DM1 (éléments de réponses)

Exercice 1

Question 1.

□ **1.1.** Procédons par récurrence sur la taille des mots. Étant donné un plongement $p : [m] \to [n]$ de ua dans $u'a'$, deux cas sont possibles :

- si $p(m-1) = n-1$, alors nécessairement $a = (ua)[m-1] = (u'a')[n-1] = a'$ et la restriction $p|_{[m-1]}$ est strictement croissante de $[m-1]$ dans $[n-1]$, i.e. c'est un plongement de u dans u', d'où $u \preccurlyeq u'$;
- sinon, $p([m]) \subseteq [n-1]$ et donc $ua \preccurlyeq u'$.

Réciproquement, il est clair que, de chaque cas de la disjonction de droite, on peut déduire que $ua \preccurlyeq u'a'$.

□ **1.2.** De la question précédente, on déduit le programme suivant, sachant que l'équivalence que l'on vient de montrer suppose que les deux mots comportent au moins une lettre. Pour raison d'efficacité, la disjonction précédente peut être transformée en deux cas *disjoints* :

$$(a \neq a' \text{ et } ua \preccurlyeq u') \text{ ou } (a = a' \text{ et } u \preccurlyeq u')$$

puisque si $ua \preccurlyeq u'$, alors nécessairement $u \preccurlyeq u'$.

```
let teste_sous_mot v u =
  let rec aux i j =
    if i = 0 then true
    else if j = 0 then false
    else if v.[i-1] = u.[j-1] then aux (j-1) (i-1)
    else aux j (i-1)
  in aux (String.length v) (String.length u);;
```

Concernant la complexité, chaque appel récursif à `aux` se fait en temps constant, et puisque `i` décroit de 1 à chaque fois, le nombre d'appels est majoré par $|u|$. La complexité de la fonction est en $O(|u|)$.

Question 2.

□ **2.1.** On a bien $\binom{\texttt{abab}}{\texttt{ab}} = 3$, puisque c'est le nombre de façon de sélectionner dans `abab` un `a` puis un `b` :

$$\underline{\texttt{ab}}\texttt{ab} \qquad \underline{\texttt{a}}\texttt{b}\underline{\texttt{b}} \qquad \texttt{ab}\underline{\texttt{ab}}$$

□ **2.2.** La donnée d'un plongement de a^m dans a^n revient à sélectionner m positions dans l'ensemble $[n]$ à n éléments. Il y a $\binom{n}{m}$ façons de faire, d'où :

$$\binom{a^n}{a^m} = \binom{n}{m}.$$

□ **2.3.** C'est une conséquence directe de la question 1.1, sachant que les ensembles de plongements de va dans u d'une part, et de v dans u d'autre part sont disjoints (du fait du cardinal de l'ensemble de départ).

Question 3.

□ **3.1.** La terminaison de `nb_plongements` est assurée par le fait que les arguments `i` et `j` de la fonction auxilliaire `aux` sont des entiers positifs, et que les appels récursifs à cette même fonction font toujours décroître strictement `j`.

□ **3.2.** La correction de `nb_plongements` découle de la question 2.3 et, plus généralement, de la question 1.1 en ajoutant les cas de base :

$$\binom{u}{\varepsilon} = 1 \qquad v \neq \varepsilon \implies \binom{\varepsilon}{v} = 0$$
$$\binom{ua}{va} = \binom{u}{va} + \binom{u}{v} \qquad a \neq a' \implies \binom{ua'}{va} = \binom{u}{va}$$

Ces différentes équations correspondent aux quatre branches de la fonction.

Question 4.

