EA 4 : TD 1 Opérations arithmétiques

Exercice 1

Le vieil ordi : $50 \cdot 10^6 \cdot \log_{10}(10^6) / 10^6 = 50 \cdot \log_{10}(10^6) = 300$ secondes

$50 \cdot 10^7 \cdot \log_{10}(10^7) / 10^6 = 3500$ secondes

Nouvel ordi : $(10^6)^2 / 10^8 = 10\ 000$ secondes

$(10^7)^2 / 10^8 = 10^6$ secondes

$\Rightarrow$ L'algo est très important dans le tps de calcul.

Exercice 2

1)

res	m	n
0	13	14
0	26	7
26	52	3
78	104	1
$\boxed{182}$	208	0

$= 13 \times 14$

4) $\times 2 \rightarrow$ décale vers gauche

$/ 2 \rightarrow$ décale vers droite

nb impair si le dernier bit est 1.

$13 = 1101$

$14 = 1110$

res	m	n
0	1101	1110
0	11010	111
11010	110100	11
1001110	1101000	1
$\boxed{10110110}$		0

$= (182)_{10}$

3) Au pire $\log_2(n)$ divisions et multiplications

$\log_2(n)/2$ additions $\rightarrow$ une fois sur 2, n sera impair (impair $/2$ = pair, pair $/2$ = impair).

5) C'est plus rapide et plus facile en base 2 qu'en base 10 $\rightarrow$ les divisions et multiplications prennent moins de temps.

2) Preuve de correction $\rightarrow$ y'a une boucle $\Rightarrow$ chercher un invariant : $N \times \Pi + res$

Au début : $N.\Pi + res = N.\Pi$

À la fin : $N = 0$ mais $N\Pi + res$ constant donc $res = N\Pi$

$\hookrightarrow$ il faut juste montrer l'invariance du truc.

avant :	$n = 2k$	$m = l$	res	$n = 2k+1$	$m = l$	res
après :	$n = k$	$m = 2l$	res	$n = k$	$m = 2l$	$res + l$

$\hookrightarrow nm + res = 2kl + res$

$nm + res = k.2l + res + l = (2k+1)l + res$

Exercice 3

```
res = 1

while (n != 1)
    if (n % 2 == 1) res *= m
    m *= m
    n //= 2
return res
```

Invariant : $res \cdot m^N$

Si $N = 2k$

À $t+1$: $m^N \cdot res = (m^2)^k \cdot res = m^{2k} \cdot res = m^n \cdot res$

À t : $m^N \cdot res = m^n \cdot res$

Si $N = 2k+1$

À $t+1$: $m^N \cdot res = (m^2)^k \cdot m = m^{2k+1}$

Exponentiation	Multiplication
$n \times m$	$n + m$
n ^ m	$n \times m$
neutre : 1	neutre = 0

Exercice 4

1)
```
calcule_polynome(int[] tab, int x){
    int res = 0;
    for (int i = 0; i < tab.length; i++){
        res += x^i x tab[i];
    }
    return res;
}
```

2) $tab = \{1, 1, 1, 1, 1\}$

i	res
0	$+1 \times 3^0 = 1$
1	$+1 \times 3^1 = 4$
2	$+1 \times 3^2 = 13$
3	$+1 \times 3^3 = 40$
4	$+1 \times 3^4 = 121$

On fait plusieurs fois les mêmes opérations pour pouvoir calculer les puissances ($3^3 = 3^2 \times 3$)

3) $(n+1)!$ multiplications pour calculer les puissances × le coeff.
n additions pour ajouter tous les résultats.

4) On peut utiliser l'exponentiation binaire pour calculer plus rapidement les puissances successives du polynôme.

5) Tout polynôme $P(x)$ s'écrit $P(x) = a_0 x^0 + a_1 x^1 + \dots + a_n x^n$

$$= a_0 + x(a_1 + \dots + a_n x^n)$$

$$= a_0 + x(a_1 + x(a_2 + \dots + a_n x^n))$$

$$= a_0 + x(a_1 + x(a_2 + \dots + a_{n-2} + x(a_{n-1} + a_n x)))$$

Dans le programme, on part de la fin du tableau donc, dans l'expression, de la parenthèse la plus intérieure.
À chaque tour de boucle i, on multiplie le résultat précédent par x, et on ajoute le coefficient a_{n-i}.

On s'arrête quand on a ajouté tous les coefficients.
On a alors calculé la valeur du polynôme selon la formule factorisée.

On effectue n additions et n multiplications avec ce programme.

Exercice 5

1)
```
bool premier_naif(int n):
    for i in range(2, n-1):
        if (n % i == 0): return false
    return true
```

→ Pour chaque entier avant n, on teste s'il divise n.
→ complexité : n
→ on peut l'améliorer en enlevant les multiples de chaque nb testé ⇒ on ne teste que les nb premiers.

2)

i	k	tab
		{F F T T T T T T T T T}
2	4, 6, 8, 10	{F F T T F T F T F T F}
3	6, 9	{F F T T F T F T F F F}
4	∅	tab[4] = false
5	10	{F F T T F F F T F F F}
6	∅	tab[6] = false
7	∅	7 × 2 = 14 > 10
8, 9, 10	∅	tab[8] = tab[9] = tab[10] = false

3) C'est le tableau des nb premiers → on a fait le crible d'eratosthène