

# Corrigé du Contrôle d'Informatique INF 311

Sujet proposé par P. Chassignet et F. Nielsen

13 juillet 2010

Seuls les documents fournis dans le cadre du cours et les notes personnelles sont autorisés.

Durée : 2 heures. Les étudiants EV2 ont le droit à 30 minutes supplémentaires.

Les trois exercices sont indépendants. Un barème pour chaque exercice, pour un total de 28 points, est donné à titre indicatif. Nous donnons le nombre de lignes attendues pour la réponse à chaque question ; si la réponse que vous envisagez est beaucoup plus longue, il est probable qu'il s'agisse d'une méthode trop compliquée.

On attachera une grande importance à la clarté, à la précision et à la concision de la rédaction. La réponse à certaines questions demandent l'estimation de valeurs numériques ; il suffit de trouver un ordre de grandeur et une calculatrice n'est pas utile.

## Exercice 1. Un catalogue de chansons

barème envisagé : 15 points

Cet exercice considère la gestion d'un catalogue de chansons.

- (1a) [1 point] Écrivez une classe `Chanson` dont chaque objet contient les champs suivants : `artiste` et `titre` qui sont des chaînes de caractères, `duree` qui est un entier (indiquant les secondes).

Écrivez ensuite un constructeur qui permet d'initialiser les 3 champs d'un objet de cette classe.

(réponse attendue : une dizaine de lignes)

### **Solution:**

```
public class Chanson {
    String artiste, titre;
    int duree;

    Chanson(String a, String t, int d) {
        this.artiste = a;
        this.titre = t;
        this.duree = d;
    }
}
```

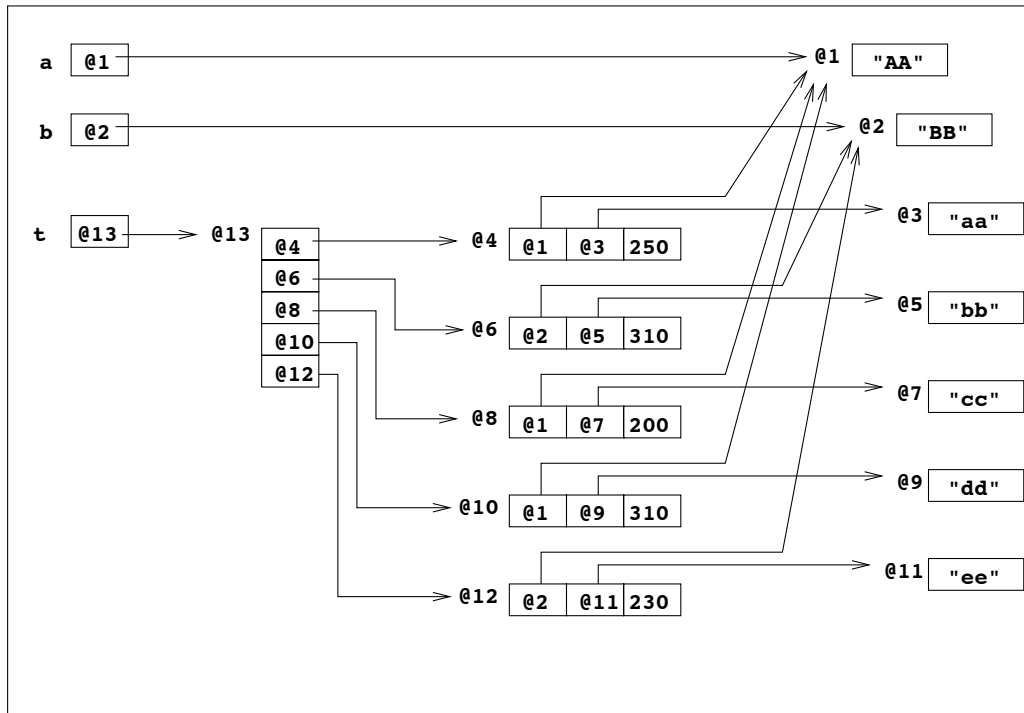


FIG. 1 – Représentation de la mémoire associée à un tableau de chansons

La figure 1 représente la mémoire associée à un tableau de 5 chansons, de durée respective 250, 310, 250, 310 et 230. Le tableau est référencé par la variable `t` et il y a deux artistes dont les noms sont référencés par les variables `a` et `b`.

