DM 2 : EXPONENTIATION RAPIDE – CORRIGÉ

Corrigé de la partie « exponentiation rapide »

1.

```
1 def naif(a,n):
2     p = 1
3     for k in range(n):
4         p = p*a
5     return p
```

L'algorithme ne présente aucun obstacle à la terminaison : la boucle for termine après n itérations et on réalise un nombre fini d'opérations dans et en dehors de la boucle.
La seule opération élémentaire est la multiplication ligne 4, réalisée n fois dans tous les cas. Donc on réalise (dans le pire cas) n opérations élémentaires, l'algorithme a une complexité d'ordre n.

2.

k	p_k	a_k	n_k	o_k
0	1	5	84	0
1	1	5^2	42	5
2	1	5^4	21	5
3	5^4	5^8	10	6
4	5^4	5^{16}	5	5
5	5^{20}	5^{32}	2	6
6	5^{20}	5^{64}	1	5
7	5^{84}	5^{128}	0	6

```
1 def exporap(a,n):
2     p = 1
3     while n>0:
4         if n%2==1:
5             p = p*a
6         a = a*a
7         n = n//2
8     return p
```

On s'arrête à la fin de la 7ème itération car $n_7 = 0$. On retourne alors $p_7 = 5^{84}$.

3. (cf le tableau ci-dessus, où on a noté o_k le nombre d'opérations élémentaires effectuées à la fin de la k-ième itération de la boucle `while`). Les 7 itérations réalisent 38 opérations mais la condition « `n > 0` » l. 3 est exécutée une 8ème fois pour sortir de la boucle, pour un total de $38+1 = \boxed{39}$ opérations élémentaires effectuées par l'instruction `exporap(5,84)`. En comparaison, `naif(5,84)` réalise 84 opérations élémentaires, ce qui est bien plus élevé.
4. Il suffit de montrer que la boucle `while` l. 3 termine. Montrons que n_k est un variant de boucle.
   (a) Comme $n_k = n_{k-1}//2$ et que n_0 est un entier, on a $n_k \in \mathbb{Z}$.
   (b) Comme $n_k = n_{k-1}//2$, on a $n_{k-1} \geq 0 \implies n_k \geq 0$. D'où, comme $n_0 \geq 0$, on a immédiatement $n_k \geq 0$ pour tout k.
   (c) Comme $n_k = n_{k-1}//2$, on a $n_{k-1} \leq \dfrac{n_k}{2} < n_k$ sauf si $n_k = 0$. Or, si $n_k = 0$, on sort de la boucle avant de commencer l'itération $k+1$, si bien que lorsque ce cas arrive, la boucle termine. Si on exclut ce cas, on a donc $n_{k-1} < n_k$: la suite (n_k) est strictement décroissante.

   Finalement, (n_k) est bien un variant de boucle : la boucle termine.

On s'intéresse maintenant à la complexité de `exporap`, qu'on exprimera par rapport à l'argument n. On fixe donc $n \in \mathbb{N}^*$ et on cherche à déterminer le nombre d'opérations de `exporap(a,n)` dans le pire cas.

5. Non : modifier a (et donc a_0) ne va conduire qu'à modifier a_k et p_k, mais cela n'aura aucun impact sur n_k : n_k est toujours modifié par la l. 7, à chaque itération, indépendamment des valeurs de a_k et p_k. Ainsi, il n'y a pas de pire cas à n fixé.
6. Les opérations élémentaires de `exporap` sont :
   - l.3 : « `n>0` » : une opération élémentaire
   - l.4 : « `n%2==1` » : deux op. élém.
   - l.5 : « `p*a` » : une op. élém.

- l.6 : « `a*a` » : une op. élém.
- l.7 : « `n//2` » : une op. élém.

Ainsi, chaque itération réalise donc 5 op. élém. si la variable `n` est paire (la l.5 n'est pas exécutée) et 6 op. élém. si `n` est impaire. Une fois l'itération terminée, on effectue ensuite (comme il n'y a pas de pire cas) autant d'opérations que `exporap(a,n//2)`. Donc, on a $\boxed{c_{2n} = 5 + c_n}$, tandis que $\boxed{c_{2n+1} = 6 + c_n}$.

7. Par ce qui précède, on a (si $q \geq 1$)

$$c_{2^q} = 5 + c_{2^{q-1}}$$

et donc par récurrence immédiate (pour tout q)

$$c_n = c_{2^q} = 5q + c_{2^0} = 5q + c_1 = 5q + 6 + c_0 = 5q + 7$$

Or, $n = 2^q$ donc $q = \log_2(n)$. Ainsi, $\boxed{c_n = 5\log_2(n) + 7}$.

Corrigé de la partie « TP 8 Exercice 3 »

```
1  def dichotomie(T,x):
2      a = 0
3      b = len(T)-1
4      while b-a > 0 :
5          m = (a+b)//2
6          if T[m] == x:
7              return m
8          elif T[m] < x:
9              a = m+1
10         else:
11             b = m-1
12     return None        # x n'est pas dans T
```

Il suffit de montrer la terminaison de la boucle `while` ligne 6. On note a_k, b_k les valeurs des variables `a,b` après k itérations de cette boucle (idem pour les autres variables). On pose

$$v_k = b_k - a_k$$

Montrons que v_k est un variant de boucle.

- Par construction, m_k, a_k et b_k sont toujours des entiers, donc $\boxed{v_k \in \mathbb{Z}}$.
- Il est clair que $v_0 \geq 0$ (sauf si T est vide, auquel cas la terminaison est claire).
  - S'il existe k tel que $v_k < 0$, alors on sort de la boucle au début de l'itération $k+1$ (car alors $b_k - a_k < 0$) et donc la terminaison est claire.
  - Sinon, les cas ci-dessus étant exclus, on a donc $\boxed{v_k \geq 0}$[1].
- Si $T[m_k]$ vaut x, alors on sort de la boucle et la terminaison est claire. Sinon :
  - Si $T[m_k] < x$, on a $b_k = b_{k-1}$ et comme $a_{k-1} \leq b_{k-1}$, on a

$$a_k = m_k + 1 \quad > \quad m_k = (a_{k-1} + b_{k-1})//2 \quad \geq a_{k-1}$$

    donc $\boxed{v_k < v_{k-1}}$.
  - De même, si $T[m_k] > x$, on a $a_k = a_{k-1}$ et $b_k < m_k \leq b_{k-1}$. D'où $\boxed{v_k < v_{k-1}}$.

Finalement, v_k est bien un variant de boucle. Par conséquent l'algorithme termine.

1. En réalité, le cas où $v_k \geq 0$ pour tout k n'arrive jamais. Mais on a besoin de la positivité de v_k pour utiliser le fait que $a_{k-1} \leq b_{k-1}$ plus loin. On montre en fait que (v_k) est un variant de boucle dans un cas qui n'arrive jamais, c'est une sorte de raisonnement par l'absurde.