Numéro d'anonymat :

Examen 2I003

Mardi 15 Janvier 2019, 2 heures
aucun document autorisé

Exercice 1 – Tas

Dans tout cet exercice, les arbres binaires sont étiquetés sur $\mathbb{N}$.

Question 1

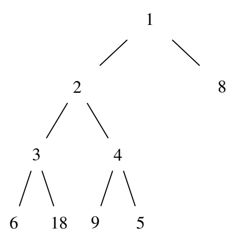
Rappeler la définition **inductive** d'un arbre binaire étiqueté sur $\mathbb{N}$. Rappeler la définition d'un arbre binaire parfait. Rappeler la définition d'un tas.

__**Solution**__:
Voir cours.

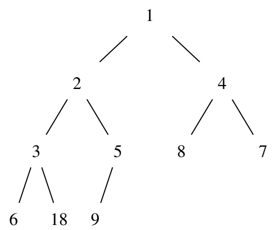
Question 2

Pour chacun des trois arbres binaires T_1, T_2, T_3 suivants, dire s'il est parfait, s'il est un tas. Justifier les réponses.

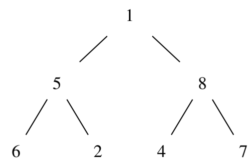
Arbre T_1 :



Arbre T_2 :



Arbre T_3 :

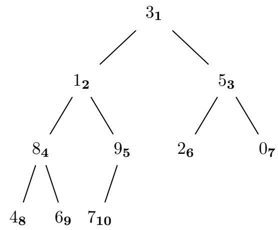


__**Solution**__:
T_1 n'est pas parfait, car l'avant-dernier niveau n'est pas entièrement rempli.
T_2 est un tas : il est parfait et les étiquettes croissent de la racine vers les feuilles.
T_3 est parfait mais n'est pas un tas : sur le chemin de la racine vers la feuille d'étiquette 2, on a $5 > 2$.

On rappelle qu'un arbre parfait de taille n peut être représenté au moyen d'un tableau $A[0..N]$, avec $N \geq n$, tel que :

— $A[0]$ contient la taille n de T
— les cases $A[1..n]$ sont remplies en parcourant T de gauche à droite, niveau par niveau.

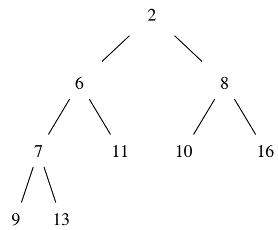
Ainsi, on peut numéroter les nœuds de l'arbre parfait T au moyen de leur position dans le tableau A comme illustré dans l'exemple suivant :



Le tableau associé à est $[10, 3, 1, 5, 8, 4, 9, 2, 0, 4, 6, 7]$

Question 3

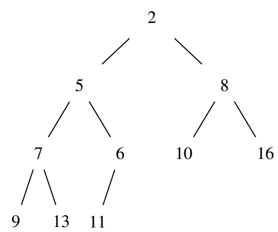
On considère le tas ExT :



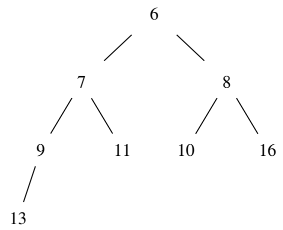
1. Donner le tableau ExA associé à ExT.
2. Réaliser l'insertion de la clé 5 dans le tas ExT, en maintenant la structure de tas. Décrire brièvement (deux phrases maximum) l'algorithme utilisé. Donner le tableau associé au résultat.
3. Réaliser la suppression de la clé minimale dans le tas ExT, en maintenant la structure de tas. Décrire brièvement (deux phrases maximum) l'algorithme utilisé. Donner le tableau associé au résultat.

Solution:

1. $ExA = [9, 2, 6, 8, 7, 11, 10, 16, 9, 13]$
2. On insère la nouvelle valeur v à la fin du tableau et on la fait remonter à sa bonne place dans le tas. Pour cela, on échange v avec la clé de son père tant que v est inférieure à la clé de son père. Tableau :



   $[10, 2, 5, 8, 7, 6, 10, 16, 9, 13, 11]$.
3. On remplace la clé de la racine par la valeur v du dernier élément du tableau et on fait redescendre v à sa bonne place. Pour cela, on échange v avec le plus petit de ses fils tant que v est supérieure à au moins l'un de ses fils. Tableau : $[8, 6, 7, 8, 9, 11, 10, 16, 13]$.



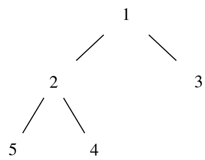
Question 4

Soit L une liste de n entiers naturels.

