Correction interro n°3

Cours d'analyse, algorithmique

—Master 2 CCI—

Tout document, calculatrice, téléphone portable ou ordinateur est interdit.

Les exercices sont indépendants. Si l'on ne sait pas justifier une question, on peut néanmoins utiliser la réponse dans la suite.

Durée : 1 heure

▸ Exercice 1. (Remplissage bit-retourné d'un tableau)

On considère les fonctions `C` suivantes :

```
1   void bitrev_rec(int T[], int min, int max, int n, int pow2)
2   {
3     int mid;
4     if (min == max) T[min] = n;
5     else
6       {
7         mid = (min+max)/2;
8         bitrev_rec(T, min, mid, n, 2*pow2);
9         bitrev_rec(T, mid+1, max, n+pow2, 2*pow2);
10      }
11  }
12  void bitrev(int T[], int taille)
13  {
14    bitrev_rec(T, 0, taille-1, 0, 1);
15  }
```

1. Quel est, après un appel `bitrev(tab, 8)`, le contenu du tableau `T` ?
   Indication : Le tableau contient les nombres de 0 à 7 dans un certain ordre.

   ✂ ----------

   Voici une trace de l'exécution de `bitrev`

   ```
   Appel de bitrev avec min=0, max=7, n=0, pow2=1
   mid = 3
   Appel de bitrev avec min=0, max=3, n=0, pow2=2
   mid = 1
   Appel de bitrev avec min=0, max=1, n=0, pow2=4
   mid = 0
   ```

```
Appel de bitrev avec min=0, max=0, n=0, pow2=8
Affectation T[0] <- 0
Appel de bitrev avec min=1, max=1, n=4, pow2=8
Affectation T[1] <- 4
Appel de bitrev avec min=2, max=3, n=2, pow2=4
mid = 2
Appel de bitrev avec min=2, max=2, n=2, pow2=8
Affectation T[2] <- 2
Appel de bitrev avec min=3, max=3, n=6, pow2=8
Affectation T[3] <- 6
Appel de bitrev avec min=4, max=7, n=1, pow2=2
mid = 5
Appel de bitrev avec min=4, max=5, n=1, pow2=4
mid = 4
Appel de bitrev avec min=4, max=4, n=1, pow2=8
Affectation T[4] <- 1
Appel de bitrev avec min=5, max=5, n=5, pow2=8
Affectation T[5] <- 5
Appel de bitrev avec min=6, max=7, n=3, pow2=4
mid = 6
Appel de bitrev avec min=6, max=6, n=3, pow2=8
Affectation T[6] <- 3
Appel de bitrev avec min=7, max=7, n=7, pow2=8
Affectation T[7] <- 7
```

On obtient donc `[0 4 2 6 1 5 3 7]` ;

---

2. Montrer que lors d'une exécution de `bitrev(tab, n)` le nombre d'appels à la fonction `bitrev_rec` est $2n - 1$.

On montre ce résultat par récurrence : On suppose le résultat vrai pour les appels récursif et on montre qu'il est vrai pour l'appel général :
Notons $R(\texttt{max}, \texttt{min})$ le nombre d'appels effectués lors de l'exécution de `bitrev_rec(T, min, max, n, pow2)`.
On veut montrer que pour tout `max`, `min`,

$$R(\texttt{max}, \texttt{min}) = 2(\texttt{max} - \texttt{min} + 1) - 1\,.$$

- Montrons que cette propriété est vraie dans le cas de base : si $\texttt{min} = \texttt{max}$, il n'y a pas d'appel récursif, le nombre d'appel est $1 = 2 * 1 - 1 = 2i - 1$.
- On suppose la propriété vraie lors des appels récursifs, on montre qu'elle est vraie en général : On exécute `bitrev_rec(T, min, max, n, pow2)` où

  $$\texttt{max} - \texttt{min} + 1 = n + 1\,.$$

  Cela induit les deux appels récursifs suivants :

```
\texttt{bitrev_rec(T, min, mid, n, pow2)}
\texttt{bitrev_rec(T, mid+1, max, n, pow2)}
```

Par hypothèse de récurrence :
- $R(\texttt{min}, \texttt{mid}) = 2(\texttt{mid} - \texttt{min}) + 1$
- $R(\texttt{mid}+1, \texttt{max}) = 2(\texttt{max} - (\texttt{mid}+1)) + 1$ appels.

