ESERCIZIO 1

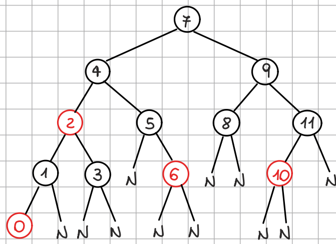
i) Un albero Red-Black è un albero binario che presenta le seguenti caratteristiche:
- Tutti i nodi sono colorati di rosso o nero;
- Un nodo rosso non può avere figlio rosso, quindi non ci possono essere due rossi consecutivi.
- Tutte le foglie sono nere e NILL (non contengono dati)
- A partire da un qualsiasi nodo interno, ogni percorso che raggiunge le foglie deve avere lo stesso numero di nodi neri.
- I dati sono solo sui nodi interni.

Un albero è perfettamente bilanciato $\iff \forall x \in T \left(\big| |x.sx| - |x.dx| \big| \leq 1 \right)$.
Un albero è completo sui nodi non foglia è pieno e per ogni nodo il sottoalbero sinistro non è meno pesante del destro.
Un albero è AVL $\iff \forall x \in T \left(|h(x.sx) - h(x.dx)| \leq 1 \right)$

Fatte queste specifiche possiamo dire che:
T_1: non è completo, non è AVL, non è PB, non può essere RB in quanto non esiste una colorazione per cui valga la proprietà sulla lunghezza nera.

T_2: non è PB, non è completo, non è AVL e non può essere neanche RB

T_3: non è PB, è AVL, non è completo, può essere RB con questa colorazione:



T_4: non è PB, non è completo, non è AVL, non può essere AVL.

ii) a) $\log(n^3) = \Theta(\log(n \cdot \log(n^n)))$
$3\log(n) = \Theta(\log(n) + \log(n \log(n)))$
$3\log(n) = \Theta(2\log(n) + \log(\log(n)))$

$$\lim_{n\to\infty} \frac{3\log(n)}{2\log(n)+\log(\log(n))} = \lim_{n\to\infty} \frac{\frac{d}{dn}\, 3\log(n)}{\frac{d}{dn}\, 2\log(n)+\log(\log(n))} = \lim_{n\to\infty} \frac{\frac{3}{n\ln(2)}}{\frac{2\ln(n)+1}{n\ln(2)\ln(n)}} =$$

$$\lim_{n\to\infty} \frac{3}{\cancel{n\ln(2)}} \cdot \frac{\cancel{n\ln(2)}\ln(n)}{2\ln(n)+1} = \lim_{n\to\infty} \frac{3}{2} \frac{\ln(n)}{\ln(n)+1} = \lim_{n\to\infty} \frac{3}{2} \cdot \frac{\cancel{\ln(n)}}{\cancel{\ln(n)}\left(1+\underbrace{\frac{1}{\ln(n)}}_{=0}\right)} = \frac{3}{2}$$ ✓

b) $n^{2n} + n^3 + \Theta(n^n)$

$$\lim_{n\to\infty} \frac{n^{2n}+n^3}{n^n} = \lim_{n\to\infty} \underbrace{\frac{n^{2n}}{n^n}}_{=\infty} + \underbrace{\frac{n^3}{n^n}}_{=0} = \infty \text{ (per la gerarchia degli infiniti)}$$ ✗

c) $\log(n^{3n}) + \sqrt{n} = \Theta\left(\log \sqrt{(n^n)}\right)$

$3n \log(n) + \sqrt{n} = \Theta\left(\frac{n}{2} \log(n)\right)$

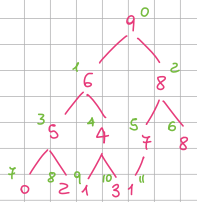
$$\lim_{n\to\infty} \frac{3n\log(n)+\sqrt{n}}{\frac{n}{2}\log(n)} = \lim_{n\to\infty} \frac{\cancel{\log(n)}\left(3n + \overbrace{\frac{\sqrt{n}}{\log(n)}}^{=0}\right)}{\frac{n}{2}\cancel{\log(n)}} = \lim_{n\to\infty} \frac{3n}{\frac{n}{2}} = 6$$ ✓

iii)

```
Function Heapify (A,n,i)
  (m,l,r) ← (i, 2i+1, 2i+2)
  if (l<n) ∧ (A[m]<A[l]) then
    m ← l
  if (r<n) ∧ (A[m] < A[r]) then
    m ← r
  if m ≠ i then
    Swap (A,i,m)
    Heapify (A,n,m)
```

```
Function Build Heap (A,n)
  for i from ⌊n/2⌋-1 down to 0 do
    Heapify (A,n,i)
```

a)