1. Décrire brièvement (deux phrases maximum) l'algorithme permettant de trier la liste L en utilisant un tas. Quelle en est la complexité en nombre de comparaisons dans le pire cas ?
2. Soit $L_0 = (5, 2, 3, 1, 4)$.
   (a) Construire le tas T_0 obtenu en insérant successivement les éléments de L_0 et le tableau A_0 associé à ce tas.
   (b) Détruire le tas T_0 par suppressions successives du minimum. On dessinera l'arbre obtenu à chaque suppression et on donnera le tableau associé à chaque tas.

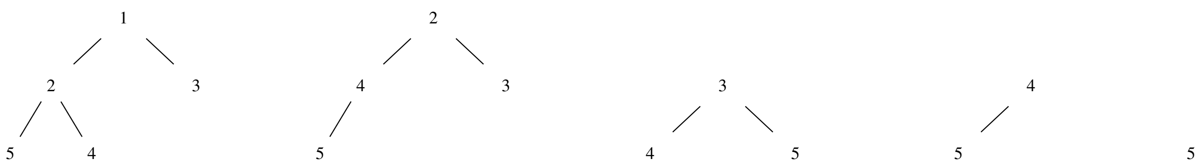
Solution:

1. On construit le tas obtenu en insérant successivement les éléments de L puis on détruit le tas par suppressions successives du minimum. La complexité pire cas est en $O(n \log n)$.
2. $L_0 = (5, 2, 3, 1, 4)$.
   (a) Tas T_0 :



   Taleau $A_0 = [5, 1, 2, 3, 5, 4]$

   (b) Valeurs successives du tas :



   Valeurs successives du tableau : $[5, 1, 2, 3, 5, 4]$, $[4, 2, 4, 3, 5, 1]$, $[3, 3, 4, 5, 2, 1]$, $[2, 4, 5, 3, 2, 1]$, $[1, 5, 4, 3, 2, 1]$ et enfin $[0, 5, 4, 3, 2, 1]$.

On dispose d'une procédure `echanger(A,i,j)` qui échange les valeurs en position i et en position j dans le tableau A.

On considère les procédures `ordonner` et `faireTas` suivantes :

```
def ordonner(A,n,i):
  if 2*i <= n:
    if 2*i + 1 <= n:
      if A[2*i] < A[2*i + 1]:
        j = 2*i
      else:
        j = 2*i + 1
    else:
      j = 2*i
    if A[i] > A[j]:
      echanger(A,i,j)
      ordonner(A,n,j)

def faireTas(A):
  n = len(A) - 1
  A[0] = n
  i = n // 2
  c = 0
  print('Pour␣c␣=',c,':␣i=', i, 'et␣A␣=', A)
  while i >= 1:
    c = c + 1
    ordonner(A,n,i)
    i = i - 1
    print('Pour␣c␣=',c,':␣i=', i, 'et␣A␣=', A)
  print('Tableau␣A␣:', A)
```

Rappel : `//` est l'opérateur de la division entière (par exemple `6//2` vaut 3, `7//2` vaut 3).
On admet que `faireTas(A)` transforme `A[1..n]` en un tas.

Question 5
Exécuter `faireTas(A)` avec `A=[0,3,1,5,8,9,2,0,4,6,7]`, en écrivant tous les affichages et en dessinant l'arbre parfait associé au tableau, à chaque étape.
<u>**Solution**</u>:

```
Pour c = 0 : i= 5 et A = [10, 3, 1, 5, 8, 9, 2, 0, 4, 6, 7]
Pour c = 1 : i= 4 et A = [10, 3, 1, 5, 8, 7, 2, 0, 4, 6, 9]
Pour c = 2 : i= 3 et A = [10, 3, 1, 5, 4, 7, 2, 0, 8, 6, 9]
Pour c = 3 : i= 2 et A = [10, 3, 1, 0, 4, 7, 2, 5, 8, 6, 9]
Pour c = 4 : i= 1 et A = [10, 3, 1, 0, 4, 7, 2, 5, 8, 6, 9]
Pour c = 5 : i= 0 et A = [10, 0, 1, 2, 4, 7, 3, 5, 8, 6, 9]
Tableau A : [10, 0, 1, 2, 4, 7, 3, 5, 8, 6, 9]
```

Un peu de dessin à faire...

Définition. La *profondeur* d'un nœud x dans un arbre binaire T est le nombre de nœuds se trouvant sur le chemin qui va de la racine à x. Par exemple, la racine est à profondeur 1, les fils de la racine sont à profondeur 2, etc.

Question 6
Notons $c(n)$ le nombre de comparaisons entre clés effectuées par `faireTas(A)` dans le pire cas.