- (1b) [1 point] Écrivez des instructions pour construire cet exemple. Les valeurs de l'exemple doivent être mises explicitement dans le code (il ne s'agit pas de lire ces données).  
(réponse attendue : 5 à 10 lignes)

**Solution:**

```
String a = "AA";
String b = "BB";
Chanson[] t = {
    new Chanson(a, "aa", 250),
    new Chanson(b, "bb", 310),
    new Chanson(a, "cc", 200),
    new Chanson(a, "dd", 310),
    new Chanson(b, "ee", 230) };
```

- (1c) [1 point] Écrivez une fonction **statique** `nombreVerifiantDuree` qui prend en paramètres un tableau de chansons et deux entiers `min` et `max` et qui retourne le nombre de chansons dont la durée `d` vérifie  $\min \leq d \leq \max$ .  
Cette fonction sera dans la classe `Chanson`.

(réponse attendue : quelques lignes)

**Solution:**

```
static int nombreVerifiantDuree(Chanson[] t,
                                int min, int max) {
    int n = 0;
    for (int i = 0; i < t.length; ++i)
        if (min <= t[i].duree && t[i].duree <= max)
            ++n;
    return n;
}
```

On ajoute la classe `Catalogue` qui commence comme suit :

```
public class Catalogue {
    Chanson[] chansons;

    Catalogue(Chanson[] t) {
        this.chansons = t;
    }
    ...
}
```

- (1d) [1 point] Écrivez une méthode **d'objet** `chansonsVerifiantDuree` de la classe `Catalogue` qui prend en paramètres deux entiers `min` et `max` et qui retourne un tableau des chansons de ce catalogue dont la durée  $d$  vérifie  $\min \leq d \leq \max$ . On ne doit pas recopier les chansons (seulement les références) et on doit utiliser la fonction `nombreVerifiantDuree` de la question précédente.

(réponse attendue : une dizaine de lignes)

**Solution:**

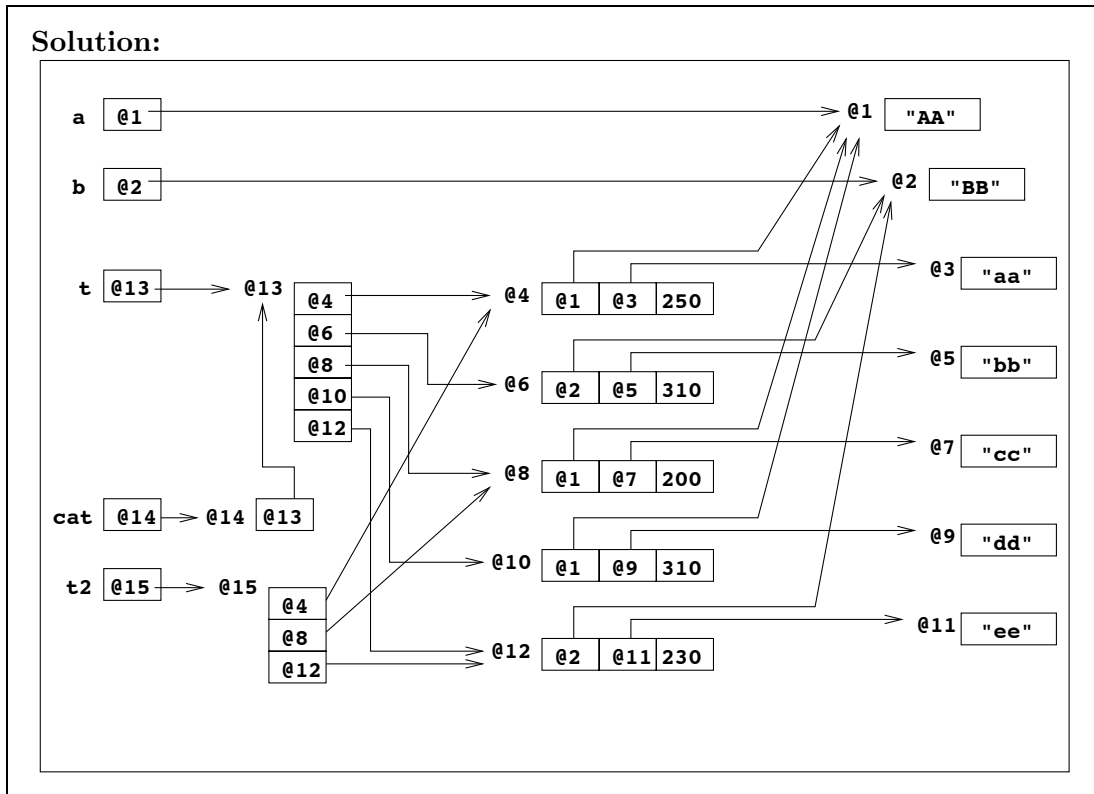
```
Chanson[] chansonsVerifiantDuree(int min, int max) {
    int n = Chanson.nombreVerifiantDuree(this.chansons,
                                          min, max);

    Chanson[] t = new Chanson[n];
    n = 0;
    for (int i = 0; i < this.chansons.length; ++i)
        if (min <= this.chansons[i].duree
            && this.chansons[i].duree <= max) {
            t[n] = this.chansons[i];
            ++n;
        }
    return t;
}
```

