Corrigé des exercices

Exercice1.1

L'algorithme le plus simple que nous dénotons A1 découle directement de la définition d'un nombre premier. Rappelons qu'un nombre premier n est un nombre entier qui n'est divisible que par 1 et par lui-même. L'algorithme va donc consister en une boucle dans laquelle on va tester si le nombre n est divisible par 2, 3, …, n-1.

```
Algorithme A1 ;
début
        premier := vrai ;
        i := 2 ;
        tant que (i <= n-1) et premier  faire
                si (n mod i = 0) alors premier := faux sinon i := i+1 ;
fin.
```

Le pire cas qui nécessite le plus long temps, correspond au cas où n est premier car c'est dans ce cas que la boucle s'exécute avec un nombre maximum d'itérations. Dans ce cas ce nombre est égal à n-2. La complexité est donc en *O(n)*.

Nous savons que pour améliorer l'algorithme, il est judicieux d'arrêter la boucle à n/2 car si n est divisible par 2, il est aussi divisible par n/2 et s'il est divisible par 3, il est aussi divisible par n/3. De manière générale, si n est divisible par i pour i = 1 … $\lfloor n/2 \rfloor$ où $\lfloor n/2 \rfloor$ dénote la partie entière de n/2, il est aussi divisible par n/i. Il n'est donc pas nécessaire de vérifier qu'il est divisible par un nombre supérieur à n/2. Le deuxième algorithme est donc :

```
Algorithme A2 ;
début
        premier := vrai ;
        i := 2 ;
        tant que ((i <= ⌊n/2⌋) et premier)  faire
                si (n mod i = 0) alors premier := faux sinon i := i+1 ;
fin
```

Le cas le plus défavorable qui nécessite le plus long temps correspond toujours au cas où n est premier et dans ce cas le nombre d'itérations est égal à $\lfloor n/2 \rfloor$ -1. La complexité est donc en *O(n)*.

Une autre amélioration possible consiste à tester si n est impair et dans ce cas dans la boucle, il ne faut tester la divisibilité de *n* que par les nombres impairs. L'algorithme A3 est donc comme suit :

```
Algorithme A3 ;
début
        premier := vrai ;
        si ((n <> 2) et (n mod 2 = 0)) alors premier := faux
        sinon si (n <> 2) alors
                début
                   i :=3 ;
                   tant que ((i <= n-2) et premier) faire
                         si (n mod i = 0) alors premier := faux sinon i := i+2 ;
                fin
fin
```

Le pire cas correspond au cas où *n* est premier et dans ce cas le nombre maximum d'itérations de la boucle est égal à $\lfloor n/2 \rfloor - 2$, la complexité est en *O(n)*.

L'algorithme A4 peut être obtenu en hybridant A2 et A3 et on obtient :

```
Algorithme A4 ;
début
        premier := vrai ;
        si (n <> 2) et (n mod 2 = 0) alors premier := faux
        sinon si ( n <> 2) alors
                début
                   i :=3 ;
                   tant que (i <= [n/2]) et premier  faire
                         si (n mod i = 0) alors premier := faux sinon i := i+2 ;
                fin
fin.
```

Le nombre d'itérations de la boucle pour un nombre premier est égal à la moitié du nombre d'itérations de A3, il est égal à $\lfloor n/4 \rfloor - 1$. La complexité est donc *O(n)*.

Une bonne amélioration de l'algorithme serait d'arrêter la boucle non pas à $\lfloor n/2 \rfloor$ mais à $\sqrt{n}$ car en effet si *n* est divisible par *i*, il est aussi divisible par *n/i*. Et donc il serait judicieux de ne pas répéter le test de la divisibilité au-delà de $i = n/i$ et dans ce cas $n = i^2$ et $i = \sqrt{n}$. L'algorithme A5 s'écrit donc comme suit :

```
Algorithme A5 ;
début   premier := vrai ;
        i := 2 ;
        tant que ((i <=⌊√n⌋) et premier) faire
                si (n mod i = 0) alors premier := faux sinon i := i+1 ;
fin
```

Le nombre maximum d'itérations est égal à $\lfloor \sqrt{n} \rfloor - 1$, la complexité est en $O(\sqrt{n})$. Enfin, on peut concevoir un algorithme en hybridant A5 et A3, on obtient l'algorithme A6 suivant :

```
Algorithme A6 ;
début   premier = vrai ;
        si (n <> 2) et (n mod 2 = 0) alors premier := faux
        sinon si ( n <> 2) alors
          début   i :=3 ;
                  tant que ((i <= ⌊√n⌋) et premier)  faire
                        si (n mod i = 0) alors premier := faux sinon i := i+2 ;
        fin
fin
```

Le nombre maximum d'itérations de la boucle est égal à $\frac{\lfloor \sqrt{n} \rfloor}{2} - 1$. La complexité est donc en $O(\sqrt{n})$.