1. Calculer $c(1)$, $c(3)$.

2. Dans cette question, on considère des arbres parfaits qui sont complets à tous les niveaux. La taille d'un tel arbre est de la forme $n = 2^h - 1$.
   (a) Montrer que, si $n = 2^h - 1$ et $n' = 2^{h+1} - 1$ alors $c(n') = c(n) + 2n$.
   Indication : si i est la position d'un nœud à profondeur k dans l'arbre parfait associé à un tableau A de taille $n = 2^h - 1$ alors `ordonner(A,n,i)` fait au plus $2(h-k)$ comparaisons entre clés.
   (b) En déduire que, si $n = 2^h - 1$, alors $c(n) = 2(n-h)$ (faire une récurrence sur $h \geq 1$).
3. Dans le cas général, la taille n d'un arbre parfait de hauteur h est telle que : $2^{h-1} < n \leq 2^h - 1$. En utilisant le résultat précédent et le fait que $c(n)$ est croissante, montrer que la complexité pire cas de `faireTas(A)` est en $O(n)$.

Solution:

1. $c(1) = 0$, $c(3) = 2$.
2. Si i est la position d'un nœud à profondeur k dans l'arbre parfait associé à un tableau A de taille $n = 2^h - 1$ alors `ordonner(A,n,i)` fait au plus $2(h-k)$ comparaisons entre clés et si i' est la position d'un nœud à profondeur k dans l'arbre parfait associé à un tableau A' de taille $n' = 2^{h+1} - 1$ alors `ordonner(A',n',i')` fait au plus $2(h+1-k)$ comparaisons entre clés. Il y a donc 2 comparaisons de plus pour chacun des n nœuds internes de T', d'où $c(n') = c(n) + 2n$.
3. Pour $h = 1$, on a $n = 1$, $c(1) = 0$ et $2(n-h) = 0$, donc la propriété est vraie.
   Soit $h \geq 1$. Supposons que $c(n) = 2(n-h)$ alors $c(n') = c(n) + 2n = 2(n-h) + 2n = 4n - 2h$. Comme $n' = 2^{h+1} - 1 = 2.2^h - 1 = 2(n+1) - 1 = 2n + 1$, on a $2n = n' - 1$, d'où $c(n') = 2(n'-1) - 2h = 2n' - 2(h+1)$.
   On peut conclure que $c(n) = 2(n-h)$ pour tout $h \geq 1$ et $n = 2^h - 1$.
4. Soit h tel que $2^{h-1} < n \leq 2^h - 1 = n'$. Puisque $c(n)$ est croissante, on a $c(n) \leq c(n') = 2(n'-h) \leq 2n' \leq 4n$, donc la complexité pire cas de `faireTas(A)` est en $O(n)$.

Exercice 2 – Somme et multiplication de polynômes

Dans cet exercice, $\mathcal{A}(x)$ et $\mathcal{B}(x)$ désignent deux polynômes de même degré $n - 1 \geq 0$. On pose $\mathcal{A}(x) = \sum_{i=0}^{n-1} a_i x^i$ et $\mathcal{B}(x) = \sum_{i=0}^{n-1} b_i x^i$. Pour alléger les notations, les polynômes pourront également être désignés par respectivement $\mathcal{A}$ et $\mathcal{B}$. Ces polynômes sont stockés sous la forme de deux listes de n éléments $A = (a_0, a_1, \cdots, a_{n-1})$ et $B = (b_0, b_1, \cdots, b_{n-1})$.

Soit $W = (w_0, w_1, \cdots, w_{m-1})$ une liste de m entiers associée au polynôme $\mathcal{W}(x) = \sum_{i=0}^{m-1} w_i x^i$. Pour tout couple $(j,k) \in \mathbb{N} \times \mathbb{N}^\star$ avec $0 \leq j < k \leq m$, on pose en utilisant la convention du langage python, $W[j:k] = (w_j \cdots, w_{k-1})$. Le polynôme correspondant à la liste $W[j:k]$ est alors $\sum_{i=j}^{k-1} w_i x^{i-j}$.

Question 1

1. Dans cette question, on pose $\mathcal{A}(x) = 2 + 3x + x^2 + 4x^3$ et $\mathcal{B}(x) = 3 + 2x + 2x^2 + 5x^3$. Donnez les listes A et B.
2. On suppose dans cette question que $W = (4, 1, 9, 3, 2, 1, 8, 0, 2)$. Que vaut $W[2:6]$? Quel est le polynôme associé à $W[2:6]$?

Solution:

1. $A = (2, 3, 1, 4)$ et $B = (3, 2, 2, 5)$.
2. $W[2:6] = (9, 3, 2, 1)$. Le polynôme associé est $\mathcal{W}^{2,6}(x) = 9 + 3x + 2x^2 + x^3$.

Question 2
Soient les fonctions `somme(A,B)`, `multUnevaleur(v,A)` et `difference(A,B)` qui construisent une nouvelle liste correspondant respectivement aux polynômes $\mathcal{A}(x) + \mathcal{B}(x)$, $v \times \mathcal{A}(x)$ et $\mathcal{A}(x) - \mathcal{B}(x)$ sans se soucier du degré du polynôme obtenu qui est toujours stocké dans une liste de n éléments.