À la suite du code écrit pour la question (1b), on ajoute les deux instructions :

```
Catalogue cat = new Catalogue(t);
Chanson[] t2 = cat.chansonsVerifiantDuree(210, 305);
```

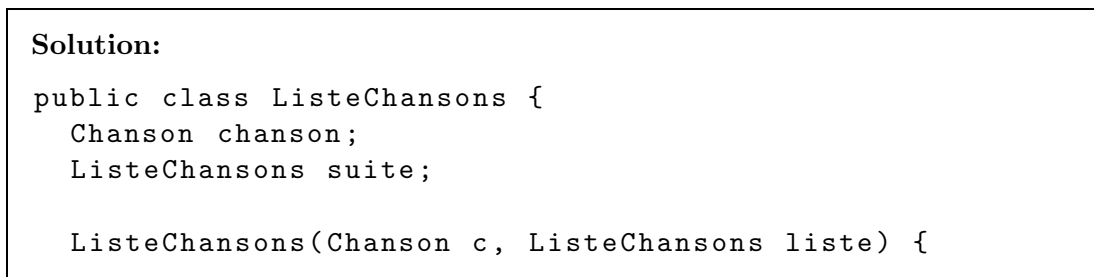
(1e) [1 point] Recopiez le dessin de la mémoire donné pour la question (1b) et complétez le pour représenter le résultat de ces deux instructions.



On désire maintenant gérer des listes de chansons. Pour cela, nous allons définir une structure de liste chaînée dont chaque élément contient (une référence à) une chanson et une référence à la suite de la liste.

(1f) [1 point] Écrivez une définition de la classe `ListeChansons` dont chaque objet est un élément de liste chaînée, et le constructeur qui prend un paramètre de type `Chanson` et un paramètre de type `ListeChansons` et qui permet d'ajouter un élément en tête d'une liste chaînée.

(réponse attendue : une dizaine de lignes)



```
        this.chanson = c;
        this.suite = liste;
    }
}
```

- (1g) [1 point] Écrivez une méthode **d'objet** `chansonsDe` de la classe `Catalogue` qui prend le nom d'un artiste en paramètre et qui retourne la liste chaînée de toutes les chansons de l'artiste qui figurent dans ce catalogue. L'ordre dans la liste est sans importance. (réponse attendue : de 5 à 10 lignes)

**Solution:**

```
ListeChansons chansonsDe(String artiste) {
    ListeChansons liste = null;
    for (int i = 0; i < this.chansons.length; ++i)
        if (this.chansons[i].artiste.equals(artiste))
            liste = new ListeChansons(this.chansons[i], liste);
    return liste;
}
```

Étant donnée une liste de chansons, on veut maintenant calculer la durée cumulée (en secondes) des chansons de cette liste.

- (1h) [1 point] Écrivez une fonction **statique** `duree`, dans la classe de votre choix, qui prend en paramètre une liste chaînée et qui retourne la durée cumulée des chansons de la liste. La liste doit être parcourue de manière **récursive**. (réponse attendue : quelques lignes)

**Solution:**

```
static int duree(ListeChansons liste) {
    if (liste == null)
        return 0;
    else
        return liste.chanson.duree + duree(liste.suite);
}
```

On veut maintenant couper une liste pour que sa durée cumulée ne dépasse pas une certaine durée limite.

- (1i) [2 points] Écrivez une fonction `static ListeChansons couper(ListeChansons liste, int duree)` **récursive** qui retourne la liste coupée de la manière suivante. Si la première chanson de la liste a une durée strictement supérieure au paramètre `duree`, la fonction retourne `null`, autrement on conserve cette chanson et on traite la suite de la liste.

