# Architecture des ordinateurs : TD1

Université de Tours

Département informatique de Blois

*Représentation de l'information & arithmétique des ordinateurs*

*
* *

## Problème 1

1. Donner la représentation des nombres suivants en complément à 2 selon un nombre de bits $k \in 4\mathbb{N}$ (multiple de 4) que l'on précisera.

   $\langle 42 \rangle_7 = 4 \times 7^1 + 2 \times 7^0 = \langle 30 \rangle_{10} = 16 + 8 + 8 + 2\, \langle 0001\ 1110 \rangle_2$

2. $\langle 101010 \rangle_8$

   $\langle 101010 \rangle_8 = \langle (1)(0)(1)(0)(1)(0) \rangle_8 = \langle 001\ 000\ 001\ 000\ 001\ 000 \rangle_2$

3. $\langle -2018 \rangle_{10}$

   $\langle 2018 \rangle_{10} = \langle 0111\ 1110\ 0010 \rangle_2$. Dès lors :

   $\langle -2018 \rangle_{10} = \langle 1111\ 1110\ 0010 \rangle_{b2}$

   $\langle -2018 \rangle_{10} = \langle 1000\ 0001\ 1101 \rangle_{1c}$

   $\langle -2018 \rangle_{10} = \langle 1000\ 0001\ 1110 \rangle_{2c}$

4. $\langle -CA7 \rangle_{16}$

   Comme $\log_2(16) = 4$, on sait qu'un symbole hexadécimal représente 4 bits.

   $\langle CA7 \rangle_{16} = \langle 1100\ 1010\ 0111 \rangle_2$. Dès lors :

   $\langle -CA7 \rangle_{10} = \langle 1000\ 1100\ 1010\ 0111 \rangle_{b2}$

   $\langle -CA7 \rangle_{10} = \langle 1111\ 0011\ 0101\ 1000 \rangle_{1c}$

   $\langle -CA7 \rangle_{10} = \langle 1111\ 0011\ 0101\ 1001 \rangle_{2c}$

5. Donner la représentation IEEE 754 en base 2 sous 32 bits des nombres en base décimale et la représentation décimale des nombres sous représentation IEEE 754 :

   (a) $\langle -2^{120} \rangle_{10}$

   $\langle 2^{120} \rangle_{10} = \left\langle 1\ \underbrace{0000\ 0000\ ...}_{120\ \text{fois}} \right\rangle_2$, soit $1 \times 2^{120}$

   Dès lors :

   - Bit de signe $s = 1$,
   - Mantisse $M = 0000\ 0000\ 0000\ 0000\ 0000\ 000$

- Exposant $E = 120 + 127 = 247 = \langle 1111\ 0111 \rangle_2$

| | | |
|---|---|---|
| 1 | 1111 0111 | 0000 0000 0000 0000 0000 000 |

(b) $\langle 13,658203125 \rangle_{10}$

On représente le nombre selon le standard IEEE 754 - `float` soit $k = 32$.

$13 = \langle 1101 \rangle_2$.

On applique l'algorithme de calcul de la partie flottante avec $0,658203125$ :

| $0,658203125$ | | |
|---|---|---|
| $0,658203125 \times 2$ | 1.31640625 | 1 |
| $0,31640625 \times 2$ | 0.6328125 | 0 |
| $0,6328125 \times 2$ | 1.265625 | 1 |
| $0,265625 \times 2$ | 0.53125 | 0 |
| $0,53125 \times 2$ | 1.0625 | 1 |
| $0,0625 \times 2$ | 0,125 | 0 |
| $0,125 \times 2$ | 0,25 | 0 |
| $0,25 \times 2$ | 0,5 | 0 |
| $0,5 \times 2$ | 1 | 1 |
| 0 | 0 | 0 |

Dès lors $13,658203125 = \langle \langle 1101 \rangle_2 , \langle 1010\ 1000\ 1 \rangle_2 \rangle$

On décale de 3 rangs vers la gauche. Alors $\langle 1,1011\ 0101\ 0001 \rangle \times 2^3$, on en déduit que :

- Le bit de signe $s = 0$,
- La mantisse $m = 1011\ 0101\ 0001$ que l'on complète pour obtenir $M$ sur 23 bits, soit $M = 1011\ 0101\ 0001\ 0000\ 0000\ 000$
- L'exposant $e = 3$. Dès lors $E = e + \varepsilon = 130 = \langle 1000\ 0010 \rangle_2$

Dès lors $13,658203125$ est représenté tel que :

| | | |
|---|---|---|
| 0 | 1000 0010 | 1011 0101 0001 0000 0000 000 |

(c) $\langle 263,3 \rangle_{10}$

On représente le nombre selon le standard IEEE 754 - `float` soit $k = 32$.