1. Quelle est la complexité de ces 3 fonctions si les listes sont stockées dans des tableaux ? Justifiez vos réponses.
2. Même question si les listes sont stockées dans des listes simplement chaînées. Justifiez vos réponses.

Solution:

1. La fonction `somme` commence par déclarer un tableau de n éléments et va calculer toutes les valeurs $a_i + b_i$ pour $i \in \{0, \cdots, n-1\}$. C'est donc proportionnel à n dans tous les cas, donc en $\Theta(n)$. La fonction `multUnevaleur` va de la même manière calculer toutes les valeurs $v \times a_i$, elle sera donc en $\Theta(n)$. La différence va calculer `somme(A,multUnevaleur(-1,B)` et est donc en $\Theta(n)$.
2. Dans le cas de listes chaînées, la fonction `somme` contruit un à un tous les monomes du polynôme et les insère en queue. Si, après chaque insertion, on garde l'adresse du dernier élément inséré, chacune de ces insertions sera en $\Theta(1)$. L'algorithme sera donc en $\Theta(n)$. Sinon, il faut reparcourir la liste la liste à chaque itération, la complexité de l'algorithme sera alors proportionnelle à $\sum_{i=0}^{n-1} i = \frac{(n-1)(n-1)}{2}$ soit en $\Theta(n^2)$. Les autres fonctions seront selon le même principe également en $\Theta(n)$ ou en $\Theta(n^2)$ selon l'implémentation.

Question 3
Soit la fonction `sommeFacteur(W,A,B,k)` qui retourne la liste associée au polynôme $\mathcal{W}(x) + a_k x^k \times \mathcal{B}(x)$ pour $0 \leq k < n$. Quelle est la liste retournée par l'appel `sommeFacteur(W,A,B,1)` pour les listes $W = (4,1,9,3,2,1,8,0,2)$, $A = (2,3,1,4)$ et $B = (3,2,2,5)$?
Solution:
$W = (4,10,15,9,17,1,8,0,2)$.

On s'intéresse par la suite à calculer le produit $\mathcal{W}(x) = \mathcal{A}(x) \times \mathcal{B}(x)$. On rappelle que $\mathcal{A}(x)$ et $\mathcal{B}(x)$ sont des polynômes de degré $n-1 \geq 0$. Pour cela, on considère le code python suivant :

```
def produit(A,B):
    n=len(A)
    W = [0]*(2*n-1)
    k=0
    while (k<len(A)) :
            W = sommeFacteur(W,A,B,k)
            print(W)
            k=k+1
    return W
```

Le fonction `len(A)` (resp. `len(B)`) retourne le nombre d'eléments de la liste A (resp. B). L'instruction `W = [0]*(2*n-1)` construit une liste de $2n-1$ termes nuls.

Question 4
Exécutez `produit(A,B)` pour les polynômes $\mathcal{A}(x) = 2 + 3x + x^2 + 4x^3$ et $\mathcal{B}(x) = 3 + 2x + 2x^2 + 5x^3$.
Solution:

```
>>> produit(A,B)
[6, 4, 4, 10, 0, 0, 0]
[6, 13, 10, 16, 15, 0, 0]
[6, 13, 13, 18, 17, 5, 0]
[6, 13, 13, 30, 25, 13, 20]
```

Question 5

1. En supposant que $a_{n-1} \neq 0$ et $b_{n-1} \neq 0$, quel est le degré du polynôme $\mathcal{W}(x)$ en fonction de n ? En déduire la taille de la liste associée W en fonction de n. Justifiez votre réponse.
2. Soit k_0 et W_0 les valeurs initiales de k et de W et pour $j \in \{1, \cdots, n\}$, k_j et W_j les valeurs de k et de W à la fin de la j-ième itération (les itérations sont numérotées de 1 à n). $\mathcal{W}_j(x)$ désigne le polynôme associé à W_j.

   Montrez par récurrence sur j que $k_j = j$ et que W_j est la représentation du polynôme $\mathcal{W}_j(x) = \sum_{i=0}^{j-1} a_i x^i \times \mathcal{B}(x)$.
3. En déduire la validité de la fonction `produit`.

__Solution__:

1. Le terme de plus haut degré de $\mathcal{W}(x)$ est $a_{n-1}x^{n-1} \times b_{n-1}x^{n-1}$, donc le polynôme $\mathcal{W}(x)$ est de degré $2n-2$. La liste est donc de taille $2n-1$.
2. On démontre la propriété $\Pi(j)$: $k_j = j$ et W_j est la représentation du polynôme $\mathcal{W}_j(x) = (\sum_{i=0}^{j-1} a_i x^i) \times \mathcal{B}(x)$ par récurrence faible.

   B Pour $j = 0$, $k_0 = 0$ et W ne contient que des 0 et $\mathcal{W}_0(x) = \sum_{i=0}^{-1} a_i x^i \times \mathcal{B}(x) = 0$. Donc $\Pi(0)$ est vérifiée.