Le nombre total d'appels est donc

$$\begin{aligned} R(\texttt{max}, \texttt{min}) &= 1 + R(\texttt{min}, \texttt{mid}) + R(\texttt{mid}+1, \texttt{max}) \\ &= 1 + (2(\texttt{mid} - \texttt{min}) + 1) + (2(\texttt{max} - (\texttt{mid}+1)) + 1) \\ &= 2\texttt{max} - 2\texttt{min} + 1 = 2(\texttt{max} - \texttt{min} + 1) - 1 = 2(n+1) - 1 \end{aligned}$$

3. Quel est la complexité de `bitrev` ?

On a montrer que le nombre d'appel est $2n + 1$. C'est donc une complexité en $\Theta(n)$, c'est-à-dire linéaire.

▸ Exercice 2. (Tri par insertion)

1. Rappeler l'algorithme du tri par insertion.
2. Exécuter pas à pas le tri par insertion sur le tableau `[12 1 8 5 9 7]`.
3. Combien de comparaisons d'éléments avez vous effectuées ?
4. Combien de copies d'éléments avez vous effectuées ?

▸ Exercice 3. (Multiplication)

On dispose d'une machine qui compte en base 2 mais ne possède ni l'instruction de multiplication ni celle de division. En revanche la machine sait faire les additions, les soustractions et les comparaisons.

1. À l'aide d'une boucle écrire un algorithme qui calcule nm. Quel est la complexité de cet algorithme ?

```
res <- 0
pour i de 1 à n faire
    res <- res + m
retourner res
```

Le nombre d'addition est n, la complexité est $\Theta(n)$.

Pour aller plus vite, on dispose également des instructions de décalage qui permettent les multiplications et divisions par deux : si n est un entier

- $n \gg 1$ renvoie $\lfloor n/2 \rfloor$
- $n \ll 1$ renvoie $2n$.

Enfin, grâce au masques binaires on sait tester la parité d'un nombre : $n \& 1$ renvoie 0 si n est pair et 1 sinon. On a donc les identités suivantes :

- Si n est pair, en posant $k = n \gg 1$ on a $n = 2k$ et
$$nm = 2km = (km) \ll 1 = k(m \ll 1)\,.$$
- Si $n =$ est impair, en posant $k = n \gg 1$ on a $n = 2k + 1$ et
$$nm = (2k+1)m = 2km + m = (km) \ll 1 + m = k(m \ll 1) + m\,.$$

2. En déduire un algorithme de type diviser pour régner qui calcule le produit mn.

```
Algorithme Multiplication(n,m):
    si n = 0 retourner 0
    k <- n >> 1
    si n & 1 = 0 alors
        retourner Multiplication(k, m) << 1
    sinon
        retourner Multiplication(k, m) << 1 + m
```

On peut aussi l'écrire en itératif :

```
Algorithme Multiplication(N,M)
    res <- 0; n <- N; m <- M
    tant que n <> 0 faire
        // Invariant de boucle: N*M = res + n*m
        si n & 1 = 1 alors
            res <- res + m
        n <- n >> 1
        m <- m << 1
    retourner res
```

3. Quel est la complexité de cet algorithme ?

Soit $R(n)$ le nombre d'appels récursif pour `Multiplication`(n, m). On $R(0) = 1$ et $R(n) = 1 + R(n/2)$. Donc $R(n) \in O(\log(n))$.

L'algorithme itératif est du type :

```
tant que n <> 0 faire
    ...
    n <- n / 2
```

Il est donc aussi en $O(\log(n))$.