□ **4.1.** Nous notons ici $T(i,j)$ à la place de $T(v,u)$ pour $i = |v|$ et $j = |u|$. On a, dans le pire des cas (si $i \geqslant 1$, $j \geqslant 1$ et $u[i-1] = v[i-1]$)

$$T(i,j) \leqslant 1 + T(i,j-1) + T(i-1,j-1)$$

où la constante C tient compte des différents tests et opérations arithmétiques. De plus, ayant $T(i,0) = 1$ (on effectue au plus deux tests donc, sans tenir compte de la récupération des longueurs des arguments, on a $A = 2$), il est clair par récurrence que :

$$\forall i,j \in \mathbb{N}, T(i,j) \leqslant 2 \times 2^j - 1$$

puisque $T(i,0) \leqslant 1 = 2 \times 2^0 - 1$ et, pour l'hérédité :

$$T(i,j+1) \leqslant 1 + T(i,j) + T(i-1,j) \leqslant 1 + 2 \times (2 \times 2^j - 1) = 2 \times 2^{j+1} - 1$$

En particulier, on a :

$$\forall i,j \in \mathbb{N}, T(i,j) \leqslant 2 \times 2^j$$

soit, en termes de mots et en posant $C_1 = 2$,

$$\forall v,u \in A^\star \quad T(v,u) \leqslant 2^{|u|} \times C_1$$

□ **4.2.** Il n'est pas possible de majorer $T(v,u)$ par une fonction polynômiale de $\binom{u}{v}$ (et même par n'importe quelle fonction de $\binom{u}{v}$) puisque $T(v,u)$ peut prendre des valeurs arbitrairement grandes tout en ayant $\binom{u}{v} = 0$. On peut considérer par exemple des mots de la forme

$$u = a^n \qquad v = ba^m$$

puisque le mot u ne contient pas la lettre `b`, mais l'algorithme, en lisant les lettres de droite à gauche, doit au moins parcourir le mot u en entier.

□ **4.3.** En écrivant :

$$\binom{u}{v} = 1 + 1 + 1 + \cdots + 1 + 1,$$

on remarque qu'il y a $\binom{u}{v}$ fois la constante 1, et $\binom{u}{v} - 1$ fois le signe $+$. Or, chacun de ces symboles correspond à un appel à la fonction `aux` (avec les cas `i = 0` pour 1, et `v[i - 1] = u.[j - 1]` pour $+$). Cela implique que :

$$T(v,u) \geqslant 2\binom{u}{v} - 1$$

Question 5. On peut accélérer la fonction en sauvegardant les valeurs obtenus lors des appels récursifs dans une matrice d'entiers. Cela donne le programme suivant :

```
let nb_plongements_rapide (v : string) (u : string) =
  let len_u = String.length u
  and len_v = String.length v in
    let m = Array.make_matrix (len_v + 1) (len_u + 1) 0 in
    (* Si v = epsilon, (v parmi u) = 1 *)
      for j = 0 to len_u do
        m.(0).(j) <- 1
      done;
      for j = 1 to len_u do
        for i = 1 to len_v do
          if v.[i - 1] = u.[j-1] then
            m.(i).(j) <- m.(i).(j-1) + m.(i-1).(j-1)
          else
            m.(i).(j) <- m.(i).(j-1)
        done
      done;
  m.(len_v).(len_u);;
```

La complexité est alors clairement en $O(|u| \times |v|)$ en temps comme en espace. En particulier, on a bien un temps polynomial en $|u| + |v|$.

Exercice 2

Question 1. Avant d'établir la relation de récurrence donnant le nombre c_n de comparaisons effectuées par le tri fusion sur une liste de n éléments, redonnons les trois fonctions associées à ce tri.

```
(* division de deux listes *)
let rec divise = function
    | a::b::q -> let lst1, lst2 = divise q in a::lst1, b::lst2
    | lst -> [], lst;;

(* fusion ordonnée de deux listes *)
let rec fusion lst1 lst2 = match lst1, lst2 with
    | (t1::q1), (t2::q2) when t1 <= t2 -> t1::(fusion q1 lst2)
    | lst1, (t2::q2) -> t2::(fusion lst1 q2)
    | lst1, [] -> lst1;;

(* tri fusion *)
let rec tri lst = match divise lst with
  | [], lst -> lst
  | lst1, lst2 -> fusion (tri lst1) (tri lst2);;
```

Si la fonction `tri` reçoit une liste à un seul élément, la fonction `divise` renvoie une liste vide et la liste initiale. La fonction `tri` renvoie alors directement cette seconde sous-liste qui n'est autre que la liste initiale. Aucune comparaison n'étant nécessaire, $c_1 = 0$.