   I Soit j fixé, $j < n$, tel que $\Pi(j)$ soit vérifiée. Alors, d'après la définition de `sommeFacteur`, le polynôme $\mathcal{W}_{j+1}(x)$ associé à W_{j+1} vérifie $\mathcal{W}_{j+1}(x) = \mathcal{W}_j(x) + a_j x^j \times \mathcal{B}(x)$. De plus, par hypothèse de récurrence, $\mathcal{W}_j(x) = \sum_{i=0}^{j-1} a_i x^i \times \mathcal{B}(x)$, donc $\mathcal{W}_{j+1}(x) = \mathcal{W}_j(x) + a_j x^j \times \mathcal{B}(x) = \sum_{i=0}^{j} a_i x^i \times \mathcal{B}(x)$. De plus, $k_{j+1} = k_j + 1 = j + 1$. Donc, $\Pi(j+1)$ est vérifiée.

   C La propriété est vérifiée par récurrence faible.
3. En sortie de boucle, $k = n$ et W_n est la liste associée au polynôme $\mathcal{W}_n(x) = \sum_{i=0}^{n-1} a_i x^i \times \mathcal{B}(x) = \mathcal{A}(x) \times \mathcal{B}(x)$. Donc la fonction retourne $\mathcal{A}(x) \times \mathcal{B}(x)$, elle est donc valide.

Question 6

Supposons que l'appel `sommeFacteur(W,A,B,k)` est en $\Theta(n)$ pour des tableaux et des listes simplement chaînées. Quelle est la complexité de la fonction `produit` si les listes sont stockées dans des tableaux ? Quelle est la complexité de la fonction `produit` si les listes sont stockées dans des listes simplement chaînées ? Justifiez vos réponses.

__Solution__:

La boucle principale est exécutée n fois.

- Si les listes sont stockées dans des tableaux, l'appel de fonction est en $\Theta(n)$ et la recopie du tableau également. Donc, la fonction est en $\Theta(n^2)$.
- Si les listes sont stockées dans des listes chaînées, cela dépend si on mémorise ou non l'adresse des k et $k+n$-ième éléments, ou si l'on recommence du début.
  1. Si on mémorise ces adresses, chaque accès est en $\Theta(1)$. `sommeFacteur(W,A,B,k)` est en $\Theta(n)$, Donc, la fonction est en $\Theta(n^2)$.
  2. Sinon, chaque accès est en $\Theta(n)$, `sommeFacteur(W,A,B,k)` est en $\Theta(n)$; la fonction est donc en $\Theta(n^3)$.

Question 7

Par la suite, on souhaite écrire une fonction pour accélérer le calcul du produit de 2 polynômes de même degré. On suppose que n est une puissance de 2, soit il existe $\ell \in \mathbb{N}$ avec $n = 2^\ell$. Soient alors les polynômes

$\mathcal{A}_0$ et $\mathcal{A}_1$ définis par $\mathcal{A}_0 = \sum_{i=0}^{\frac{n}{2}-1} a_i x^i$ et $\mathcal{A}_1 = \sum_{i=0}^{\frac{n}{2}-1} a_{i+\frac{n}{2}} x^i$. On a alors $\mathcal{A} = \mathcal{A}_0 + x^{\frac{n}{2}}\mathcal{A}_1$. De la même manière, $\mathcal{B}_0 = \sum_{i=0}^{\frac{n}{2}-1} b_i x^i$ et $\mathcal{B}_1 = \sum_{i=0}^{\frac{n}{2}-1} b_{i+\frac{n}{2}} x^i$. On a alors $\mathcal{B} = \mathcal{B}_0 + x^{\frac{n}{2}}\mathcal{B}_1$.

1. Dans cette question, on pose $\mathcal{A}(x) = 2 + 3x + x^2 + 4x^3$ et $\mathcal{B}(x) = 3 + 2x + 2x^2 + 5x^3$. Que valent les polynômes $\mathcal{A}_0$, $\mathcal{A}_1$, $\mathcal{B}_0$, et $\mathcal{B}_1$?
2. Démontrez que dans le cas général $\mathcal{A}_1 \times \mathcal{B}_0 + \mathcal{A}_0 \times \mathcal{B}_1 = (\mathcal{A}_0 + \mathcal{A}_1) \times (\mathcal{B}_0 + \mathcal{B}_1) - \mathcal{A}_0 \times \mathcal{B}_0 - \mathcal{A}_1 \times \mathcal{B}_1$. En déduire que $\mathcal{A} \times \mathcal{B} = x^n \mathcal{A}_1 \times \mathcal{B}_1 + x^{\frac{n}{2}}((\mathcal{A}_0 + \mathcal{A}_1) \times (\mathcal{B}_0 + \mathcal{B}_1) - \mathcal{A}_0 \times \mathcal{B}_0 - \mathcal{A}_1 \times \mathcal{B}_1) + \mathcal{A}_0 \times \mathcal{B}_0$.

