TD4 : complexité des algorithmes récursifs

Exercice 1 : Savoir établir une équation de récurrence

On propose d'analyser la fonction `f` suivante :

```
def f (n):
  if n < 0:
    return 1
  else:
    r = f(n-2)
    if n % 2 != 0:
      r = r + f(n-1)
    return r
```

Q 1.1 Etablir l'équation de récurrence permettant de compter le nombre total d'appels à la fonction `f`.

Exercice 2 : Savoir établir et résoudre « à la main » une équation de récurrence simple

Soit la fonction `g` suivante :

```
def g(t):
  n = len(t)
  if n <= 1:
    return 0
  else:
    return t[n-1] - t[n-2] + g(t[0:n-1])
```

Q 2.1 Etablir une équation de récurrence permettant de compter le nombre d'appels à `g`.

Q 2.2 Donner la complexité asymptotique en nombre d'appels récursifs.

Exercice 3 : Savoir établir et résoudre « à la main » une équation de partition

On souhaite procéder à la recherche du maximum dans un tableau quelconque. On propose la stratégie récursive suivante : on divise le tableau en deux ; on lance la recherche récursivement dans les deux sous-tableaux ; on conserve le maximum des valeurs retournées par les deux appels récursifs.

On supposera traiter des tableaux d'entiers positifs ou nuls et que le tableau contient au moins un entier.

On utilisera la fonction `max` qui permet de calculer le maximum entre deux valeurs.

Q 3.1 Ecrire en Python cet algorithme.

Q 3.2 Donner sous forme d'une équation de récurrence la complexité en nombre d'appels à `maxtab`.

Q 3.3 En utilisant la méthode par arbre, déterminer le comportement asymptotique en temps.

Q 3.4 Donner également sous forme d'une équation de récurrence la complexité en espace.

Q 3.5 De même déterminer le comportement asymptotique en espace.

Exercice 4 : Complexité de l'algorithme de résolution des tours de Hanoï

On rappelle l'algorithme qui s'énonce en français ainsi :

```
hanoi (n, depart, arrivee) =
   si n = 0
      fin
   sinon
      hanoi (n-1,depart,intermediaire)
      deplacer(depart,arrivee)
      hanoi (n-1,intermediaire,arrivee)
```

Q 4.1 Exprimez sous la forme d'une fonction récursive le nombre de déplacements en fonction de n. N'oubliez pas les conditions initiales.

Q 4.2 *[A faire en dehors du TD]* En programmant l'algorithme et en utilisant Gnuplot, déterminer expérimentalement une approximation asymptotique de la complexité si les déplacements sont comptés.

Q 4.3 Mêmes questions mais en comptant les appels à la fonction.

Exercice 5 : Mise en application des théorèmes

Q 5.1 Reprendre les questions des exercices 2 et 3 en appliquant le théorème général et la résolution des équations linéaires du 1er ordre

Exercice 6 : Mieux que le tri fusion ?

Le professeur Batavia s'est levé avec une idée de génie. Puisque le tri fusion est très efficace, il se dit qu'il est peut-être plus intelligent de couper le tableau en 4 parties plutôt qu'en deux parties à chaque fois.

Il écrit donc une fonction qui fusionne quatre tableaux triés et une fonction de tri qu'il exprime ainsi :

```python
def mon_super_tri (t):
  n = len(t)
  # les longueurs des quatre tranches du tableau
  l1 = ...
  l2 = ...
  l3 = ...
  l4 = ...
  if n = 1:
    return t.copy()
  else
    t1 = mon_super_tri(t[0:l1])
    t2 = mon_super_tri(t[l1:l1+l2])
    t3 = mon_super_tri(t[l1+l2:l1+l2+l3])
    t4 = mon_super_tri(t[l1+l2+l3:])
    return fusionner4(t1,t2,t3,t4)
```

Q 6.1 Décrivez le fonctionnement de `fusionner4`.

Q 6.2 Exprimez la complexité en temps de `fusionner4` dans le pire des cas.

Q 6.3 Exprimez la complexité en temps du tri proposé dans le pire des cas.

Q 6.4 Donnez une borne asymptotique dans le pire des cas. Justifiez.

Q 6.5 Que pensez-vous de ce tri ?

Exercice 7 : Paire de points les plus proches

On se donne un ensemble P de n points dans le plan. On cherche à identifier la paire de points la plus proche.

Q 7.1 Quelle est la complexité en temps d'un algorithme naïf pour résoudre ce problème ?

Q 7.2 On vous donne l'algorithme récursif suivant :

Entrée : P : les n points, dans l'ordre croissant de leurs ordonnées
Entrée : *debut* : premier indice du point à considérer dans P (par défaut 0)
Entrée : *fin* : dernier indice du point à considérer dans P (par défaut $|P| - 1$)
Fonction *ClosestPoints(P, debut, fin)*

```
si fin = debut + 1 alors
    retourner dist(P[debut], P[debut + 1]);
fin
d1 ← ClosestPoints(P, debut, fin/2);
d2 ← ClosestPoints(P, fin/2 + 1, fin);
d ← min(d1, d2);
pour i allant de debut à fin faire
    pour j allant de debut à min(debut + 14, fin) faire
        d ← min(dist(P[i], P[j]), d);
    fin
fin
retourner d;
```

fin

Quelle est sa complexité asymptotique en temps, sachant que la fonction dist est exécutée en temps constant ?

Exercice 8 : Pour s'entraîner sur le théorème général

Trouver le comportement asymptotique des fonctions suivantes :

1. $c(n) = 9c(n/3) + n$
2. $c(n) = c(2n/3) + 1$
3. $c(n) = 3c(n/4) + n \log_2 n$

Exercice 9 : Pour s'entraîner, encore

Résoudre :

1. $c(n) = 3c(n-1) + 2,\ c(0) = 0$
2. $c(n) = 4c(\frac{n}{2}) + n$
3. $c(n) = 4c(\frac{n}{2}) + n^2$
4. $c(n) = 4c(\frac{n}{2}) + n^3$

Exercice 10 : Un peu dur, pour celles et ceux qui aiment jouer avec les changements de variable

Résoudre l'équation

$$c(n) = 2c(\sqrt{n}) + \log n$$

.