Si la fonction `tri` reçoit une non vide liste `lst` de taille n, elle renvoie une liste obtenue la fusion de deux-sous listes ordonnées. Le coût de cette opération de fusion ordonnées est linéaire en la taille de `lst` puisqu'il s'agit de reconstruire une liste de n éléments. L'obtention de la sous-liste ordonnée `tri lst1` a un coût $c_{\lfloor n/2 \rfloor}$, celle de la sous-liste ordonnée `tri lst2` a un coût $c_{\lceil n/2 \rceil}$. Noter la différence sur les parties entières liée au choix adopté pour diviser une liste en deux sous-listes (fonction `divise`) : si `lst` contient un nombre impair d'élément, après division, la deuxième sous-liste comporte un élément de plus que la première sous-liste. Ainsi :

$$\forall n \geqslant 2 \qquad \underbrace{c_n}_{\text{coût de tri lst}} = \underbrace{c_{\lfloor n/2 \rfloor}}_{\text{coût de tri lst1}} + \underbrace{c_{\lceil n/2 \rceil}}_{\text{coût de tri lst2}} + \underbrace{n}_{\text{coût de fusion}}$$

Pour tout entier naturel p, posons : $u_p = c_{2^p}$. On a $u_0 = c_1 = 0$ et, d'après la relation précédente :

$$\forall p \geqslant 1 \quad u_p = u_{p-1} + u_{p-1} + 2^p$$

relation qui peut s'écrire :

$$\forall p \geqslant 1 \quad u_p = 2u_{p-1} + 2^p$$

En divisant par 2^p, il vient :

$$\begin{aligned} \forall p \geqslant 1 \quad \frac{u_p}{2^p} &= \frac{2u_{p-1}}{2^p} + \frac{2^p}{2^p} \\ &= \frac{u_{p-1}}{2^{p-1}} + 1 \end{aligned}$$

Posons à présent $v_p = u_p/2^p$. Alors $v_0 = 0$ et :

$$\forall p \geqslant 1 \quad v_p = v_{p-1} + 1$$

On déduit d'abord :

$$\forall p \geqslant 1 \quad v_p = p$$

Puis u_p, c'est-à-dire c_{2^p} :

$$\forall p \geqslant 1 \quad c_{2^p} = p \times 2^p$$

Question 2. On pose $d_n = c_{n+1} - c_n$.

□ **2.1.** Notons tout d'abord que $d_1 = c_2 - c_1$. Or $c_2 = 2c_1 + 2 = 2$. Donc $d_1 = 2$.

Pour établir une relation de récurrence définit sur la suite (d_n), envisageons deux cas.

- Si n est pair, il existe un entier naturel p tel que $n = 2p$. Alors $\lfloor n/2 \rfloor = p$ et $\lceil n/2 \rceil = p$ d'une part et $\lfloor (n+1)/2 \rfloor = p$ et $\lceil (n+1)/2 \rceil = p + 1$ d'autre part. Puis :

$$\begin{aligned} d_n &= c_{n+1} - c_n \\ &= (c_p + c_{p+1} + 2p + 1) - (c_p + c_p + 2p) \\ &= c_{p+1} - c_p + 1 \\ &= d_p + 1 \end{aligned}$$

- Si n est impair, il existe un entier naturel p tel que $n = 2p+1$. Alors $\lfloor n/2 \rfloor = p$ et $\lceil n/2 \rceil = p+1$ d'une part et $\lfloor (n+1)/2 \rfloor = p+1$ et $\lceil (n+1)/2 \rceil = p+1$ d'autre part. Puis :

$$\begin{aligned} d_n &= c_{n+1} - c_n \\ &= (c_{p+1} + c_{p+1} + 2p + 2) - (c_p + c_{p+1} + 2p + 1) \\ &= c_{p+1} - c_p + 1 \\ &= d_p + 1 \end{aligned}$$