Cette fonction ne doit pas construire de nouvel objet et son coût maximal en temps doit être proportionnel à la longueur de la liste. On ne cherche pas à préserver la liste initiale.

(réponse attendue : 5 à 10 lignes)

**Solution:**

```
static ListeChansons couper(ListeChansons liste,
                             int duree) {
    if (liste == null || duree <= 0)
        return null;
    if (liste.chanson.duree > duree)
        return null;
    liste.suite = couper(liste.suite,
                          duree - liste.chanson.duree);
    return liste;
}
```

- (1j) [1 point] Si la première chanson ne convient pas, on peut regarder dans la suite de la liste pour voir si on trouve une chanson qui convient, etc. Écrivez une variante de la fonction `couper` qui fonctionne selon ce principe. Il suffit normalement de modifier le traitement d'un cas particulier.

(réponse attendue : 5 à 10 lignes)

**Solution:**

```
static ListeChansons couperMieux(ListeChansons liste,
                                  int duree) {
    if (liste == null || duree <= 0)
        return null;
    if (liste.chanson.duree > duree)
        return couperMieux(liste.suite, duree);
    liste.suite = couperMieux(liste.suite,
                              duree - liste.chanson.duree);
    return liste;
}
```

On cherche maintenant à trier les chansons d'un album par ordre de durée croissante.

- (1k) [1 point] Écrivez une fonction

```
static ListeChansons elementDureeMinimale(ListeChansons liste)
```

qui retourne un élément de la liste qui référence une chanson de durée minimale (la première rencontrée s'il y en a plusieurs). Vous devez prévoir le cas où la liste est vide. Cette fonction ne doit pas construire de nouvel objet et ne doit pas détruire la liste.

(réponse attendue : une dizaine de lignes)

**Solution:** voir ci-dessous

- (11) [1 point] Écrivez une fonction qui permet d'échanger les chansons référencées par deux éléments (non null) d'une liste, sans modifier l'ordre des éléments de cette liste. Cette fonction ne doit pas construire de nouvel objet.  
(réponse attendue : quelques lignes)

**Solution:** voir ci-dessous

- (1m) [2 points] Écrivez une fonction **récursive**  
`static void triSelection(ListeChansons liste)`  
qui utilise les fonctions précédentes et qui trie les chansons de `liste` par ordre de durée croissante en utilisant le principe du tri par sélection.  
Cette fonction ne doit pas modifier l'ordre des éléments de la liste.  
(réponse attendue : quelques lignes)

**Solution:**

```
static ListeChansons elementDureeMinimale
                                (ListeChansons lc) {
    ListeChansons lMin = null;
    int d = Integer.MAX_VALUE;
    while (lc != null) {
        if (lc.chanson.duree < d) {
            d = lc.chanson.duree;
            lMin = lc;
        }
        lc = lc.suite;
    }
    return lMin;
}

static void echangerChansons(ListeChansons lc1,
                              ListeChansons lc2) {
    Chanson tmp = lc1.chanson;
    lc1.chanson = lc2.chanson;
    lc2.chanson = tmp;
}

static void triSelection(ListeChansons lc) {
    if (lc != null) {
        echangerChansons(lc, elementDureeMinimale(lc));
        triSelection(lc.suite);
    }
}
```

Quelle est la complexité dans le pire des cas ? dans le meilleur des cas ?  
(réponse attendue : quelques lignes)

**Solution:** Dans tous les cas, `elementDureeMinimale` doit parcourir toute la liste qu'elle reçoit en paramètre. Les autres opérations se font en temps constant. Cela donne un coût en  $O(n^2)$  dans tous les cas.

## Exercice 2. Coefficients binomiaux

barème envisagé : 10 points

Pour  $n \geq k \geq 0$ , le coefficient binomial  $C_{n,k}$  est le coefficient de  $X^k$  dans le développement de  $(1 + X)^n$ . Par exemple,  $(1 + X)^4 = 1 + 4.X + 6.X^2 + 4.X^3 + X^4$  et  $C_{4,2} = 6$ .

On sait que  $C_{n,k} = \frac{n!}{k!(n-k)!}$ , mais dans cet exercice on va considérer le calcul par la récurrence :

$C_{n,k} = C_{n-1,k-1} + C_{n-1,k}$ , avec les cas particuliers  $C_{n,n} = 1$  et  $C_{n,0} = 1$ .