$263 = \langle 100000111 \rangle_2$.

On applique l'algorithme de calcul de $0,3$ :

*Récurrence des termes*

| $0,3$ | | |
|---|---|---|
| $0,3 \times 2$ | 0,6 | 0 |
| $0,6 \times 2$ | 1,2 | 1 |
| $0,2 \times 2$ | 0,4 | 0 |
| $0,4 \times 2$ | 0.8 | 0 |
| $0,8 \times 2$ | 1.6 | 1 |
| $0,6 \times 2$ | 1,2 | 1 |
| $0,2 \times 2$ | 0,4 | 0 |
| $\vdots$ | $\vdots$ | $\vdots$ |

On observe une récurrence dans le développement binaire de $0,3 = \langle 0\ 1001\ 1001 \cdots \rangle_2$

Dès lors, il semblerait que : $0,3 = \sum_{i=0}^{\infty} \left(\frac{1}{2^{2+4n}} + \frac{1}{2^{5+4n}}\right)$.
$= \sum_{i=0}^{\infty} \frac{1}{2^{2+4n}} \left(1 + \frac{1}{2^3}\right) = \frac{9}{32} \sum_{i=0}^{\infty} \frac{1}{2^{4n}}$.

On calcule la série $\frac{9}{32} \sum_{i=0}^{\infty} \left(\frac{1}{2^4}\right)^n = \frac{9}{32} \times \frac{1}{1-\frac{1}{2^4}}$
$= \frac{9}{32} \times \frac{16}{15}$
$= \frac{9}{30} = 0,3$

Dès lors $263,3 \approx \langle \langle 1\ 0000\ 0111 \rangle_2 , \langle 0\ 1001\ 1001\ 1001\ 10 \rangle \rangle$

On code alors $\langle 1,0000\ 0111\ 0100\ 1100\ 1100\ 110 \rangle \times 2^8$, on en déduit que :

- Le bit de signe $s = 0$,
- La mantisse $M = 0000\ 0111\ 0100\ 1100\ 1100\ 110$.
- L'exposant $e = 8$. Dès lors $E = e + \varepsilon = 135 = \langle 1000\ 0111 \rangle_2$

Dès lors $263,3$ est représenté tel que :

| | | |
|---|---|---|
| 0 | 1000 0111 | 0000 0111 0100 1100 1100 110 |

(d) $\langle 0\ 10000100\ 01010000000000000000000 \rangle_{I3E}$

$\langle 0\ 10000100\ 01010000000000000000000 \rangle_{I3E} = \left(1 + \frac{1}{4} + \frac{1}{16}\right) \times 2^{132-127}$
$= \left(1 + \frac{1}{4} + \frac{1}{16}\right) \times 2^5 = 2^5 + 2^3 + 2 = \langle 42 \rangle_{10}$
$= 2^5 + 2^3 + 2 = \langle 42 \rangle_{10}$

## Problème 2

On considère le programme suivant permettant le calcul d'une suite $(u_n)_{n \in \mathbb{N}}$ :

```
float u_n(int n) {
   return n == 0 ? (float) 1.0 / 3 : 4 * u_n(n-1) - 1;
}
```

1. Modélisation mathématique :

(a) Démontrer que : $\forall n \in \mathbb{N}, u_n = \frac{1}{3}$.

La suite modélisée par le programme est une suite récurrente $(u_n)_{n \in \mathbb{N}}$ de la forme :

$$\begin{cases} u_0 & = \frac{1}{3} \\ u_{n+1} & = 4u_n - 1 \end{cases}$$

On veut montrer que la propriété $P(n) : u_n = \frac{1}{3}$ est vraie pour tout $n \in \mathbb{N}$.

