessa distanza dalla radice. Calcolare il costo dell'algoritmo proposto e argomentare la sua correttezza.

Traccia della soluzione dell'esercizio 2.
Applichiamo una BFS sull'albero. La BFS memorizza su ogni nodo $v_i \in N$ la sua distanza $d(v_i)$ dalla radice e costa $\Theta(N + A)$ che su un albero è $\Theta(N)$ visto che $A = \Theta(N)$. Sia V un vettore che memorizza in posizione i il numero di nodi distanti i dalla radice dell'albero. Per costruire V basta scorrere tutti i nodi v_i e eseguire l'istruzione $V(d(v_i)) = V(d(v_i)) + 1$ (costo $\Theta(N)$). A questo punto basta calcolare il massimo di V (costo $\Theta(N)$). Costo totale $\Theta(N)$.

Esercizio 3.[11 punti]
Fornire un algoritmo greedy polinomiale (descrivendo prima l'idea a parole e poi fornendo lo pseudocodice) per risolvere "Coloring" (dato un grafo non pesato e non orientato

colorare i suoi nodi con il numero minimo di colori, evitando di assegnare lo stesso colore a coppie di nodi collegate da un arco). Dimostrare che l'algoritmo fornito non è ottimo. Determinare il suo costo computazionale.

Traccia della soluzione dell'esercizio 3.
L'esercizio in questione può essere risolto in molti modi diversi. Qui di seguito riportiamo una possibile soluzione.

$Coloring(G)$

```
while esistono nodi non ancora colorati
        scegli un nodo u non ancora colorato
        se possibile, assegna a u un colore già assegnato
        altrimenti assegna a u un colore nuovo
```

Il costo computazionele di $Coloring$ è evidentemente polinomiale...
Controesempio: sia $G = (\{1,2,3,4,5\}, \{(1,2),(1,3),(2,3),(2,4),(2,5),(3,5),(4,5)\})$

colorazione prodotta dall'algoritmo non ottima:
nodo 1 $\rightarrow$ *colore A*
nodo 2 $\rightarrow$ *colore B*
nodo 3 $\rightarrow$ *colore C* fino a qui la scelta dei colori è obbligata
nodo 4 $\rightarrow$ *colore A* scelgo un colore ammissibile già assegnato (scelta greedy sbagliata)
nodo 5 $\rightarrow$ *colore D* serve un quarto colore !

colorazione ottima:
nodo 1 $\rightarrow$ *colore A*
nodo 2 $\rightarrow$ *colore B*
nodo 3 $\rightarrow$ *colore C* fino a qui la scelta dei colori è obbligata
nodo 4 $\rightarrow$ *colore C* scelgo un colore ammissibile già assegnato
nodo 5 $\rightarrow$ *colore A* 3 colori invece di 4 !