Finalement :

$$\forall n \in \mathbb{N} \quad d_n = d_{\lfloor n/2 \rfloor} + 1$$

□ **2.2.** Pour établir que $d_n = 2 + \lfloor \log_2 n \rfloor$, on peut commercer par écrire :

$$d_n = d_{\lfloor n/2 \rfloor} + 1 \qquad d_{\lfloor n/2 \rfloor} = d_{\lfloor \lfloor n/2 \rfloor /2 \rfloor} + 1 \qquad d_{\lfloor \lfloor n/2 \rfloor /2 \rfloor} = d_{\lfloor \lfloor \lfloor n/2 \rfloor /2 \rfloor /2 \rfloor} + 1 \qquad \ldots$$

et s'interroger sur la dernière relation à écrire. Pour alléger les notations, posons :

$$n_0 = n \qquad n_1 = \lfloor n/2 \rfloor \qquad n_2 = \lfloor \lfloor n/2 \rfloor /2 \rfloor \qquad \ldots$$

On peut écrire :

$$\forall i \in \mathbb{N} \qquad n_{i+1} = \left\lfloor \frac{n_i}{2} \right\rfloor$$

Cette relation de récurrence définit une suite d'entiers naturels strictement décroissante puis stationnaire à partir d'un certain rang. En effet, pour tout entier naturel n, il existe un unique enter naturel N tel que :

$$2^N \leqslant n < 2^{N+1}$$

Cet entier N vaut :

$$N = \lfloor \log_2 n \rfloor$$

Et puisque :

$$1 \leqslant \frac{n}{2^N} < 2$$

il vient : $n_N = 1$. On peut donc mettre les relations du début de la question sous la forme suivante.

$$\begin{aligned} d_{n_0} &= d_{n_1} + 1 \\ d_{n_1} &= d_{n_2} + 1 \\ d_{n_2} &= d_{n_3} + 1 \\ &\ldots \\ d_{n_{N-1}} &= d_{n_N} + 1 \end{aligned}$$

Par somme télescopique, il vient alors :

$$d_{n_0} = d_{n_N} + N$$

Or $d_{N_n} = d_1 = 2$ et $d_{n_0} = d_n$. D'où :

$$\forall n \in \mathbb{N} \qquad d_n = 2 + N = 2 + \lfloor \log_2 n \rfloor$$

□ **2.3.** Sachant $d_n = c_{n+1} - c_n$, par somme télescopique à nouveau, il vient :

$$\forall n \in \mathbb{N}^* \qquad \sum_{j=1}^{n-1} d_j = \sum_{j=1}^{n-1} (c_{j+1} - c_j)$$

soit :

$$\forall n \in \mathbb{N} \qquad c_n - c_1 = \sum_{j=1}^{n-1} \left(2 + \lfloor \log_2 j \rfloor\right) = 2(n-1) + \sum_{j=1}^{n-1} \lfloor \log_2 j \rfloor$$

Question 3.

□ **3.1.** Un entier naturel non nul k peut être codé sur $\nu(k) = 1 + \lfloor \log_2 k \rfloor$ bits.

Sur $p \in \mathbb{N}$ bits, on peut coder les entiers 0, 1, 2, 3, ..., $2^p - 1$. Ce qui représente un total de 2^p entiers. Les entiers supérieurs ou égaux à $2^p + 1$ sont donc codés sur au moins $p+1$ bits. Si $n > 2^p + 1$, l'ensemble $[\![2^p + 1, n]\!]$ contient $n - 2^p$ entiers. Ainsi, parmi les entiers 1, 2, ..., $n-1$, il y en a exactement $n - 2^p$ qui ont au moins $(p+1)$ bits dans leur représentation binaire, avec $p \leqslant \lfloor \log_2 n \rfloor$.