- Initialisation (pour $n = 0$)
  Vraie par définition de la suite. $P(0)$ est vraie.
- Hérédité
  On suppose que $\exists n \in \mathbb{N}$ telle que $P(n)$ est vraie. On veut montrer que $P(n) \Rightarrow P(n+1)$.
  $u_{n+1} = 4u_n - 1$
  $= 4 \times \frac{1}{3} - 1$
  $= \frac{4-3}{3} = \frac{1}{3}$

$P(n+1)$ est vraie.

- Conclusion
  La propriété $P$ est initialisée pour $n=0$ et est héréditaire. Dès lors, $\forall n \in \mathbb{N}, u_n = \frac{1}{3}$.

(b) Démontrer que le nombre $\frac{1}{3}$ ne peut pas être représenté de manière exacte à l'aide de la norme IEEE 754.

On utilise l'algorithme de représentation des nombres flottants en binaire pour $n$ indéterminé. On précise que $b=2, x=\frac{1}{3}$.

$$\text{Rec}\left\{\begin{array}{|c|c|c|}\hline \multicolumn{3}{|c|}{1/3} \\ \hline \frac{1}{3}\times 2 & \frac{2}{3} & 0 \\ \frac{2}{3}\times 2 & \frac{4}{3} & 1 \\ \frac{1}{3}\times 2 & \frac{2}{3} & 0 \\ \vdots & & \\ \hline \end{array}\right.$$

On constate une boucle infinie au sein de l'algorithme. On voit que $\frac{1}{3} = \langle 0101\ 0101\ 0101\ ...\rangle_2$

***Démonstration***

$$\begin{aligned}\sum_{n=0}^{\infty} \frac{1}{2^{2+2n}} &= \frac{1}{4}\sum_{n=0}^{\infty}\left(\frac{1}{4}\right)^n \\ &= \frac{1}{4}\times\frac{1}{1-\frac{1}{4}} \\ &= \frac{1}{4}\times\frac{4}{3} = \frac{1}{3}\end{aligned}$$

2. Donner la représentation IEEE 754 de $\frac{1}{3}$ au sein du programme.

Au sein du programme, $\frac{1}{3}$ est codé à l'aide d'un `float`, on utilise 32 bits, dont 1 pour le signe $s$, 8 pour l'exposant $E$ et 23 pour la mantisse $M$.

On sait que $\frac{1}{3} = (\langle 0\rangle_2, \langle 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ ...\rangle_2)$

- On décale la virgule 2 rangs vers la droite : $e=-2$. Dès lors $E = e+\varepsilon = -2+127 = 125 = \langle 0111\ 1101\rangle_2$,
- La mantisse $M = \langle 0101\ 0101\ 0101\ 0101\ 0101\ 010\rangle_2$,
- Le signe $s=0$.

Dès lors :

| | | |
|---|---|---|
| 0 | 0111 1101 | 0101 0101 0101 0101 0101 010 |

3. L'appel `u_n(42)` retourne la valeur $1.9215359 \times 10^{17}$.

(a) Expliquer brièvement la différence entre le résultat théorique et celui donné par le programme. Selon vous, et sans le démontrer, quelle type de croissance suit cette erreur (logarithmique, linéaire, quadratique, exponentielle, autre) ?

On a vu que la représentation de $\frac{1}{3}$ au sein du programme est incomplète. En particulier, il faudrait une infinité de bits pour modéliser ce nombre. Ainsi, au sein du programme, on commet une erreur $\delta > 0$ d'approximation de la valeur. Cette erreur est positive et comme le coefficient multiplicateur de l'erreur est supérieur à 1 (Ici 4), alors l'erreur ne sera pas bornée et va diverger.

On constate aisément qu'elle est de nature exponentielle. (Voir question suivante).

(b) Sur la base d'une erreur d'approximation $\delta > 0$, déterminer l'expression de l'erreur $\Delta_n$ commise par la méthode `u_n()` au rang $n$.

On a $\Delta_0 = \delta$,

$\Delta_1 = 4\Delta_0 = 4\delta$,

$\Delta_2 = 4\Delta_1 = 4^2\Delta_0 = 4^2\delta$

...

Par récurrence, on obtient : $\Delta_n = 4^n\delta$.

(c) Calculer l'erreur $\delta_{\max}$ puis appliquer le calculer de l'erreur $\Delta_{42}$ commise par la suite au rang $n = 42$ sur la base d'une erreur $\delta = \delta_{\max}$. Le résultat vous semble t-il cohérent par rapport au résultat rendu par l'ordinateur ? Argumentez votre réponse.