(2a) [1 point] Écrivez une fonction statique **réursive** qui calcule  $C_{n,k}$  pour  $n \geq k \geq 0$  en utilisant la récurrence donnée ci-dessus.

(réponse attendue : quelques lignes)

**Solution:**

```
static long binomial(int n, int k) {
    if (k == 0 || n == k)
        return 1;
    return binomial(n - 1, k - 1) + binomial(n - 1, k);
}
```

Le type `int` est aussi accepté pour cette question.

(2b) [1 point] Au maximum, combien d'appels à cette fonction se trouvent dans la pile d'exécution pendant le calcul de  $C_{n,k}$  ? (réponse attendue : quelques lignes)

**Solution:** Comme  $n$  décroît strictement et reste positif, on empile au maximum  $n$  appels.

(2c) [1 point] En fonction de  $n$  et  $k$ , combien d'appels à cette fonction sont requis pour calculer  $C_{n,k}$  ? (réponse attendue : quelques lignes)

**Solution:** Le nombre d'appel pour  $C_{n,n}$  et  $C_{n,0}$  est 1. Dans le cas général, le nombre d'appels pour  $C_{n,k}$  est le nombre d'appels pour  $C_{n-1,k-1}$  plus le nombre d'appels pour  $C_{n-1,k}$ . Cela définit exactement la même récurrence. Le nombre d'appels pour calculer  $C_{n,k}$  est donc  $C_{n,k}$ .

On s'intéresse maintenant aux coefficients centraux ( $k = n/2$ ) pour  $n$  pair.

On a  $C_{2k,k} < 4^k$  et aussi, pour  $k \leq 31$ ,  $\frac{4^k}{10} < C_{2k,k}$ .

- (2d) [1 point] Quel type primitif faut-il utiliser pour représenter la valeur exacte de  $C_{60,30}$  ?  
(réponse attendue : quelques lignes)

**Solution:** On a  $C_{60,30} < 4^{30} = 2^{60}$ , le calcul exact est donc réalisable dans le type `long` dont la plus grande valeur est  $2^{63} - 1$ . On ne peut pas utiliser le type `int` dont la plus grande valeur est  $2^{31} - 1$ .

On peut considérer que le calcul de  $C_{30,15}$  prend de l'ordre de 2 secondes au moyen de la fonction récursive.

- (2e) [1 point] Quel est le problème pour le calcul de  $C_{60,30}$  au moyen de la fonction récursive ?  
(réponse attendue : quelques lignes)

**Solution:** Le temps de calcul est de l'ordre de  $2 \cdot 2^{30}$  secondes !

On s'intéresse maintenant au calcul itératif des coefficients binomiaux.

- (2f) [1 point] Écrivez une fonction **itérative** `static long[] binomiaux(long[] t)` qui suppose que le tableau passé en paramètre contient tous les coefficients binomiaux  $C_{n-1,k}$  pour un certain  $n$  qui est donné par la longueur du tableau, c'est-à-dire que  $t[k] = C_{n-1,k}$ , pour tout  $k < n$ , et qui retourne un tableau qui contient les  $C_{n,k}$ , pour tout  $k \leq n$ .  
(réponse attendue : une dizaine de lignes)

**Solution:**

```
static long[] binomiaux(long[] t) {
    long[] t2 = new long[t.length + 1];
    t2[0] = 1;
    for (int k = 1; k < t.length; k++)
        t2[k] = t[k] + t[k - 1];
    t2[t.length] = 1;
    return t2;
}
```

- (2g) [1 point] Écrivez une fonction `main` qui utilise la fonction précédente pour calculer les valeurs  $C_{n,k}$  pour tous les  $k \leq n$ , pour une valeur de  $n$  donnée, et qui les affiche.  
(réponse attendue : une dizaine de lignes)