□ **3.2.** Avec la définition de l'énoncé, c'_n représente la somme des nombres de bits nécessaires pour représenter chacun des entiers 1, 2, 3, ..., $n-1$. C'est donc le nombre total de bits nécessaires pour représenter tous les entiers de $[\![1, n-1]\!]$. Par ailleurs, d'après le résultat de la question précédente, parmi entiers 1, 2, ..., $n-1$:

- $n-2^0$ ont au moins $0+1=1$ bit dans leur représentation binaire ; ce sont les entiers de $[\![2^0, n-1]\!]$;
- $n-2^1$ ont au moins $1+1=2$ bits dans leur représentation binaire ; ce sont les entiers de $[\![2^1, n-1]\!]$;
- $n-2^2$ ont au moins $2+1=3$ bits dans leur représentation binaire ; ce sont les entiers de $[\![2^2, n-1]\!]$;
- ...
- $n-2^i$ ont au moins $i+1$ bits dans leur représentation binaire ; ce sont les entiers de $[\![2^i, n-1]\!]$;
- ...
- $n-2^{\lfloor \log_2 n \rfloor}$ ont au moins $(\lfloor \log_2 n \rfloor + 1)$ bits dans leur représentation binaire ; ce sont les entiers de $[\![2^{\lfloor \log_2 n \rfloor}, n-1]\!]$;

Pour $i \in [\![0, \lfloor \log_2 n \rfloor]\!]$, chaque entier de $[\![2^i, n-1]\!]$ contribue à hauteur de 1 bit dans le décompte $n-2^i$. Ajouter ces nombres revient alors à comptabiliser chaque bit de chaque nombre. Dit autrement, la somme de tous ces nombres représente le nombre total de bits nécessaires pour représenter tous les entiers de $[\![1, n-1]\!]$, c'est-à-dire c'_n. Ainsi :

$$\forall n \in \mathbb{N}^* \qquad c'_n = \sum_{j=0}^{\lfloor \log_2 n \rfloor} (n - 2^j)$$

On vient d'établir que :

$$\sum_{j=1}^{n-1} (1 + \lfloor \log_2 j \rfloor) = \sum_{j=0}^{\lfloor \log_2 n \rfloor} (n - 2^j)$$

La figure 1 illustre ce résultat en représentant chaque bit par une case. En sommant horizontalement ou verticalement le nombre de cases, on établit visuellement cette égalité.

□ **3.3.** Cette dernière relation permet le calcul de c'_n sous la forme :

$$\forall n \in \mathbb{N}^* \qquad c'_n = n \times (1 + \lfloor \log_2 n \rfloor) - (2^{1+\lfloor \log_2 n \rfloor} - 1)$$

Par ailleurs, en utilisant l'expression de $\nu(j)$:

$$\forall n \in \mathbb{N}^* \qquad c'_n = \sum_{j=1}^{n-1} (1 + \lfloor \log_2 j \rfloor) = n - 1 + \sum_{j=1}^{n-1} \lfloor \log_2 j \rfloor$$

et, d'après la question 2.3 :

$$\forall n \in \mathbb{N}^* \qquad c_n = 2(n-1) + \sum_{j=1}^{n-1} \lfloor \log_2 j \rfloor$$

On voit donc que :

$$\forall n \in \mathbb{N}^* \qquad c_n = (n-1) + c'_n$$

Ainsi, $c_n - (n-1)$ représente le nombre total de bits nécessaires pour représenter tous les entiers de $[\![1, n-1]\!]$. En outre, on a :

$$\forall n \in \mathbb{N}^* \qquad c_n = (n-1) + n \times (1 + \lfloor \log_2 n \rfloor) - (2^{1+\lfloor \log_2 n \rfloor} - 1)$$

soit finalement :

$$\forall n \in \mathbb{N}^* \qquad c_n = n + n \times (1 + \lfloor \log_2 n \rfloor) - 2^{1+\lfloor \log_2 n \rfloor}$$

