Feuille d'exercices n°4 - Complexité pire cas

Notions abordées

- notations de Landau
- comparaison des croissances logarithmique, polynomiale, exponentielles...
- complexité pire cas
- nombre de tours de boucle

Notations de Landau

Exercice 1 Quelques exemples

Justifier les propositions suivantes (où les suites sont implicitement indexées par $n \in \mathbb{N}$).

- $1000n^2 \in \mathcal{O}(n^3)$
- $24n^2 + 48n^3 \in \mathcal{O}(n^3)$
- $2^{n+45} \in \mathcal{O}(2^n)$
- $18 + 3n + n^2 \in \Theta(n^2)$
- $50n + 18n^2 \in \Omega(n^3)$
- $\dfrac{n(n+1)(2n+1)}{6}$

Exercice 2 Échelle de croissances asymptotiques

Comparer les comportements asymptotiques des suites ci-dessous implicitement indexées par $n \in \mathbb{N}$. Il s'agit d'identifier quelle suite domine quelle autre, et quelle suite est équivalente à quelle autre. On pourra résumer ces comparaisons en plaçant les suites sur un axe horizontal.

- $6 + 2 \times (-1)^n$
- $\log_2(n)$
- $\ln(n)$
- $6n^4 + 3n^2$
- $3n + 12$
- $\exp(n)$
- 2^n
- $2n$
- $n + \log_2(n)$
- $\log_3(n)$
- $\log_2(n^2)$
- $\log_2(2^n)$
- $\left(\log_2(n)\right)^2$
- $\sqrt{n}$
- $\sqrt{\log_{10}(n)}$
- $n!$
- 3^n
- $n \log_2(n)$
- n^n
- $\dfrac{n^5}{5!}$

Exercice 3 Manipulation des définitions

Soient f, g, h, k quatre suites de $\mathbb{N}$ dans $\mathbb{N}$ (*i.e.* $(f, g, h, k) \in \left(\mathbb{N}^{\mathbb{N}}\right)^4$).
Dire si les implications suivantes sont vraies ou fausses. Dans le premier cas, dire pourquoi, dans le second donner un contre-exemple.

		Vrai	Faux	Éléments de preuve ou contre-exemple
1	$\left.\begin{array}{l} f \in O(h) \\ g \in O(h) \end{array}\right\} \Rightarrow f+g \in O(h)$			
2	$\left.\begin{array}{l} f \in \Omega(h) \\ g \in \Omega(h) \end{array}\right\} \Rightarrow f+g \in \Omega(h)$			
3	$\left.\begin{array}{l} f \in O(h) \\ g \in \Omega(h) \end{array}\right\} \Rightarrow f+g \in \Theta(h)$			
4	$\left.\begin{array}{l} f \in O(h) \\ g \in \Theta(h) \end{array}\right\} \Rightarrow f+g \in \Theta(h)$			
5	$\left.\begin{array}{l} f \in O(h) \\ g \in O(h) \end{array}\right\} \Rightarrow fg \in O(h)$			
6	$\left.\begin{array}{l} f \in O(h) \\ g \in O(k) \end{array}\right\} \Rightarrow fg \in O(hk)$			
7	$\left.\begin{array}{l} \forall k \in \mathbb{N}, f_k \in O(h) \\ F = n \mapsto \sum_{k=0}^{n} f_k(n) \end{array}\right\} \Rightarrow F \in O(h)$			
8	$\left.\begin{array}{l} \forall k \in \mathbb{N}, f_k \in O(h) \\ F = n \mapsto \sum_{k=0}^{n} f_k(k) \end{array}\right\} \Rightarrow F \in O(h)$			
9	$\left.\begin{array}{l} f \in O(h) \\ f \in \Omega(h) \end{array}\right\} \Rightarrow f \in \Theta(h)$			
10	$f = h \Rightarrow f \in \Theta(h)$			
11	$f = 2h \Rightarrow f \in \Omega(h)$			
12	$f = 2h \Rightarrow f \in O(h)$			
13	$\left.\begin{array}{r} f \leqslant g \\ g \in O(h) \end{array}\right\} \Rightarrow f \in O(h)$			
14	$f \in O(h) \Rightarrow f \geqslant h$			
15	$f \in O(h) \Rightarrow f \leqslant h$			
16	$f \in O(h) \Rightarrow h \in \Omega(f)$			
17	$\forall n \in \mathbb{N}^*,\ f_n = g_{n-1} \Rightarrow f \in \Theta(g)$			
18	$\forall n \in \mathbb{N},\ f_n = g_{n+1} \Rightarrow f \in \mathcal{O}(g)$			