Récapitulatif :

Algorithme	Nombre maximum d'itérations en fonction de n	Complexité théorique	Nombre réel d'itérations pour n = 990181
A1	n-2	O(n)	990179
A2	[n/2] -1	O(n)	495089
A3	[n/2] -1	O(n)	495089
A4	[n/4] -2	O(n)	247563
A5	$[\sqrt{n})]$-1	$O(\sqrt{n})$	994
A6	$[\sqrt{n}/2]$ -2	$O(\sqrt{n})$	495

Nous remarquons que lorsque l'on change d'ordre de complexité, le temps réel change de grandeur : un nombre de 6 chiffres pour les algorithmes A1 à A4 et un nombre de 3 chiffres pour A5 et A6.

Pour conclure, nous faisons remarquer que de simples améliorations au niveau de l'algorithme initial qui est le plus basique, nous a conduit à écrire un code très rapide. En effectuant les différentes améliorations, nous avons fait chuter le nombre d'itérations de 990179 à 495.

Exercice1.2

1) 1 heure = 3600s = $3.6\ 10^3$s
1 jour = 86400s = $8.64\ 10^4$s
1 semaine = 604800s ≈ $6.05\ 10^5$s
1 mois= 2 592 000s ≈ $2.59\ 10^6$s
1 année =31 536 000 ≈ $3.15\ 10^7$s
1 siècle =3 153 600 000≈$3.15\ 10^9$s
1 millénaire = 31 536 000 000 ≈ $3.15\ 10^{10}$s

2) Le temps nécessaire au traitement des tailles du problème pour n=10, n=100 et n=1000 pour une unité de temps égale à une milliseconde est montré dans le tableau suivant :

Algorithme	complexité	temps		
		n=10	n= 100	n=1000
A0	Ln n	0,002s	0,005s	0,009s
A1	$\sqrt{n}$	0,003s	0,01s	0,031s
A2	n	0,01s	0,1s	1s
A3	n^2	0,1s	10s	16mn40s
A4	n^3	1s	16 mn 40s	11j13h46mn40s
A5	n^4	10s	1j3h46mn40s	31 ans8 mois 15j 19h 3mn 28s
A6	2^n	1,02s	$3.2\ 10^{16}$ millénaires	$3.2\ 10^{286}$millénaires

3) Le temps nécessaire au traitement des tailles de problème n=10, n =100 et n=1000 pour une unité de temps égale à une microseconde est montré dans le tableau suivant :

Algorithme	complexité	temps		
		n=10	n= 100	n=1000
A0	Ln n	$2,3*10^{-6}$s	$4,6*10^{-6}$ s	$9,9*10^{-6}$s
A1	$\sqrt{n}$	$3,1*10^{-6}$s	10^{-5} s	$3,1*10^{-5}$s
A2	n	10^{-5}s	10^{-4} s	10^{-3}s
A3	n^2	10^{-4}s	0,01s	1s
A4	n^3	10^{-3}s	1s	16mn40s
A5	n^4	10^{-2}s	1mn40s	11j13h46mn40s
A6	2^n	10^{-3}s	$3.2\ 10^{13}$ millénaires	$3.2\ 10^{283}$ millénaires

4) Nous concluons que l'augmentation de la performance de la machine de calcul apporte les effets suivants :

   a. Améliore le temps de calcul pour des complexités polynomiales.

   b. n'atténue en rien les valeurs prohibitives des complexités exponentielles des grandes tailles de problème et ne peut donc pas constituer une solution pour contourner le problème de l'explosion combinatoire.

   c. Pour les petites tailles, la fonction exponentielle est plus intéressante que certaines fonctions polynomiales (n = 10, A6 est plus rapide que A5).