Outils logiques et algorithmiques – TD 2 – Complexité

Exercise 1 (Occurrences d'une chaîne) Cette fonction compte le nombre d'occurrences de la séquence `s` dans le texte `t`.

```
static int countOccurrences(String s, String t) {
  int ls = s.length();
  int lt = t.length();
  int c=0;
  mainLoop:
  for (int i=0; i+ls <= lt; i++) {
    for (int j=0; j<ls; j++) {
      if (s.charAt(j) != t.charAt(i+j)) continue mainLoop;
    }
    c++;
  }
  return c;
}
```

1. En fonction des tailles de `s` et `t`, combien cette fonction effectue-t-elle de comparaisons de caractères dans le meilleur cas ? Donner un exemple réalisant cette borne.
2. Même question pour le pire cas.
3. Que dire sur la complexité en moyenne ?

□

Exercise 2 (Exponentiation) Voici deux programmes calculant a^n, pour $n \in \mathbb{N}$.

```
static int power1(int a, int n) {
  int r = 1;
  while (n > 0) {
    r *= a;
    n--;
  }
  return r;
}
```

```
static int power2(int a, int n) {
  int r = 1;
  while (n > 0) {
    if (n % 2 == 1) r *= a;
    a = a*a;
    n = n/2;
  }
  return r;
}
```

1. Quel est le nombre exact d'opérations arithmétiques ou tests effectués par `power1` ?
2. Donner un encadrement du nombre d'opérations arithmétiques ou tests effectuées par `power2`.
3. À partir de quelle valeur de n la deuxième fonction nécessite-t-elle à coup sûr moins d'opérations que la première ?

□

Exercise 3 (Sommes nulles) Les deux programmes suivants cherchent dans un tableau d'entiers l'ensemble des triplets dont la somme vaut zéro.

```
static int threeSum1(int[] tab) {
  int n = tab.length;
  int count = 0;
  for (int i=0; i<n; i++) {
    for (int j=i+1; j<n; j++) {
      for (int k=j+1; k<n; k++) {
        if (t[i] + t[j] + t[k] == 0)
          count++;
      }
    }
  }
  return count;
}
```

```
static int threeSum2(int[] tab) {
  int n = tab.length;
  int count = 0;
  for (int i=0; i<n; i++) {
    int ti = t[i];
    for (int j=i+1; j<n; j++) {
      int tj = t[j];
      for (int k=j+1; k<n; k++) {
        if (ti + tj + t[k] == 0)
          count++;
      }
    }
  }
  return count;
}
```

1. Combien de fois est exécutée chacune des boucles ?
2. Combien chacun de ces deux programmes réalise-t-il de lectures dans le tableau `t` ? Donner la valeur exacte, un ordre de grandeur et un équivalent.
3. Proposer un nouvel algorithme permettant de calculer le même résultat avec un nombre d'accès au tableau $\Theta(n^2 \log(n))$. On pourra supposer que tous les éléments du tableau sont distincts.

□

Exercise 4 (Boucles trompeuses) Voici une fonction mystérieuse, et ses temps d'exécution en micro-secondes mesurés pour des entrées aléatoires de différentes tailles.

```
static int[] elefant(int[] tab) {
  int[] p = new int[tab.length];
  for (int i=1; i<tab.length; i++) {
    int j = i-1;
    while (t[j] >= t[i]) j = p[j];
    p[i] = j;
  }
  return p;
}
```

Taille	Temps
100	27
200	50
400	100
800	180
1600	360

1. En se fiant uniquement au code, quel serait l'ordre de grandeur de la complexité de cette fonction ?
2. Comparer avec les temps mesurés.
3. Que pouvez-vous dire de la boucle **while** ?

□

Exercise 5 (Plus longue répétition) La fonction suivante cherche dans une chaîne la longueur maximale d'une séquence répétant un même caractère.

```
static int longestRepetition(String s) {
  int i = 0, r = 0;
  while (i < s.length()) {
    int k = 1;
    while (i+k < s.length() && s.charAt(i+k) == s.charAt(i)) k++;
    if (k > r) r = k;
    i += k;
  }
  return r;
}
```

Combien cette fonction effectue-t-elle de comparaisons dans le meilleur cas ? Dans le pire cas ? En moyenne ? □

Exercise 6 (Grosse omelette) On se trouve dans un immeuble de N étages, avec une réserve inépuisable d'œufs. On cherche le plus petit étage C à partir duquel un œuf lâché par la fenêtre casse en arrivant au sol. On suppose $0 \leq C \leq N$.

1. Comment trouver C avec $\sim \log(N)$ lancers ?
2. Comment trouver C avec $\sim 2\log(C)$ lancers ?
3. À quelle condition reliant C et N la deuxième méthode devient-elle avantageuse ?

Supposons maintenant que nous n'avons que 2 œufs.

4. Comment trouver C avec $\sim 2\sqrt{N}$ lancers ?

□

Exercise 7 (Compteur binaire) On souhaite énumérer tous les tableaux de n booléens, pour un n arbitraire. Stratégie : chaque tableau correspond à l'écriture binaire d'un nombre $k \in [0, 2^n[$, on les énumère dans l'ordre. On considère que le bit de poids faible est à l'indice 0 du tableau. Pour cela on se donne la fonction java suivante, qui modifie le tableau `t` en le tableau du nombre suivant.

```
static void incr(boolean[] t) {
  int n = t.length();
  int i = 0;
  while (i < n && t[i]) {
    t[i] = false;
    i++;
  }
  if (i < n) t[i] = true;
}
```

1. Combien la fonction `incr` modifie-t-elle de cases du tableau `t` dans le meilleur cas ? dans le pire cas ?
2. On veut calculer la complexité moyenne de `incr`. Pour cela on calcule $S(n)$ la somme des nombres de cases modifiées par `incr` sur tous les tableaux de booléens de taille n.
   (a) Dans combien de tableaux de taille n `incr` modifie-t-elle exactement k cases ? Déduire une expression pour $S(n)$.
   (b) Dans combien de tableaux de taille n `incr` modifie-t-elle la case d'indice k ? Déduire une autre expression pour $S(n)$.
   (c) Résoudre l'une ou l'autre de ces deux expressions. *Les deux sont solubles ! Mais une est beaucoup plus simple...*
   (d) En déduire la complexité moyenne de l'opération `incr`.

□