# Analyse d'algorithmes et programmation - $1^{er}$ contôle continu

Enseignants: Christina Boura, Gaëtan Leurent

2 mars 2017

*Durée du contrôle : 1h30.*

**Les documents ainsi que l'accès aux programmes élaborés pendant les séances de TP sont autorisés.**

**Bonne chance !**

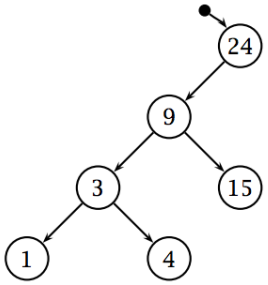
## 1 Questions

**Exercice 1**

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

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

**Exercice 2** On considère l'arbre binaire de recherche ci-dessous :




**(a)** Donner 5 ordres d'insertion possibles (permutations) des clés qui auraient pu produire cet arbre.

**(b)** Dessiner ce même arbre après l'insertion des clés 6, 45, 32, 98, 55 et 69 en respectant cet ordre.

**(c)** Dessiner l'arbre résultant de la question (b) après la suppression des clés 6 et 45. Quelle est sa hauteur ?

**(d)** Dessiner un arbre binaire de recherche contenant les mêmes clés que l'arbre de la question (c) et ayant une hauteur $h = 3$.

**Exercice 3** Le temps d'execution $T(n)$ d'un algorithme de tri est donné en fonction du nombre $n$ d'éléments à trier par la formule récursive suivante :

$$T(n) = \begin{cases} 0 & \text{si } n = 1, \\ 3T(n/3) + 2cn & \text{sinon,} \end{cases}$$

avec $c$ une constante positive. Résoudre cette équation afin de dériver une formule explicite pour $T(n)$ si on suppose que $n = 3^m$. Donner la complexité de cet algorithme en utilisant la notation $\mathcal{O}$.

**Exercice 4** On s'intéresse au nombre minimal de produits scalaires nécessaires afin de calculer le produit matriciel $M_1 M_2 M_3 M_4 M_5$ où la dimension de la matrice $M_i$ est notée $p_{i-1} \times p_i$ ($p_{i-1}$ lignes et $p_i$ colonnes) pour $1 \leq i \leq 5$. Si on note $c_{i,j}$, $j \geq i$, le nombre minimal de produits scalaires nécessaires pour calculer le produit $M_i M_{i+1} \cdots M_j$, donner l'expression de $c_{2,5}$ en fonction des quantités $c_{i,j}$ avec $j - i < 3$ et des variables $p_i$.

## 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
```

Recopier cette classe dans votre fichier.

**Tri par insertion sur un tableau** 1. 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`.

2. Vous pouvez tester votre implémentation en créant cinq objets `Etudiant` à partir des données suivantes :

   `Louis 11.8`, `Anne 11.2`, `Henri 8.5`, `Fatima 17.3`, `Quentin 13`.

   Votre programme doit afficher après le tri le nom de l'étudiant(e) ayant la note la plus élevée.

**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 $n$.

1. 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 ci-dessus. Vous pouvez utiliser la méthode `math.floor` du module `math`. Les collisions devront être résolues par chaînage.
2. 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.