On peut chercher également à calculer $\delta$, que l'on peut majorer par $\delta_{\max}$.

On sait que l'erreur $\delta_{\max} = 2^{e-|M|-1} = 2^{-2-23-1} = 2^{-26}$

Comme on sait que $\delta < \delta_{\max}$, on a alors au rang $n = 42$, une erreur telle que : $\Delta_{42} < 4^{42} \times 2^{-56} \approx 2,9 \times 10^{17}$.

Ce résultat correspond bien à l'ordre de grandeur ($10^{17}$) de la sortie programme ce qui confirme nos calculs.

## Problème 3

Soit $(n, k) \in \mathbb{N}^2$ avec $n \geq k$. On cherche ici à calculer les coefficients binomiaux $C_n^k$ (ou $\binom{n}{k}$) de la formule du binôme de Newton. On rappelle que :

$$C_n^k = \frac{n!}{k!(n-k)!}$$

1. On considère le programme C donné en annexe.

   Que pouvez-dire de cette méthode de calcul. Argumentez vos propos à l'aide de la sortie programme.

Les deux dernières lignes de la sortie écran de l'annexe montre que le programme est erroné.

En effet, d'après la relation du triangle de Pascal : $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$.

Or, pour $C_{13}^1 = 4 \neq C_{12}^0 + C_{12}^1 = 1 + 12 = 13$.

Cette erreur s'explique par la croissance de la factorielle, extrêmement rapide et dont le résultat qui, même pour de petites valeurs d'appel, ne peut être contenu dans un `unsigned int`, à la capaicité max de $2^{32} - 1$.

2. Démontrer que : $\ln\left(C_n^k\right) = \sum_{i=1}^{k} \ln(n+1-i) - \ln(i)$.

$$\begin{aligned}
\ln\left(C_n^k\right) &= \ln\left(\tfrac{n!}{k!(n-k)!}\right) \\
&= \ln(n!) - \ln(k!(n-k)!) \quad \textit{Car } \ln\left(\tfrac{a}{b}\right) = \ln(a) - \ln(b) \\
&= \ln(n!) - [\ln(k!) + \ln((n-k)!)] \quad \textit{Car } \ln(a \times b) = \ln(a) + \ln(b) \\
&= \sum_{i=1}^{n} \ln(i) - \left[\sum_{i=1}^{k} \ln(i) + \sum_{i=1}^{n-k} \ln(i)\right] \quad \textit{Propriété précédente et } n! = \prod_{i=1}^{n} i
\end{aligned}$$

$$= \sum_{i=n-k+1}^{n} \ln(i) - \left[\sum_{i=1}^{k} \ln(i)\right] \quad \textit{Télescopage sur la somme 1 et la somme 3}$$

$$= \sum_{i=1}^{k} \ln(n+1-i) - \left[\sum_{i=1}^{k} \ln(i)\right] \quad \textit{Changement d'indice}$$

$$= \sum_{i=1}^{k} \ln(n+1-i) - \ln(i)$$

3. La propriété utilisée à la question 2. est appellée *réduction logarithmique.*

   (a) En ce basant sur cette propriété, proposer un progamme `double binomial(int n, int k)` en C ou en Java permettant le calcul efficace des coefficients binomiaux. On précise que l'on donne accès aux fonctions de la bibliothèque `math`, `log(x)` et `exp(x)`, retournant respectivement les valeurs $\ln(x)$ et $e^x$.

```
double binomial(int n, int k) {
    double bino = 0.0;
    for(int i = 1 ; i <= k; i++) {
      bino += log(n - 1 - i) - log(i);
     }
    return exp(bino);
}
```

   (b) Donner la notation IEEE 754 du nombre maximal $N$ pouvant être représenté avec un `double`. On précise que $N \approx 1,79 \times 10^{308}$.

   Déterminer à l'aide des valeurs données par l'annexe la valeur maximale $n$ pour laquelle le programme du 3.a peut-être appellé sans comettre d'erreur.

   Un double peut contenir jusqu'à $2^{1024} - 1$. Dès lors, on cherche

$$e^{\ln\left(C_n^k\right)} \leq 2^{1024} - 1 \Leftrightarrow e^{\ln\left(C_n^k\right)} < 2^{1024}$$
$$\Leftrightarrow \ln\left(C_n^k\right) < \ln(2^{1024})$$
$$\Leftrightarrow \ln\left(C_n^k\right) < 709.78$$

   On regarde dans le tableau donné en annexe. Le programme est valable jusqu'à $n = 1029$.

