rapport : Mini-Projet d'algorithmique confitures

Hakim CHEKIROU & Thomas DIEN

09 novembre 2018

Table des matières

Chapitre 1

partie 1

1.1 Recherche exhaustive

1.1.1 Question 1

Cas de base

Pour, $S < 0$:
$rechercheExhaustive$ retourne $+\infty$.
Pour, $S = 0$:
$rechercheExhaustive(k, V, S)$ retourne 0 qui est le nombre minimum de bocaux nécessaire.
Cas inductif supposons que $\forall p \leq n$ $RechercheExhaustive(k, v, p)$ se termine et retourne le nombre minimum de bocaux nécessaire pour une quantité p de confiture.
montrons que cela reste vrai pour une quantité $S = n + 1$:
avant la première itération de la boucle $nbcont = S$(le nombre minimum de bocaux de 1 dg).
Pour chaque itération i de la boucle pour :$rechercheExhaustive(k, v, S - V[i])$ est appelé.
Puisque $S - V[i] < S \Leftrightarrow S - V[i] \leq n$, donc $recherhceExhaustive(kv, S - V[i])$ retourne le nombre minimum x de bocaux par hypothèse de récurrence, si $x + 1 < nbcont$ alors $nbCont = x + 1$.
Donc à la fin de la i^{me} itération de la boucle, l'algorithme retourne le nombre minimum de bocaux si on prend d'abord un bocal dont la capacité $c \in \{1, ..., i\}$. **Conclusion** à la fin de la k^{me} itération de la boucle, $nbcont$ contient le nombre minimum de bocaux nécessaire pour une quantité $S = n + 1$ en prenant d'abord le bocal dont la capacité $c \in \{1, ..., k\}$, donc l'algorithme se termine et est valide.

1.1.2 Question 2

a.

$$a(s) = \begin{cases} 0 & \text{si } s = 0 \\ 2 & \text{si } s = 1 \\ a(s-1) + a(s-2) + 2 & \text{si } s \geq 2 \end{cases}$$

b.

montrons que pour tout entier $s \geq 0$, on a $b(s) \leq a(s) \leq c(s)$:

1 montrons que pour tout entier $s \geq 0$ on a $a(s) \leq c(s)$:
par récurrence :
Cas de base , $S = 0$: $a(0) = 0 \leq 0 = c(0)$ propriété vérifiée.
hypothèse de récurrence : supposons que pour un entier s , $a(s) \leq c(s)$.
montrons que $a(s + 1) \leq c(s + 1)$ est vérifiée.
on a $a(s) \leq c(s)$ par hypothèse.
on a aussi $a(s - 1) \leq a(s - 1) + a(s - 2) + 2 = a(s)$ par définition de a(s)
et $a(s) \leq c(s)$ donc $a(s - 1) < c(s)$
donc on obtient $a(s) + a(s - 1) < 2c(s)$
$\leftrightarrow a(s) + a(s - 1) + 2 < 2c(s) + 2 \leftrightarrow a(s + 1) < c(s + 1)$ propriété vérifiée.

2 montrons que pour tout entier $s \geq 0$ on a $b(s) \leq a(s)$:
par récurrence :
Cas de base , $S = 0$: $b(0) = 0 \leq 0 = a(0)$ propriété vérifiée.
$s = 1$: $b(1) = 2 \leq 2 = a(1)$ propriété vérifiée.
hypothèse de récurrence : supposons que pour un entier $p < s$, $b(p) \leq a(p)$, montrons que $b(s) \leq a(s)$ est vérifiée.
on a $b(s-2) \leq a(s-2) < a(s-2) + a(s-3) + 2 = a(s-1)$ par hypothèse.
donc on obtient $2b(s-2) < a(s-1) + a(s-2)$
$\leftrightarrow 2b(s-2) + 2 \leq a(s-1) + a(s-2) + 2 \leftrightarrow b(s) < a(s)$ propriété vérifiée.
donc de la question, b.1 et b.2, on déduit que :

$$b(s) \leq a(s) \leq c(s)$$

c.