Complexité temporelle de fonctions

Exercice 4 Compter les opérations dans des boucles

Question 1 ✍

Pour chacune des fonctions ci-dessous, donner le nombre d'additions effectuées en fonction des valeurs des arguments à l'appel.

```c
void a(int n, int m){
  i = 1;
  j = 1;
  while(i<=n && j<=m){
    i++;
    j++;
  }
}
```

```c
void b(int n, int m){
  i = 1;
  j = 1;
  while(i<=n || j<=m){
    i++;
    j++;
  }
}
```

```c
void c(int n, int m){
  i = 1;
  j = 1;
  while(i <= n){
    if (j <= m)
      j++;
    else
      i++;
  }
}
```

```c
void d(int n, int m){
  i = 1;
  j = 1;
  while(i <= n){
    if (j <= m)
      j++;
    else{
      i++;
      j = 1;
    }
  }
}
```

Exercice 5 Complexités de fonctions déjà vues

Étudier la complexité pire cas des fonctions `aff_triangle_pascal`, `mult_simple` et `mult_rapide` dont le code est rappelé ci-dessous.

```c
void aff_triangle_pascal(int n){
  //hyp : n >= 0
  //affiche les n premières lignes du triangle de Pascal

  int i = 0;
  int fi = 1;
  int j, fj , fimj;

  //fi= i!
  while (i < n){
    j = 0;
    fj = 1;
    fimj = fi;

    //fj= j! et fimj= (i-j)!
    while (j < i){
      printf("%d  ",fi/(fj*fimj));
      fimj = fimj/(i-j);
      j = j + 1;
      fj = fj*j;
    }

    //ici i=j
    printf("%d  ",fi/(fj*fimj));
    printf("\n");
    i = i + 1;
    fi = fi*i;
  }
}
```

```c
int multi_rapide(int a, int b){
  //hyp: a >= 0
  //retourne a*b

  int res = 0;
  int aa = a;
  int bb = b;

  //a*b = aa*bb + res
  while (bb > 0){
    if(bb%2 == 1){
      res = res + aa;
    }
    bb = bb/2;
    aa = aa+aa;
  }

  return res;
}
```

```c
int multi_simple(int a, int b){
  //hyp : b >= 0
  //retourne a fois b

  int c = 1;
  int d = 0;

  //invariant : d = a*(c-1)
  while(c <= b){
    d = d + a;
    c = c + 1;
  }
  //rmq : ici c=b+1 donc d = a*b

  return d;
}
```

Complexité temporelle pire cas

Exercice 6 Complexité pire cas

Question 1
Si la complexité pire des cas d'un algorithme est en $\mathcal{O}(n^2)$, est-il possible qu'il soit en $\mathcal{O}(n)$ sur toutes entrées possibles ?

Question 2
Si la complexité pire des cas d'un algorithme est en $\Theta(n^2)$, est-il possible qu'il soit en $\mathcal{O}(n)$ sur certaines entrées ?

Question 3 ✍
Si la complexité pire des cas d'un algorithme est en $\Theta(n^2)$, est-il possible qu'il soit en $\mathcal{O}(n)$ sur toutes les entrées ?

Exercice 7 Recherche de minimum

Dans cet exercice on s'intéresse au problème de trouver le minimum d'un tableau d'entiers, puis au comptage du nombre d'occurrence du minimum.

Question 1 ✍
Formaliser ces deux problèmes. S'agit-il de problèmes de décision ? De problèmes d'optimisation ?

Question 2 ✍
Proposer une fonction `min_tab` qui retourne le minimum d'un tableau d'entiers.

Question 3 ✍
Dans la fonction `min_tab` appelée sur un tableau de n entiers, combien d'opérations de comparaison (resp. d'affectation) sont réalisées dans le pire cas. Donner un exemple de tel pire cas.

Question 4 ✍
Donner la complexité de la fonction `min_tab`. Par quel adjectif qualifier une telle complexité ?

Question 5 ✍
Proposer une fonction `nb_occ_min` qui retourne le nombre d'occurrences du minimum d'un tableau d'entiers.

Question 6 ✍
Donner la complexité de la fonction `nb_occ_min`. Par quel adjectif qualifier une telle complexité ?