<u>**Solution**</u>:

1. $\mathcal{A}_0 = 2 + 3x$, $\mathcal{A}_1 = 1 + 4x$, $\mathcal{B}_0 = 3 + 2x$, et $\mathcal{B}_1 = 2 + 5x$.
2. Si on développe la partie droite de l'égalité : $(\mathcal{A}_0 + \mathcal{A}_1) \times (\mathcal{B}_0 + \mathcal{B}_1) - \mathcal{A}_0 \times \mathcal{B}_0 - \mathcal{A}_1 \times \mathcal{B}_1 = \mathcal{A}_0 \times \mathcal{B}_0 + \mathcal{A}_0 \times \mathcal{B}_1 + \mathcal{A}_1 \times \mathcal{B}_0 + \mathcal{A}_1 \times \mathcal{B}_1 - \mathcal{A}_0 \times \mathcal{B}_0 - \mathcal{A}_1 \times \mathcal{B}_1 = \mathcal{A}_1 \times \mathcal{B}_0 + \mathcal{A}_0 \times \mathcal{B}_1$.
   D'autre part, $\mathcal{A} \times \mathcal{B} = (\mathcal{A}_0 + x^{\frac{n}{2}}\mathcal{A}_1) \times (\mathcal{B}_0 + x^{\frac{n}{2}}\mathcal{B}_1) = x^n \mathcal{A}_1 \times \mathcal{B}_1 + x^{\frac{n}{2}}(\mathcal{A}_1 \times \mathcal{B}_0 + \mathcal{A}_0 \times \mathcal{B}_1) + \mathcal{A}_0 \times \mathcal{B}_0$.
   En remplaçant le terme central par la valeur obtenue précédemment, on obtient l'égalité demandée.

Par la suite, on pose $\mathcal{P}_0 = \mathcal{A}_0 \times \mathcal{B}_0$, $\mathcal{P}_1 = \mathcal{A}_1 \times \mathcal{B}_1$ et $\mathcal{P}_2 = (\mathcal{A}_0 + \mathcal{A}_1) \times (\mathcal{B}_0 + \mathcal{B}_1)$. On obtient

$$\mathcal{A} \times \mathcal{B} = x^n \mathcal{P}_1 + x^{\frac{n}{2}}(\mathcal{P}_2 - \mathcal{P}_0 - \mathcal{P}_1) + \mathcal{P}_0.$$

Question 8

1. Dans cette question, on pose $\mathcal{A}(x) = 2 + 3x + x^2 + 4x^3$ et $\mathcal{B}(x) = 3 + 2x + 2x^2 + 5x^3$. Vérifiez que $\mathcal{P}_0 = 6 + 13x + 6x^2$, $\mathcal{P}_1 = 2 + 13x + 20x^2$, $\mathcal{P}_2 = 15 + 56x + 49x^2$ et $\mathcal{P}_2 - \mathcal{P}_0 - \mathcal{P}_1 = 7 + 30x + 23x^2$. En déduire que $\mathcal{A} \times \mathcal{B} = 6 + 13x + 13x^2 + 30x^3 + 25x^4 + 13x^5 + 20x^6$.
2. On suppose dans cette question que $\mathcal{A}(x) = a_0 + a_1 x$ et $\mathcal{B}(x) = b_0 + b_1 x$. Que valent les polynômes $\mathcal{A}_0$, $\mathcal{A}_1$, $\mathcal{B}_0$, et $\mathcal{B}_1$? Calculez $\mathcal{P}_0$, $\mathcal{P}_1$, $\mathcal{P}_2$.
3. Calculez $\mathcal{P}_0$, $\mathcal{P}_1$, $\mathcal{P}_2$ pour les couples de polynômes suivant :
   — $\mathcal{A}(x) = 2 + 3x$ et $\mathcal{B}(x) = 3 + 2x$
   — $\mathcal{A}(x) = 1 + 4x$ et $\mathcal{B}(x) = 2 + 5x$
   — $\mathcal{A}(x) = 3 + 7x$ et $\mathcal{B}(x) = 5 + 7x$

<u>**Solution**</u>:

1. $\mathcal{P}_0 = (2 + 3x)(3 + 2x) = 6 + 13x + 6x^2$, $\mathcal{P}_1 = (1 + 4x)(2 + 5x) = 2 + 13x + 20x^2$ $\mathcal{P}_2 = (2 + 3x + 1 + 4x)(3 + 2x + 2 + 5x) = (3 + 7x)(5 + 7x) = 15 + 56x + 49x^2$, et $\mathcal{P}_2 - \mathcal{P}_0 - \mathcal{P}_1 = (15 - 6 - 2) + (56 - 13 - 13)x + (49 - 6 - 20) = 7 + 30x + 23x^2$. $\mathcal{A} \times \mathcal{B} = (2 + 13x + 20x^2)x^4 + (7 + 30x + 23x^2)x^2 + 6 + 13x + 6x^2 = 6 + 13x + (7 + 6)x^2 + 30x^3 + (2 + 23)x^4 + 13x^5 + 20x^6$.
2. $\mathcal{A}_0 = a_0$, $\mathcal{A}_1 = a_1$, $\mathcal{B}_0 = b_0$, et $\mathcal{B}_1 = b_1$. $\mathcal{P}_0 = a_0 b_0$, $\mathcal{P}_1 = a_1 b_1$, $\mathcal{P}_2 = (a_0 + a_1)(b_0 + b_1)$ et $\mathcal{P}_2 - \mathcal{P}_0 - \mathcal{P}_1 = a_0 b_1 + a_1 b_0$.
3. — $\mathcal{P}_0 = 6$, $\mathcal{P}_1 = 6$ et $\mathcal{P}_2 = 25$
   — $\mathcal{P}_0 = 2$, $\mathcal{P}_1 = 20$ et $\mathcal{P}_2 = 35$
   — $\mathcal{P}_0 = 15$, $\mathcal{P}_1 = 49$ et $\mathcal{P}_2 = 120$

Question 9
Soit la fonction récursive `produitK(A,B)` suivante :

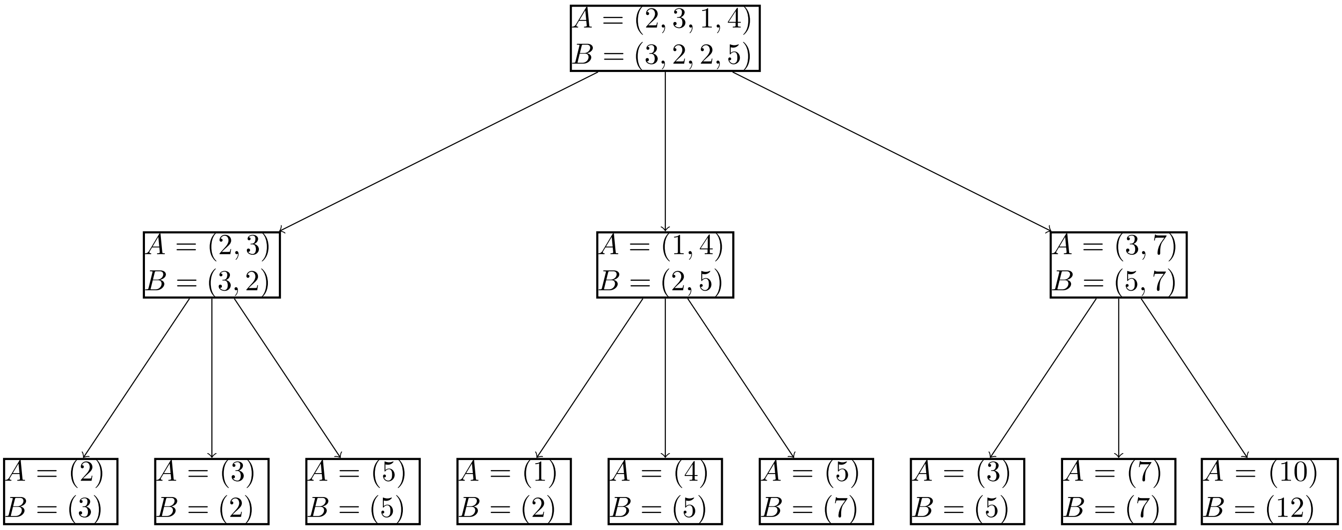
```
def produitK(A,B):
    print "produitK(",A,",",B, ")"
    n=len(A) ;
    A0=A[0:n/2] ; A1=A[n/2:n]
    B0=B[0:n/2] ; B1=B[n/2:n]
    if (n==1):
        W=[A[0]*B[0]]
    else:
        P0=produitK(A0, B0)
        P1=produitK(A1, B1)
        P2=produitK(somme(A0,A1), somme(B0,B1))
        W=calculProduit(P0,P1,P2)
    print A, '*', B, '=', W
    return W
```

L'appel `calculProduit(P0,P1,P2)` retourne la liste correspondant au polynôme $x^n\mathcal{P}_1 + x^{\frac{n}{2}}(\mathcal{P}_2 - \mathcal{P}_0 - \mathcal{P}_1) + \mathcal{P}_0$. Exécutez la fonction pour les listes $A = (2, 3, 1, 4)$ et $B = (3, 2, 2, 5)$. Vous donnerez la liste des affichages et l'arbre des appels.

Indication : Pour éviter les calculs inutiles, utilisez les résultats numériques des questions 7 et 8 et commencez par construire l'arbre des appels.