## Problème 4

Le but de cet exercice est de permettre à l'ordinateur de calculer la racine d'un nombre de manière efficace, rapide tout en limitant l'erreur d'approximation commise.
Pour se faire, on peut utiliser l'algorithme de Newton.

1. Donner la suite $(x_n)_{n\in\mathbb{N}}$ associée à l'algorithme de Newton permettant de calculer $\alpha = \sqrt{a}$, pour tout $a \in \mathbb{R}^+$.

On pose $g(a) = \sqrt{a}$. Dès lors $f(x) = g^{-1}(x) - a = x^2 - a$.

$f$ est bien de classe $\mathcal{C}^1$ (même de classe $\mathcal{C}^\infty$). On vérifie que $f(\alpha) = 0$.

$$f(\alpha) = f(\sqrt{a})$$
$$= \sqrt{a}^2 - a$$
$$= a - a = 0$$

L'itération de Newton est telle que

$$\begin{cases} x_0 \in \mathbb{R} \\ x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \end{cases}$$

Dès lors, on a l'itération de Newton associée :

$$x_{n+1} = x_n - \frac{x_n^2 - a}{2x_n}$$

On peut simplifier, il vient que $x_{n+1} = x_n - \frac{1}{2}(x_n - \frac{a}{x_n}) = \frac{1}{2}(x_n + \frac{a}{x_n})$

2. Appliquer l'algorithme pour le calcul de $\sqrt{2}$, on pourra s'arrêter à $n = 3$ et on donnera la forme fractionnaire.

On propose de prendre $x_0 = 1$.

$x_1 = \frac{1}{2}(1+2) = \frac{3}{2}$

$x_2 = \frac{1}{2}\left(\frac{3}{2} + \frac{4}{3}\right) = \frac{17}{12}$

$x_3 = \frac{1}{2}\left(\frac{17}{12} + \frac{24}{27}\right) = \frac{577}{408}$

3. Représenter la valeur $\sqrt{2}$ en half-precision IEEE 754. Ce format inclue :

- 1 bit de signe
- 5 bits d'exposant
- 10 bits de mantisse
- Un biais $\varepsilon = 15$.

On code la valeur approchée $X = \frac{577}{408}$. On a :

$E(X) = 1$

$F(X) = X - E(X) = \frac{169}{408}$

On applique l'algorithme du calcul de la partie flottante.

| | | |
|---|---|---|
| $\frac{169}{408}$ | | |
| $\frac{169}{408} \times 2$ | $\frac{169}{204}$ | 0 |
| $\frac{169}{204} \times 2$ | $\frac{169}{102}$ | 1 |
| $\frac{67}{102} \times 2$ | $\frac{67}{51}$ | 1 |
| $\frac{16}{51} \times 2$ | $\frac{32}{51}$ | 0 |
| $\frac{32}{51} \times 2$ | $\frac{64}{51}$ | 1 |
| $\frac{13}{51} \times 2$ | $\frac{26}{51}$ | 0 |
| $\frac{26}{51} \times 2$ | $\frac{52}{51}$ | 1 |
| $\frac{1}{51} \times 2$ | $\frac{2}{51}$ | 0 |
| $\frac{2}{51} \times 2$ | $\frac{4}{51}$ | 0 |
| $\frac{4}{51} \times 2$ | $\frac{8}{51}$ | 0 |

On a $E(X), F(X) = 1,0110101000$. Il vient que :

- $s = 0$ (car $X > 0$)

- $e = 0$, donc $E = e + \varepsilon = 15 = 01111$
- $M = 0110101000$

4. Calculer l'erreur approximation $\delta_{\max}$ commise sous ce format.

L'erreur d'approximation $\delta_{\max} = 2^{e-|M|-1} = 2^{-10-1} = 2^{-11}$.

