Analyse d'Algorithmes et Programmation

Contrôle Continu 1

Christina Boura et Yann Rotella
{christina.boura, yann.rotella}@uvsq.fr

18 mars 2022

Durée : 2h

1 Complexité asymptotique

Exercice 1

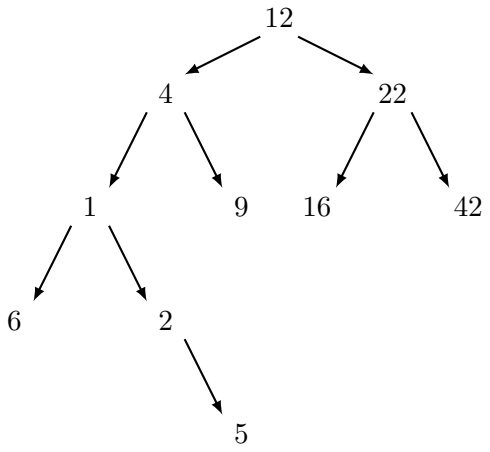
(a) Démontrer que $n! \in \mathcal{O}(n^n)$ et que $\log(n!) \in \mathcal{O}(n \log(n))$

(1 point)

(b) Démontrer que $1 + 2 + \cdots + n \in \Theta(n^2)$ ou donner un contre-exemple.

(1 point)

Exercice 2 On considère l'arbre binaire suivant.



(a) Cet arbre est-il un arbre binaire de recherche ? L'arbre binaire est-il H-équilibré ? Justifier. Quelle est la hauteur de l'arbre ? Quelle est sa taille ?

(1 point)

(b) Transformer cet arbre, en gardant les mêmes étiquettes des noeuds, afin d'avoir un arbre binaire de recherche H-équilibré.

(1 point)

(c) Insérer dans ce nouvel arbre un noeud d'étiquette 10 avec la procédure d'insertion vue en cours. Supprimer ensuite le noeud d'étiquette 9 avec la procédure de suppression vue en cours.

(1 point)

(d) Proposer une procédure d'insertion qui permet, si l'arbre est H-équilibré, préserver cette propriété.

(2 points)

Exercice 3

(a) Proposer un algorithme qui prend en entrée un entier n de t bits et renvoie sa racine carrée tronquée, c'est à dire $\lfloor \sqrt{n} \rfloor$ (sans utiliser évidemment le module `math`).

(b) Donner la classe de complexité de votre algorithme en fonction de la taille de l'entrée t.

(c) Si ce n'est pas déjà le cas, proposer un nouvel algorithme (qui calcule aussi la racine carrée) en $\mathcal{O}(t)$.

(4 points)

2 Programmation

La classe `Etudiant` ci-dessous contient deux champs : un champ `nom`, correspondant au nom de l'étudiant et un champ `note`, correspondant à sa note au cours d algorithmique.

```
class Etudiant:
        def __init__(self,nom,note):
                self.nom = nom
                self.note = note
```

Recopiez cette classe dans votre fichier.

Tri par insertion sur un tableau

(a) Écrire une fonction `insertionSort` qui effectue le tri par insertion sur une liste donnée en entrée et dont chaque case est un objet de type `Etudiant`.

(b) Vous pouvez tester votre implémentation en créant cinq objets `Etudiant` à partir des données suivantes : `Jade : 11.8`, `Anne : 11.2`, `Henri : 12`, `Fatima : 16.3`, `Quentin : 13`, `Kylian : 13.9`

(4 points)

Table de hachage On utilise maintenant une table de hachage de taille 21 pour sauvegarder les données sur les étudiants. La fonction de hachage utilisée est la fonction $h(n) = \lfloor n \rfloor$, qui prendra en entrée une note n de 0 à 20 et retournera la partie entière de n.

(a) Créer une table de hachage de taille 21 et écrire une fonction qui prend en entrée un objet de type `Etudiant` et une table de hachage et qui insère l'objet `Etudiant` dans la table à l'aide de la fonction de hachage h ci-dessus. Vous pouvez utiliser la méthode `math.floor` du module `math`. Les collisions devront être résolues par chaînage.

(b) Écrire une fonction qui prend en entrée une table de hachage construite comme ci-dessus et qui recherche puis affiche le nom de l'étudiant(e) ayant la meilleure note.

(6 points)