le terme général de la suite $c(s)$:
$c(s) = 2^{s+1} - 2$
Par récurrence :
Cas de base :
pour $s = 0$: on a $c(s) = 0 = 2^{0+1} - 2$ vérifié. hypothèse de récurrence :
pour un entier positif s , supposons : $c(s) = 2^{s+1} - 2$
montrons que c'est vrai au rang $s + 1$
on a :

$$\begin{aligned} c(s+1) &= 2c(s) + 2 \\ &= 2(2^{s+1} - 2) + 2 \\ &= 2^{s+2} - 2 \end{aligned}$$

, propriété vérifiée.
conclusion : $c(s) = 2^{s+1} - 2$

d.

par récurrence **cas de base :**
Pour $s = 0$, $b(0) = 0 = 2^{\lceil \frac{0}{2} \rceil + 1} - 2$
Pour $s = 1$, $b(1) = 2 = 2^{\lceil \frac{1}{2} \rceil + 1} - 2$
Hypothèse de récurrence : supposons que pour un entier positif $p < s$, $b(p) = c(\lceil \frac{p}{2} \rceil)$
montrons que $b(s) = c(s)$

$$\begin{aligned} b(s) &= 2b(s-2) + 2 && (1.1) \\ &= 2(c(\lceil \frac{s-2}{2} \rceil)) + 2 && (1.2) \\ &= 2(2^{\lceil \frac{s-2}{2} \rceil + 1} - 2) + 2 && (1.3) \\ &= 2(2^{\lceil \frac{s}{2} - 1 \rceil + 1} - 2) + 2 && (1.4) \\ &= 2(2^{\lceil \frac{s}{2} \rceil - 1 + 1} - 2) + 2 && (1.5) \\ &= 2 \times 2^{\lceil \frac{s}{2} \rceil} - 2 && (1.6) \\ &= 2^{\lceil \frac{s}{2} \rceil + 1} - 2 && (1.7) \\ &= c(\lceil \frac{s}{2} \rceil) && (1.8) \end{aligned}$$

e.

le terme général de la suite b(s) :

$$b(s) = 2^{\lceil \frac{s}{2} \rceil + 1} - 2$$

Encadrement du nombre d'appels récursifs, $a(s)$ réalisés par $rechercheEchaustive(2, [1, 2], s)$:

$$2^{\lceil \frac{s}{2} \rceil + 1} - 2 \leq a(s) \leq 2^{s+1} - 2$$

nous pouvons donc conclure que l'algorithme rechercheExhaustive est exponentiel.

1.2 Programmation dynamique

1.2.1 Question 3

a) la valeur de $m(S)$ en fonction des valeurs $m(s,i)$:

$$m(S) = \min_{i \in 1...k}(m(S,i)) = m(S,k)$$

b) preuve de la relation de récursion suivante pour tout $i \in \{1,\ldots,k\}$:

$$m(s,i) = \begin{cases} 0 & \text{si } s = 0 \\ min\{m(s,i-1), m(s-V[i],i)+1\} & sinon \end{cases}$$

Cas de base : Pour $s = 0$:
on a $m(0,i) = 0, \forall i \in \{1,\ldots,k\}$ par définition.
Cas inductif :supposons que
$\forall p < s$ et $p > 0 : m(p,i) = min\{m(p,i-1), m(p-V[i],i)+1\}$
avec $i \in \{1,\ldots,k\}$
montrons cette propriété au rang s.
si $V[i] > s$ **:** $m(s-V[i],i)+1 = +\infty$. On ne prend pas le bocal i.
Donc : $m(s,i) = m(s,i-1) = min\{m(p,i-1), +\infty\}$,relation vérifiée
sinon si $V[i] \leq s$ **:** On ne prend V[i] que dans le cas ou le nombre de bocaux est inférieur au cas ou on ne le prend pas.C'est-à-dire si $m(s-V[i],i)+1 \leq m(s,i-1)$.
sinon la confiture est repartie dans les $i-1$ autres bocaux, donc $m(s,i) = m(s,i-1)$
donc $m(s,i) = min\{m(s,i-1), m(s-V[i],i)+1\}$.
Conclusion : On conclut que la relation de récurrence est vérifiée.