## Problème 5

Soit un entier positif $x = \langle x_{n-1}...x_0 \rangle_2$ et $k \in [\![0, n-1]\!]$. On considère l'opérateur de *décalage à gauche* $\ll$ définie tel que :

$$x \ll k = \left\langle y_i \middle| \forall i \in [\![0, n-1+k]\!], \begin{cases} y_i = 0 & \text{si } i < k \\ y_i = x_{i-k} & \text{sinon} \end{cases} \right\rangle$$

1. Calculer $4 \ll 1$, $42 \ll 3$, $12 \ll 2$.

- $\langle 4 \rangle_2 = \langle 100 \rangle_2$, donc $4 \ll 1 = \langle 1000 \rangle_2 = 8$.
- $\langle 42 \rangle_2 = \langle 101010 \rangle_2$, donc $42 \ll 3 = \langle 101010000 \rangle_2 = 336$.
- $\langle 12 \rangle_2 = \langle 1100 \rangle_2$, donc $12 \ll 2 = \langle 110000 \rangle_2 = 48$.

2. Démontrer que $x \ll k = 2^k \times x$.

On sait que $x = \sum_{j=0}^{n-1} x_i 2^j$

Posons $y = x \times 2^k = 2^k \sum_{j=0}^{n-1} x_j 2^j$

$$= \sum_{j=0}^{n-1} x_j 2^{j+k}$$

On effectue le changement d'indice $i = j + k$, dès lors :

- Si $j = 0 \Leftrightarrow i = k$
- Si $j = n-1 \Leftrightarrow i = n-1+k$

Dès lors $y = \sum_{i=k}^{n-1+k} x_{i-k} 2^i$

On retrouve bien la définition de l'opérateur décalage à gauche.

3. Comment effectuer $x \times 5$ à l'aide des opérateurs $\ll$ et $+$ ? Et $x \times 7$ ?

- $$\begin{aligned} x \times 5 &= x \times (4+1) \\ &= x \times (2^2 + 2^0) \\ &= x \times 2^2 + x \times 2^0 \\ &= x \ll 2 + x \end{aligned}$$
- $$\begin{aligned} x \times 7 &= x \times (4+2+1) \\ &= x \times (2^2 + 2^1 + 2^0) \\ &= x \times 2^2 + x \times 2^1 + x \times 2^0 \\ &= x \ll 2 + x \ll 1 + x \end{aligned}$$

4. Déterminer un algorithme permettant de réaliser $x \times n$ à l'aide du nombre minimal opérateurs $\ll$ et $+$ pour tout $n \in \mathbb{N}^*$.

**Data :** $x = \langle x_{n-1}...x_0\rangle_2 \, , n \geq 0$
**Result :** $y = x \times n$
$y \leftarrow 0$
**for** $i \in [\![0, n-1]\!]$ **do**
  **if** $(x_i == 1)$ **then**
    $y \leftarrow y + x \ll i$
  **end**
**end**

## Annexe

### *Coefficients binomiaux : Problème 1*

```c
#include <stdio.h>

int factorielle(int n) {
    return n == 0 ? 1 : n * factorielle(n - 1);
}
int binomial(int n, int k) {
    return factorielle(n)/(factorielle(k) * factorielle(n - k));
}

int main() {
    for(int n = 0; n < 15; n++) {
        for(int k = 0 ; k < n+1; k++) {
            printf("%u \t", binomial(n, k));
        }
        printf("\n");
    }
    return 0;
}
```

```
--- Sortie programme : affichage ---
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5   10  10  5   1
1   6   15  20  15  6   1
1   7   21  35  35  21  7   1
1   8   28  56  70  56  28  8   1
1   9   36  84  126 126 84  36  9   1
1   10  45  120 210 252 210 120 45  10  1
1   11  55  165 330 462 462 330 165 55  11  1
1   12  66  220 495 792 924 792 495 220 66  12  1
1   4   24  88  221 399 532 532 399 221 88  24  4   1
1   0   1   5   14  29  44  50  44  29  14  5   1   0   1
```

### ***Valeurs de*** $\ln\left(C_n^k\right)$ ***: Problème 1***

| $n$ | $\ln\left(C_n^k\right)$ |
|---|---|
| 1026 | 707.4762607243114 |
| 1027 | 708.1684346687844 |
| 1028 | 708.8615818493444 |
| 1029 | 709.5537576845151 |
| 1030 | 710.2469048650747 |
| 1031 | 710.93908258361 |
| 1032 | 711.6322297641697 |
| 1033 | 712.3244093587805 |
| 1034 | 713.0175565393406 |
| 1035 | 713.7097380027809 |

Pour $k = n/2$.