Solution:

```
ProduitK( [2, 3, 1, 4] , [3, 2, 2, 5] )
produitK( [2, 3] , [3, 2] )
produitK( [2] , [3] )
 [2] * [3] = [6]
produitK( [3] , [2] )
[3] * [2] = [6]
 produitK( [5] , [5] )
[5] * [5] = [25]
[2, 3] * [3, 2] = [6, 13, 6]
produitK( [1, 4] , [2, 5] )
produitK( [1] , [2] )
[1] * [2] = [2]
produitK( [4] , [5] )
[4] * [5] = [20]
produitK( [5] , [7] )
 [5] * [7] = [35]
[1, 4] * [2, 5] = [2, 13, 20]
produitK( [3, 7] , [5, 7] )
produitK( [3] , [5] )
[3] * [5] = [15]
produitK( [7] , [7] )
[7] * [7] = [49]
produitK( [10] , [12] )
[10] * [12] = [120]
 [3, 7] * [5, 7] = [15, 56, 49]
 [2, 3, 1, 4] * [3, 2, 2, 5] = [6, 13, 13, 30, 25, 13, 20]
 [6, 13, 13, 30, 25, 13, 20]
```



Question 10
Démontrez par récurrence la propriété suivante : $\Pi(\ell), \ell \geq 0$: pour tout couple de polynômes $\mathcal{A}$ et $\mathcal{B}$ de degré $n - 1 = 2^\ell - 1$, l'appel `ProduitK(A,B)` se termine et retourne $\mathcal{A} \times \mathcal{B}$.

<u>Solution</u>:
On démontre la propriété par récurrence faible sur ℓ.

B Pour $\ell = 0$, $n = 1$. Dans ce cas, $A = (a_0)$, $B = (b_0)$ et la fonction retourne la liste $(a_0 \times b_0)$. Donc, la fonction se termine et retourne $\mathcal{A} \times \mathcal{B}$. $\Pi(1)$ est donc vérifiée.

I Soit $\ell \geq 1$ tel que $\Pi(\ell - 1)$ soit vérifiée. Soient alors deux polynômes $\mathcal{A}$ et $\mathcal{B}$ de degré $n - 1 = 2^\ell - 1$. Par construction, les polynômes $\mathcal{A}_0$, $\mathcal{A}_1$, $\mathcal{B}_0$, et $\mathcal{B}_1$ associés aux listes $A0$, $A1$, $B0$ et $B1$ sont tous de degré maximum $\frac{n}{2} - 1 = 2^{\ell-1} - 1$. Donc, par hypothèse de récurrence, les 3 appels récursifs se terminent et retournent respectivement les listes $P0$, $P1$ et $P2$ associées aux polynômes $\mathcal{P}_0$, $\mathcal{P}_1$, et $\mathcal{P}_2$. L'appel `calculProduit(P0,P1,P2)` retourne la liste correspondant au polynôme $x^n\mathcal{P}_1 + x^{\frac{n}{2}}(\mathcal{P}_2 - \mathcal{P}_0 - \mathcal{P}_1) + \mathcal{P}_0$, ce qui correspond bien à $\mathcal{A} \times \mathcal{B}$ d'après la question 7.
Ainsi, la fonction se termine et W contient la liste associée à $\mathcal{A} \times \mathcal{B}$. $\Pi(\ell)$ est donc vérifiée.

Question 11 – Facultative
On suppose que `calculProduit(P0,P1,P2)` est de complexité $\Theta(n)$ pour des tableaux ou des listes simplements chaînées.

Démontrez que la complexité de la fonction `produitK(A,B)` pour deux polynômes de degré $n - 1$ vérifie $c(n) = 3c(\frac{n}{2}) + \alpha \times n$. En déduire que l'algorithme est $\mathcal{O}(\log_2(n) \times n^{\log_2(3)})$.

Indication : on peut remarquer que pour $\forall \ell \geq 0$, $3^\ell = 2^{\ell \log_2(3)}$.

<u>Solution</u>:
Pour un appel avec deux listes de taille n, il y a 3 appels récursifs et des appels à des fonctions toutes en $\Theta(n)$. On en déduit que $c(n) = 3c(\frac{n}{2}) + \alpha \times n$.

On résout alors cette équation par subsituttion. On pose $n = 2^\ell$ et $u_\ell = c(n)$. On a alors $u_\ell = 3u_{\ell-1} + \alpha 2^\ell = 3(3u_{\ell-2} + \alpha 2^{\ell-1}) = 3^\ell u_0 + \alpha \sum_{i=0}^{\ell} 3^i 2^{\ell-i}$. Or, $\sum_{i=0}^{\ell} 3^i 2^{\ell-i} \leq \ell 3^\ell$ et donc $u_\ell \leq \beta \ell 3^\ell$.

On observe que $\log_2(3^\ell) = \ell \log_2(3)$ et donc $\ell 3^\ell = \ell \times 2^{\ell \log_2(3)}$. Comme $\ell = \log_2(n)$, on obtient que $\ell 3^\ell = \log_2(n) \times n^{\log_2(3)}$.