1.2.2 Question 4

a) On remplit d'abord les cas m(0, i) avec $i \in 0, ..., k$.
On remplit ensuite les cas m(s, 0) pour $s \in 1, ..., S$
Puis on commence par la case (1,1) et on remplit toute la matrice ligne par ligne.
justification : à chaque étape on cherche le minimum entre m(s, i-1) et m(s -V[i])+1 qui sont des cas déjà traités sur la même ligne s(la même colonne i respectivement).

b) pseudo-code de l'algorithme

```
algorithme AlgoProgDyn(S : entier, K : entier, V : tableau de k entiers) : entier
    M←créer un tableau de (S+1)*(K+1) entiers
    i,s : entiers
    pour i de 0 à K faire
    |   M[0][i] ← 0
    fin
    pour s de 1 à S faire
    |   M[i][0] ← +∞
    fin
    pour s de 1 à S faire
    |   pour i de 1 à k faire
    |   |   si s − V[i] < 0 alors
    |   |   |   M[s][i] ← M[s][i − 1]
    |   |   sinon
    |   |   |   si M[s][i − 1] < M[s − V[i]][i] + 1 alors
    |   |   |   |   M[s][i] ← M[s][i − 1]
    |   |   |   sinon
    |   |   |   |   M[s][i] ← M[s-V[i]][i] + 1
    |   |   |   fin
    |   |   fin
    |   fin
    fin
    retourner M[s][k]
```

c) Complexité temporelle :
l'algorithme comporte deux boucle imbriquée majorées par S et K ainsi que deux boucle d'initialisation, une en O(S) et une en O(K).
la complexité est donc en $O(S \times K)$
Complexité spatiale :
l'algorithme utilise une matrice de $S \times K$ cases donc la complexité spatiale est en $O(S \times K)$

1.2.3 Question 5

a) On remplace notre matrice d'entiers par une matrice d'enregistrements, comportant un entier et un tableau de i entiers (i étant le nombre de bocaux a chaque case). A chaque itération de la boucle, dans le cas ou m(s, i-1) $>$ m(s-V[i], i)+1 alors on incrémente la case V[i].
si m(s, i-1) $<$ m(s-V[i], i)+1 alors le tableau des valeurs prises ne change pas.
la Complexité spatiale est en $O(S \times K^2)$

b) pseudo-code de l'algorithme
algorithme algorithme-retour(S : entier ,K : entier , V tableau de K entiers, M : Matrice de (S+1)*(K+1) entiers) : tableau d'entiers

```
tab← tableau de K entiers initialisé a 0
total,p,q : entiers
total ← 0
p ← S
q ← K
tant que total < S faire
    si M[p][q] = M[p − V[q]][q] + 1 alors
        total ← total + V[q]
        p ← p − V[q]
        tab[q] ← tab[q] + 1
    sinon
        q ← q − 1
    fin
fin
retourner tab
```

Complexité temporelle :
Au pire cas , si seul des bocaux de capacité = 1 sont utilisés, l'algorithme est en $O(S + K)$ car on décrémente la variable q jusqu'à 1 puis la variable p jusqu'à 0
Complexité spatiale :
Cet algorithme utilise un tableau de K entiers,donc la complexité spatiale est en $O(K)$.