Question 4. En notant que :

$$\lfloor \log_2 n \rfloor = \log_2 n - \{\log_2 n\}$$

on peut écrire :

$$\begin{aligned} c_n &= n + n \times (1 + \log_2 n - \{\log_2 n\}) - 2^{1+\log_2 n - \{\log_2 n\}} \\ &= n \times \log_2 n + n + n \times (1 - \{\log_2 n\}) - 2^{\log_2 n} \times 2^{1-\{\log_2 n\}} \\ &= n \times \log_2 n + n + n \times (1 - \{\log_2 n\}) - n \times 2^{1-\{\log_2 n\}} \\ &= n \times \log_2 n + n \times \left[1 + (1 - \{\log_2 n\}) - 2^{1-\{\log_2 n\}}\right] \end{aligned}$$

et en posant, pour tout nombre réel x, $\varphi(x) = 1 + x - 2^x$, on obtient :

$$\forall n \in \mathbb{N}^* \qquad c_n = n \times \log_2 n + n\varphi(1 - \{\log_2 n\})$$

Un étude de la fonction φ montre qu'elle admet un maximum positif en $-\log_2 \log_2 e$. Donc, pour n assez grand, le terme $n \times \log_2 n$ domine $n\varphi(1 - \{\log_2 n\})$, ce qui établit (proprement !) que $c_n \in O(n \log_2 n)$, complexité temporelle au pire,

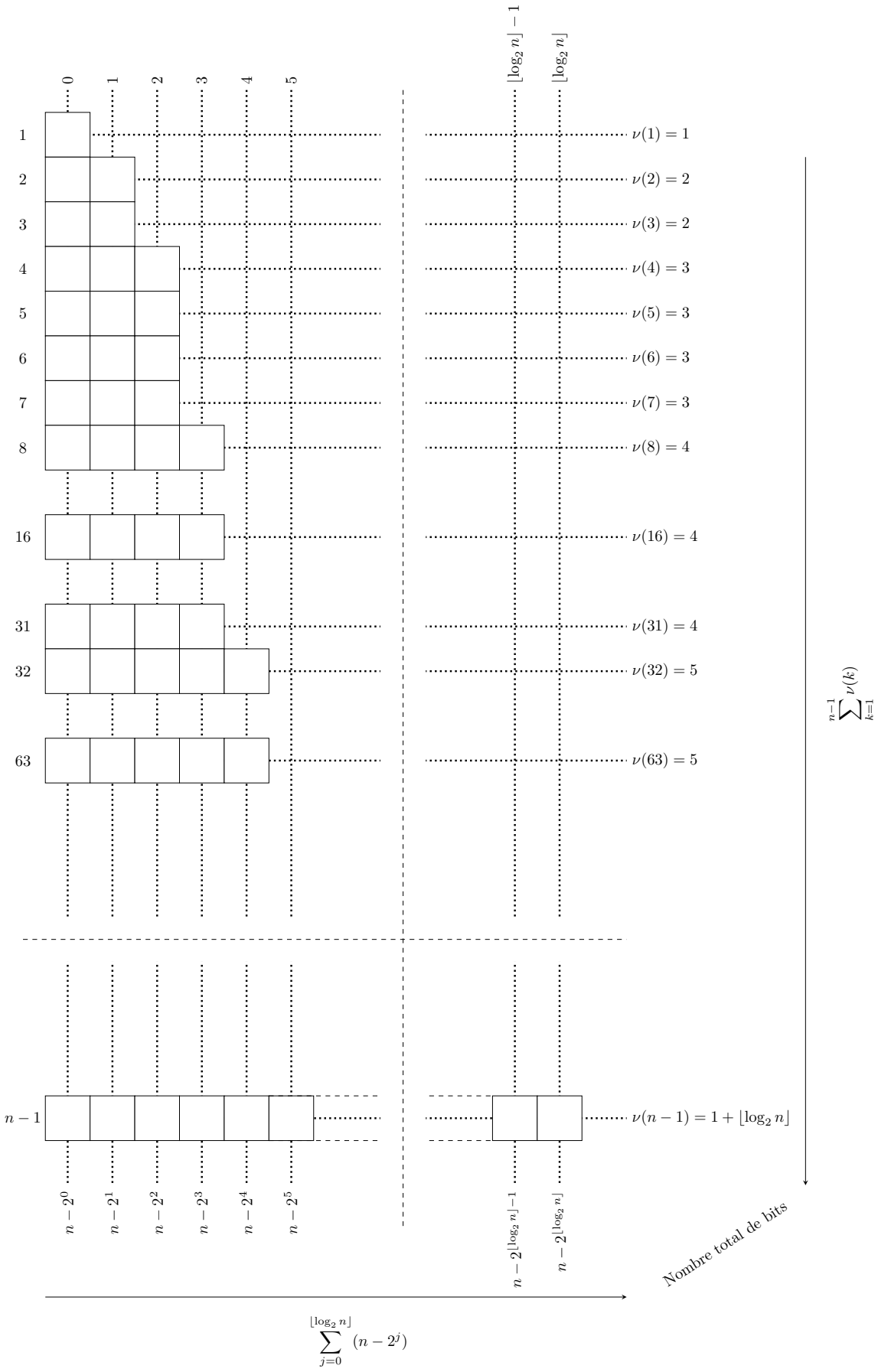
0
1
2
3
4
5
$\lfloor \log_2 n \rfloor - 1$
$\lfloor \log_2 n \rfloor$
1
2
3
4
5
6
7
8
16
31
32
63
$n-1$
$\nu(1) = 1$
$\nu(2) = 2$
$\nu(3) = 2$
$\nu(4) = 3$
$\nu(5) = 3$
$\nu(6) = 3$
$\nu(7) = 3$
$\nu(8) = 4$
$\nu(16) = 4$
$\nu(31) = 4$
$\nu(32) = 5$
$\nu(63) = 5$
$\nu(n-1) = 1 + \lfloor \log_2 n \rfloor$
$n - 2^0$
$n - 2^1$
$n - 2^2$
$n - 2^3$
$n - 2^4$
$n - 2^5$
$n - 2^{\lfloor \log_2 n \rfloor - 1}$
$n - 2^{\lfloor \log_2 n \rfloor}$
Nombre total de bits
$\sum_{k=1}^{n-1} \nu(k)$
$\sum_{j=0}^{\lfloor \log_2 n \rfloor} (n - 2^j)$

