Outils logiques et algorithmiques – TD 2 – Correction

Exercice 1

1. Cas dégénéré : `s` vide, aucune comparaison nécessaire. Sinon, dans le meilleur cas la boucle interne s'arrête après le premier test, d'où $l_t - l_s$ comparaisons. Exemple : recherche de `"baobab"` dans `"acdefghijkl"`.
2. Pire cas : la boucle interne fait systématiquement l_s comparaisons. D'où au total $l_s(l_t - l_s)$. Exemple : recherche de `"aaaab"` dans `"aaaaaaaaaaaaaaaaaaab"`.
3. La complexité en moyenne dépend (un peu) du nombre de caractères différents. Si on considère 256 caractères possibles, la probabilité de faire au moins $k+1$ tours ($k+1 \leq l_s$) est $\frac{1}{256^k}$. Cette décroissance est très rapide : la complexité moyenne reste de l'ordre de $l_t - l_s$.

Exercice 2

1. Exactement $3n+1$ opérations : $n+1$ tests, n multiplications et n soustractions.
2. Entre $5\lfloor\log(n)\rfloor+6$ et $6\lfloor\log(n)\rfloor+7$ opérations : on a $k = \lfloor\log(n)\rfloor+1$ tours de boucle, avec exactement $k+1$ occurrences du test `n > 0`, exactement k occurrences du test `n%2 == 1`, de la multiplication `a*a` et de la division `n/2`, et au plus k occurrences de la multiplication `r*a`.
3. On prend la borne supérieure pour `power_2`, on cherche donc à résoudre $6\lfloor\log(n)\rfloor + 7 < 3n+1$. C'est bon pour $n = 7$, on a égalité pour $n = 8$, puis à nouveau vérifié strictement pour tout $n \geq 9$.

Exercice 3

1. Dans chacun des deux programmes, la boucle **`for`** `i` est exécutée n fois, la boucle **`for`** `j` $\binom{n}{2} = \frac{n(n-1)}{2}$ fois, et la boucle **`for`** `k` $\binom{n}{3} = \frac{n(n-1)(n-2)}{6}$ fois.
2. Pour le premier : $3\binom{n}{3} = \frac{n(n-1)(n-2)}{2}$ lectures ($\Theta(n^3)$ et $\sim \frac{n^3}{2}$). Pour le deuxième : $n + \binom{n}{2} + \binom{n}{3} = \frac{n(n^2+5)}{6}$ lectures ($\Theta(n^3)$ et $\sim \frac{n^3}{6}$).
3. On commence par trier le tableau (temps $\mathcal{O}(n^2)$ même pour un mauvais algo), puis on remplace la boucle **`for`** `k` par une recherche dichotomique de $-i-j$. Sans l'hypothèse sur l'absence de doublons, il faut des variantes de la recherche dichotomique pour trouver la première et la dernière occurrence et en déduire le nombre d'occurrences de $-i-j$.

Exercice 4

1. Deux boucles imbriquées sur des indices de tableaux : quadratique.
2. Mais la progression des temps mesurés est linéaire !
3. Elle fait en moyenne très peu de tours.

Exercice 5 Dans tous les cas, $n-1$ comparaisons pour une chaîne de taille n. En effet, le `s[i+k]` va couvrir exactement une fois chaque case à partir de la deuxième.

Exercice 6

1. On s'inspire de la recherche dichotomique : on définit un intervalle de recherche allant de l'étage `lo = 0` à l'étage `hi = N` (inclus). On teste depuis l'étage médian `mid`. Si l'œuf se casse on continue la recherche dans la moitié basse (`mid` inclus), s'il reste entier on continue dans la moitié haute (à partir de `mid+1`). On termine lorsque l'intervalle ne couvre plus qu'un étage. En java, en supposant que `omelet(`k`)` renvoie `true` si un œuf lâché de l'étage k casse.

```java
static int findC(int n) {
  int lo = 0;
  int hi = n;
  while (lo < hi) {
    int mid = lo + (hi-lo)/2;
    if (omelet(mid)) hi = mid;
    else             lo = mid+1;
  if (omelet(lo)) return lo;
  else            return -1;
```

2. On cherche d'abord le premier étage puissance de 2 à partir duquel un œuf casse, d'où environ $\log(C)$ tests. Puis il suffit de reprendre l'algorithme précédent avec cette borne.
3. La deuxième est avantageuse si $C < \sqrt{N}$.
4. On commence par tester tous les multiples (arrondis) de $\sqrt{N}$ jusqu'au premier où l'œuf se casse. Puis recherche séquentielle dans l'intervalle entre les deux derniers multiples.

Exercice 7

1. Meilleur cas : 1 case modifiée (ou 0 si tableau vide). Pire cas : toutes les cases sont modifiées.
2. (a) Tous les tableaux ayant les $k-1$ premières cases à `True` et la suivante à `False`, les $n-k$ autres étant arbitraires. Autrement dit : 2^{n-k} tableaux, pour $1 \leq k < n$. Pour $k = n$ on ajoute a ce même cas plus le cas où toutes les cases valent `True`. Au total : $n + \sum_{1 \leq k \leq n} k2^{n-k}$.

   (b) La case d'indice k est touchée dès lors que les k premières cases sont à `True`, les $n-k$ autres étant arbitraires. D'où : 2^{n-k} tableaux, pour $0 \leq k < n$. Au total : $\sum_{0 \leq k < n} 2^{n-k}$.

   (c) Version facile : la deuxième est directement une somme de puissances de 2.

$$\sum_{0 \leq k < n} 2^{n-k} = \sum_{0 < i \leq n} 2^i = 2^{n+1} - 2$$

   (d) Ne reste qu'à diviser par 2^n, le nombre de tableaux différents. En moyenne : 2 cases modifiées ($-\frac{1}{2^{n-1}}$).