1.2.4 Question 6

L'algorithme **AlgoProgDyn** est en $O(S \times K)$ et l'**algorithme-retour** est en $O(S + K)$. La complexité totale est en $O(S \times K)$. Donc l'algorithme est bien polynomial en nombre de cases soit $S \times K$.

1.3 Cas particulier et algorithme glouton

1.3.1 Question 7

pseudo-code de l'algorithme

algorithme AlgoGlouton(K : entier , V tableau de K entiers, S : entier) : entier
total, nb, i : entiers
total ← S
nb ← 0
tant que *total != 0* **faire**
  i← 1 `/* recherche du plus grand bocal */`
  tant que $i \leq K$ *et* $total - V[i] \geq 0$ **faire**
    $i \leftarrow i + 1$
  fin
  `/* la case i-1 contient le bocal le plus grand */`
  tant que $total - V[i-1] \geq 0$ **faire**
    $total \leftarrow total - V[i-1]$
    $nb \leftarrow nb + 1$
  fin
fin
retourner nb

Dans le pire cas, c'est-à-dire si on ne prend que des bocaux de capacité 1, l'algorithme fait S itérations donc l'algorithme est en $O(S)$.

1.3.2 question 8

Preuve qu'il existe des systèmes de capacités qui ne sont pas glouton-compatibles.
Contre exemple :
Soit le système de capacité V = [1, 4, 5, 6].
Pour $S = 9$, l'algorithme Glouton choisi les bocaux $6, 1, 1, 1$ donc 4 bocaux, alors que la solution optimale est $5, 4$ avec 2 bocaux.

1.3.3 question 9

a) preuve qu'il existe un plus grand indice j pris dans $\{1, \ldots, k\}$ tel que $o_j < g_j$.
Puisque la solution g est différente de o, prenons j le plus grand bocal ou les deux solutions diffèrent.
par construction de l'algorithme glouton, g_j est toujours maximisé, donc $o_j \leq g_j$, et puisque les deux solutions différent au bocal j alors $o_j < g_j$
On en conclut qu'il existe un plus grand indice j pris dans $\{1, \ldots, k\}$ tel que $o_j < g_j$.

b) preuve de $\sum_{i=1}^{j} V[i]g_j = \sum_{i=1}^{j} V[i]o_j$
On a :

$$\sum_{i=1}^{k} V[i]g_j = \sum_{i=1}^{k} V[i]o_j \tag{1.9}$$

car o et g sont solutions

$$\sum_{i>j}^{k} V[i]g_j = \sum_{i>j}^{k} V[i]o_j \tag{1.10}$$

car j est le plus grand bocal ou g et o diffèrent

(1.11)

de (1.9) et (1.10), ont obtient :

$$\sum_{i=1}^{k} V[i]g_j - \sum_{i>j}^{k} V[i]g_j = \sum_{i=1}^{k} V[i]o_j - \sum_{i>j}^{k} V[i]o_j \tag{1.12}$$

$$\sum_{i=1}^{j} V[i]g_j = \sum_{i=1}^{j} V[i]o_j \tag{1.13}$$

c) preuve de : $V[j] \leq \sum_{i=1}^{j-1} V[i]o_i$
on a

$$\sum_{i=1}^{j} V[i]g_i = \sum_{i=1}^{j} V[i]o_i \tag{1.14}$$

$$\sum_{i=1}^{j-1} V[i]g_i + V[j]g_j = \sum_{i=1}^{j-1} V[i]o_i + V[j]o_j \tag{1.15}$$

$$\sum_{i=1}^{j-1} V[i]g_i + V[j](g_j - o_j) = \sum_{i=1}^{j-1} V[i]o_i \tag{1.16}$$

on a : $g_j - o_j \geq 1$ (question 9.a)
donc

$$V[j] \leq V[j](g_j - o_j) \leq \sum_{i=1}^{j-1} V[i]g_i + V[j](g_j - oj) = \sum_{i=1}^{j-1} V[i]o_i$$

on en déduit que :

$$V[j] \leq \sum_{i=1}^{j-1} V[i]o_i$$

Interprétation : La quantité de confiture repartie sur les bocaux de capacité inférieure a $V[j]$ est supérieure ou égale a la capacité du bocal j.