**Solution:**

```
public static void main(String[] args) {
    int n = Integer.parseInt(args[0]); // par exemple
    long[] t = { 1 }; // initialise avec binomial(0,0)
    for (int i = 1; i <= n; i++)
```

```

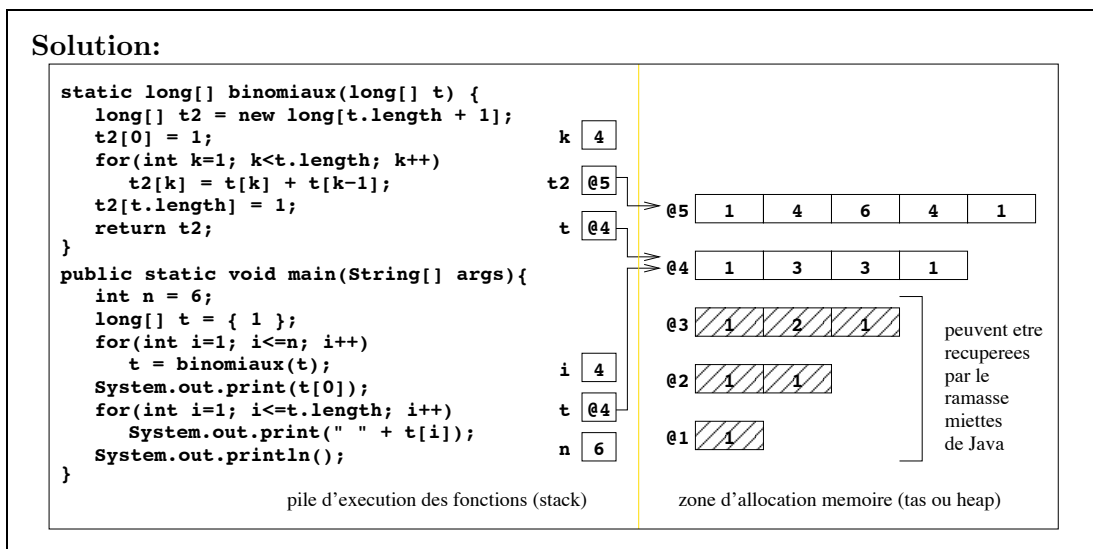
    t = binomiaux(t);
    System.out.print(t[0]);
    for (int i = 1; i < t.length; i++)
        System.out.print(" " + t[i]);
    System.out.println();
}

```

- (2h) [1 point] Quel est le coût en temps de ce calcul ?  
(réponse attendue : quelques lignes)

**Solution:** La fonction `binomiaux` a un temps d'exécution  $O(\ell)$ , où  $\ell$  est la longueur du tableau. Le coût total en temps est  $O(n^2)$ .

- (2i) [1 point] Dessinez l'état de la mémoire (pile d'exécution et tas) lorsque, pour  $n = 6$ , la fonction `binomiaux` est sur le point de retourner le tableau qui contient la valeur de  $C_{4,2}$ . Marquez sur ce dessin les zones de mémoire qui peuvent être déjà récupérées par le ramasse-miettes de Java.



- (2j) [1 point] Quel est le coût en mémoire de ce calcul ?  
(réponse attendue : quelques lignes)

**Solution:** Le coût en mémoire est  $O(n)$ . Le coût maximum est atteint durant le dernier appel à `binomiaux`.

### Exercice 3. Le programme mystérieux

barème envisagé : 3 points

Soit le programme :

```
public class Mystere {
    public static void mystere(int[] t, int b, int p) {
        if (b == 0) {
            t[b] += p;
            // ici
        } else
            mystere(t, b / 2, p + 1);
    }
    public static void main(String[] args) {
        int[] tableau = { 2, 1, 4, 3 };
        mystere(tableau, 3, 0);
        System.out.println(tableau[0]);
    }
}
```

(3a) [2 points] Que fait la fonction `mystere` en général et que fait le programme ci-dessus lorsqu'on l'exécute ?

(réponse attendue : une dizaine de lignes)

**Solution:** La fonction `mystere` est récursive. Le paramètre `b` est divisé par 2 à chaque appel. Comme il s'agit de la division euclidienne, la condition d'arrêt `b==0` est toujours atteinte. Le paramètre `p` compte le nombre d'appels récursifs, c'est-à-dire le nombre de divisions par 2 requises pour obtenir `b==0`.

Ce nombre d'appels est alors ajouté à `t[0]`, ce qui affecte le tableau passé par référence et la nouvelle valeur de `tableau[0]` est affichée par la fonction `main`.

Avec les valeurs numériques données, il faut deux divisions par 2 pour passer de 3 à 0, donc la valeur de `tableau[0]` est augmentée de 2 et le programme affiche 4.

(3b) [1 point] Dessinez l'état de la mémoire (pile d'exécution et tas) lorsque l'exécution passe par l'emplacement qui est marqué par le commentaire *ici*.

**Solution:**

