Université Claude Bernard – Lyon1 | Licence Sciences et Technologies – L3

Algorithmique, Programmation et Complexité – LIFAP6 | Printemps 2017

Interrogation 3 : structures arborescentes et Master Theorem

Durée : 30 minutes.

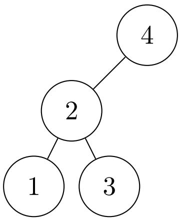
Nom :

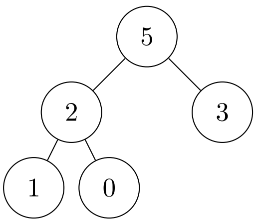
Prénom :

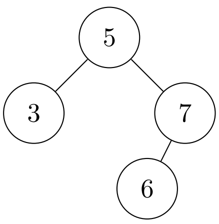
Attention : le sujet est recto-verso, n'oubliez pas de tourner la page.

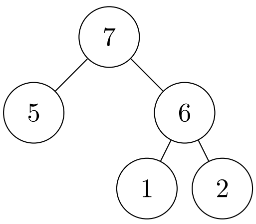
1 Arbres

a) Pour chaque arbre ci-dessous, indiquez s'il correspond à un tas binaire valide organisé pour fournir le maximum, un AVL valide, les deux, ou aucun des deux, et justifiez vos réponses. Dans le cas d'un tas binaire, indiquez le tableau correspondant.

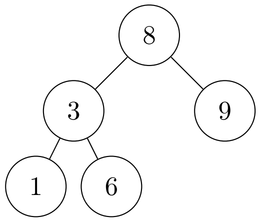








b) Soit l'arbre binaire de recherche suivant, indiquez sur chaque nœud son déséquilibre au sens des AVL, puis dessinez à côté l'arbre après insertion de la valeur 5 et rééquilibrage. Vous indiquerez également les opérations réalisées pour rééquilibrer l'arbre.



2 Master Theorem

c) Calculez les valeurs de $\log_3(9)$, $\log_2(8)$ et $\log_5(1)$.

Soit T une fonction de la forme

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

le master theorem affirme que :

1. si $f(n) = \mathrm{O}(n^c)$ avec $c < \log_b a$, alors la complexité de f est trop faible devant la complexité liée à la récursion. Dans ce cas
$$T(n) = \Theta(n^{\log_b a}).$$
2. si $f(n) = \Omega(n^c)$ avec $c > \log_b a$, alors la complexité de f est prépondérante devant la complexité liée à la récursion. Dans ce cas il n'est pas toujours possible de déterminer la complexité finale à cause des appels récursifs à f. Si f respecte le critère de régularité : $\exists k < 1, \exists n_0 \geq 0, \forall n \geq n_0, af(n/b) \leq kf(n)$, alors
$$T(n) = \Theta(f(n)).$$
3. si $f(n) = \Theta(n^{\log_b a})$, alors les deux éléments interagissent et la complexité finale est donnée par
$$T(n) = \Theta(n^{log_b a} \log n).$$

d) Quelle est la complexité de l'algorithme suivant en fonction de n ? Que calcule-t-il ?

```
Fonction calcul(réel x, entier n) → réel
    si n = 0 alors
        retourner 1
    sinon
        si n est impair alors
            rec ← calcul(x, (n − 1)/2)
            retourner rec × rec × x
        sinon
            rec ← calcul(x, n/2)
            retourner rec × rec
```