=> risultato

a) Sbagliata perché nell'array risultante, 4 è figlio destro e non sinistro di 6.

b) Corretta

c) Sbagliata perché significa è stata applicata erroneamente Heapify quando nella cella 4 c'era 1 e nella cella 9 c'era 4. In particolare, non è stato effettuato lo swap tra i due.

ESERCIZIO 2

$$T(n) = \begin{cases} 1, & \text{se } n \leq 2 \\ 4 \cdot T(\sqrt{n}) + (\log(n))^2, & \text{altrimenti} \end{cases}$$

Livello	#Nodi livello	Dimensione input	Contributo nodo	Contributo totale livello
0	1	n	$\log(n)^2$	$\log(n)^2$
1	4	$n^{\frac{1}{2}}$	$\frac{1}{4}\log(n)^2$	$\log(n)^2$
2	16	$n^{\frac{1}{4}}$	$\frac{1}{16}\log(n)^2$	$\log(n)^2$
3	64	$n^{\frac{1}{8}}$	$\frac{1}{64}\log(n)^2$	$\log(n)^2$
i	4^i	$n^{\frac{1}{2^i}}$	$\frac{1}{4^i}\log(n)^2$	$\log(n)^2$

Calcolo l'altezza :

$$\frac{1}{n^{2^h}} \leq 2 \iff \frac{1}{2^h} \log_2(n) \leq 1 \iff 2^h \geq \log_2(n) \iff h \geq \log_2(\log_2(n))$$

Dunque otteniamo : $\Theta(\log(n)^2 \cdot h) = \Theta(\log(n)^2 \cdot \log_2(\log_2(n)))$

ESERCIZIO 3

```
Function DeleteAVL(x,d)
  if x ≠ ⊥ then
    if x.dat > d then
      x.sx ← DeleteAVL(x.sx, d)
      x ← R.Balance(x)
    else if x.dat < d then
      x.dx ← DeleteAVL(x.dx, d)
      x ← LBalance(x)
    else
      x ← DeleteDataAVL(x)
  return x
```

```
Function DeleteDataAVL(x)
  if x.sx = ⊥ then
    x ← SkipRight(x)
  else if x.dx = ⊥ then
    x ← SkipLeft(x)
  else
    x ← Get&DeleteMinAVL(x.dx, x)
    x ← LBalance(x)
  return x
```

```
Function Get&DeleteMinAVL(x,p)
  if x.sx = ⊥ then
    d ← x.dat
    y ← SkipRight(x)
  else
    d ← Get&DeleteMinAVL(x.sx, x)
    y ← RBalance(x)
  SwapChild(p, x, y)
  return d
```

```
Function LBalance(x)
  if Height(x.sx) - Height(x.dx) > 1 then
    if Height(x.sx.sx) > Height(x.sx.dx) then
      x ← L2RRotAVL(x)
    else
      x ← LDRotAVL(x)
  else
    UpdateHeight(x)
  return x
```

```
Function L2RRotAVL(x)
  y ← L2RRot(x)
  UpdateHeight(x)
  UpdateHeight(y)
  return y
```

```
Function LDRotAVL(x)
  x.sx ← R2LRotAVL(x.sx)
  return L2RRotAVL(x)
```

```
Function L2RRotAVL(x)
  y ← x.sx
  x.sx ← y.dx
  y.dx ← x
  return y
```

```
Function SkipLeft(x)
  y ← x.sx
  Deallocate(x)
  return y
```

```
Function SkipRight(x)
  y ← x.dx
  Deallocate(x)
  return y
```

L'albero subito dopo l'eliminazione appare così :