Question 7 ✍
Si on suppose que les valeurs du tableau d'entiers en entrée sont forcément comprises dans l'intervalle $[a..b]$, peut-on modifier les fonctions proposer de manière à améliorer leur complexité pire cas ?

Gagner en efficacité

Exercice 8 Suite de Fibonacci

On cherche à tester si un entier n apparaît dans les n premiers termes de la suite de Fibonacci. On rappelle que cette suite, que l'on notera f, est définie par $f_0 = 0$, $f_1 = 1$, $\forall n \in \mathbb{N}$, $f_{n+2} = f_{n+1} + f_n$.

Question 1 ✍
Formaliser ces deux problèmes. S'agit-il de problèmes de décision ? De problèmes d'optimisation ?

Question 2 ✍
Quelles sont les complexités pire cas des fonctions ci-dessous en fonction de n.

Question 3 ✍
En observant que f est strictement croissante à partir du rang 2, on peut sortir de la fonction dès qu'on a un terme $> n$. Cette amélioration change-t-elle l'ordre de grandeur de la complexité pire cas de `est_terme_Fibo` ? Change-t-elle le nombre exact d'opérations effectuées dans le pire cas ?

Question 4 ✍
Proposer une version de `est_terme_Fibo` dont la complexité pire cas est significativement améliorée.

Exercice 9 Valeur en base 3

Question 1 ✍
Donner la définition mathématique de la fonction qui donne la valeur d'un nombre écrit en base 3.

Question 2 ✍
Proposer un code en C qui réalise cette fonction. Compter le nombre d'additions et de multiplications effectuées lors d'un appel de cette fonction. Modifier ce code pour réduire au maximum ce nombre.

Question 3 ✍
Pour chacune des fonctions proposées 7 dénombrer les additions et les multiplications effectuées afin d'en déduire le comportement asymptotique de leur complexité temporelle.

Exercice 10 Palindrome

Question 1 ✍
Que penser du code suivant en terme d'efficacité. On suppose que la fonction `longueur` calcule la longueur d'une chaîne de caractères en la parcourant jusqu'à trouver le caractère `'\0'`.

```
bool est_palindrome(char* a, int lg){
  int i = 0;
  while (i < longueur(a)/2){
    if (a[i] != a[longueur(a)-1-i])
      return false;
  }
  return true;
}
```

```c
long puiss_naif(int x, int p){
//hyp : p >= 0
//retourne x^p

  long res = 1;
  int i = 0;

  // res = x^i
  while (i < p){
    res = res * x;
    i = i + 1 ;
  }

  return res;
}
```

```c
long puiss_rapide(int x, int p){
//hyp: p >= 0
//retourne x^p

  long res = 1;
  int q = p;
  int y = x;

  //x^p = y^q * res
  while (q > 0){
    if(q%2 == 1){
      res = res * y;
    }
    q = q/2;
    y = y*y;
  }

  return res;
}
```

```c
long val3_naif(int* tab, int lg){
//hyp : tab est de longueur lg
//et est_valide(tab,lg)
//retourne la vlr codée en base 3
//dans tab ac les unités à gauche
  long res = 0;
  int i = 0;
  //inv :
  while(i<lg){
    res = res + tab[i] *
    ↪  puiss_naif(3,i);
    i++;
  }
  return res;
```

```c
long val3_rapide(int* tab, int lg){
//hyp : tab est de longueur lg
//et est_valide(tab,lg)
//retourne la vlr codée en base 3
//dans tab ac les unités à gauche
  long res = 0;
  int i = 0;
  while(i<lg){
    res = res + tab[i] *
    ↪  puiss_rapide(3,i);
    i++;
  }
  return res;
```

```c
long val3_malin_g(int* tab, int lg){
//hyp : tab est de longueur lg
//et est_valide(tab,lg)
//retourne la vlr codée en base 3
//dans tab ac les unités à gauche
  long res = 0;
  int i = 0;
  long p = 1;
  //p=3^i
  while(i<lg){
    res = res + tab[i] * p;
    i++;
    p = 3*p;
  }
  return res;
```

```c
long val3_malin_d(int* tab, int lg){
//hyp : tab est de longueur lg
//et est_valide(tab,lg)
//retourne la val codée en base 3
//dans tab ac les unités à gauche
  long res = 0;
  int i = lg-1;
  while(i >= 0){
    res = 3 *res + tab[i];
    i--;
  }
  return res;
```