Figure 1

en termes de nombres de comparaisons, du tri fusion. La figure 2 illustre les variations de $\psi : x \mapsto \varphi(1 - \{\log_2 x\})$ pour $x \in [1, 16]$.

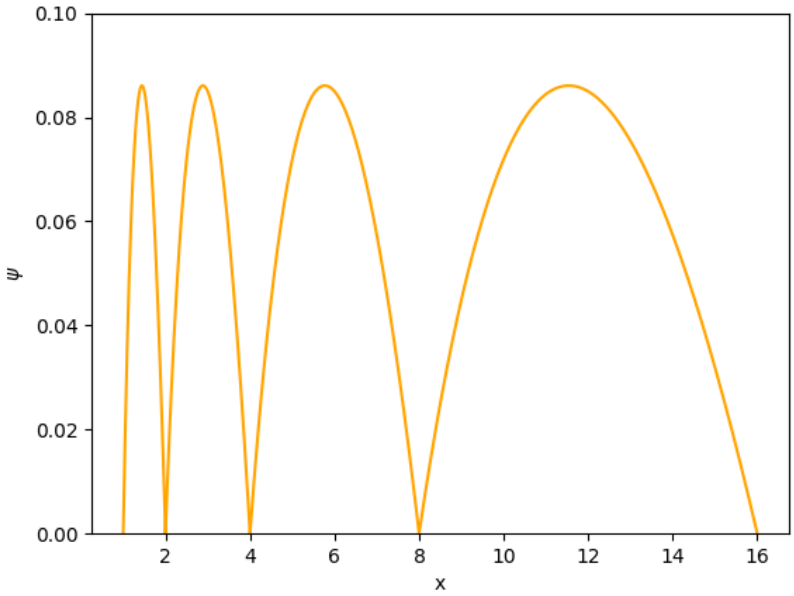

Figure 2