d) on a $o_i \leq d-1$ pour tout indice $i \in \{1, \ldots, j-1\}$.
donc :

$$\sum_{i=1}^{j-1} V[i]o_i \leq \sum_{i=1}^{j-1} V[i](d-1) = \sum_{i=1}^{j-1} d^{i-1}(d-1)$$

et

$$\sum_{i=1}^{j-1} d^{i-1}(d-1) = \frac{d^{j-1}-1}{d-1}(d-1) = d^{j-1} - 1 < d^{j-1} = V[j]$$

donc

$$\sum_{i=1}^{j-1} V[i]o_i < V[j]$$

contradiction avec le résultat précédent.

e) Preuve que o' est bien solution du problème :
on a :

$$S = \sum_{i=1}^{k} V[i]o_i \tag{1.17}$$

$$= \sum_{\substack{i\neq l \\ i\neq l+1}}^{k} V[i]o_i + o_l V[l] + o_{l+1}V[l+1] \tag{1.18}$$

$$= \sum_{\substack{i\neq l \\ i\neq l+1}}^{k} V[i]o'_i + o_l V[l] + o_{l+1}V[l+1] \tag{1.19}$$

$$= \sum_{\substack{i\neq l \\ i\neq l+1}}^{k} V[i]o'_i + (d+o'_l)V[l] + (o'_{l+1}-1)V[l+1] \tag{1.20}$$

$$= \sum_{\substack{i\neq l \\ i\neq l+1}}^{k} V[i]o'_i + (d+o'_l)d^{l-1} + (o'_{l+1}-1)d^l \tag{1.21}$$

$$= \sum_{\substack{i\neq l \\ i\neq l+1}}^{k} V[i]o'_i + d^l + o'_l d^{l-1} + o'_{l+1}d^l - d^l \tag{1.22}$$

$$= \sum_{\substack{i\neq l \\ i\neq l+1}}^{k} V[i]o'_i + o'_l V[l] + o'_{l+1}V[l+1] \tag{1.23}$$

$$= \sum_{i=1}^{k} V[i]o'_i \tag{1.24}$$

Donc ,

$$S = \sum_{i=1}^{k} V[i]o'_i$$

On en conclut que o' est bien solution du problème.
De plus, le nombre de bocaux de la solution o' est inférieur de $d-1$ bocaux à la solution o. Or o est optimale, absurde. o n'est donc pas différent de g et g est optimale. Le système Expo est donc glouton-compatible.

1.3.4 Question 10

Preuve que tout système de capacité V avec $k = 2$ est glouton-compatible :
V est un Système Expo avec $V[1] = 1$ et $V[2] = d$. Puisque tout système Expo est glouton-compatible (montré a la question précédente), alors V l'est aussi.

1.3.5 Question 11

Preuve que l'algorithme **TestGloutonCompatible** est pseudo-polynomial :
La première boucle Pour est majorée par $V[k-1] + V[k] - 1$ elle est donc en $O(V[k])$
La deuxième boucle pour fait k itérations, mais on a $k \leq V[k]$ donc elle est en $O(V[k])$
Chaque itération fait appel a $AlgoGlouton(S)$ et $AlgoGlouton(S - V[j])$, la complexité de AlgoGlouton(S,K,V) est $O(S)$. Donc chaque itération est en $O(V[k])$ car S est majorée par $V[k-1] + V[k] - 1$.
On en conclut que TestGloutonCompatible compatible est en $O(V[k]^3]$, donc pour des valeurs raisonnable de V[k] TestAlgoGlouton est polynomiale (pseudo-polynomiale).

Chapitre 2

Mise en œuvre

2.1 implémentation

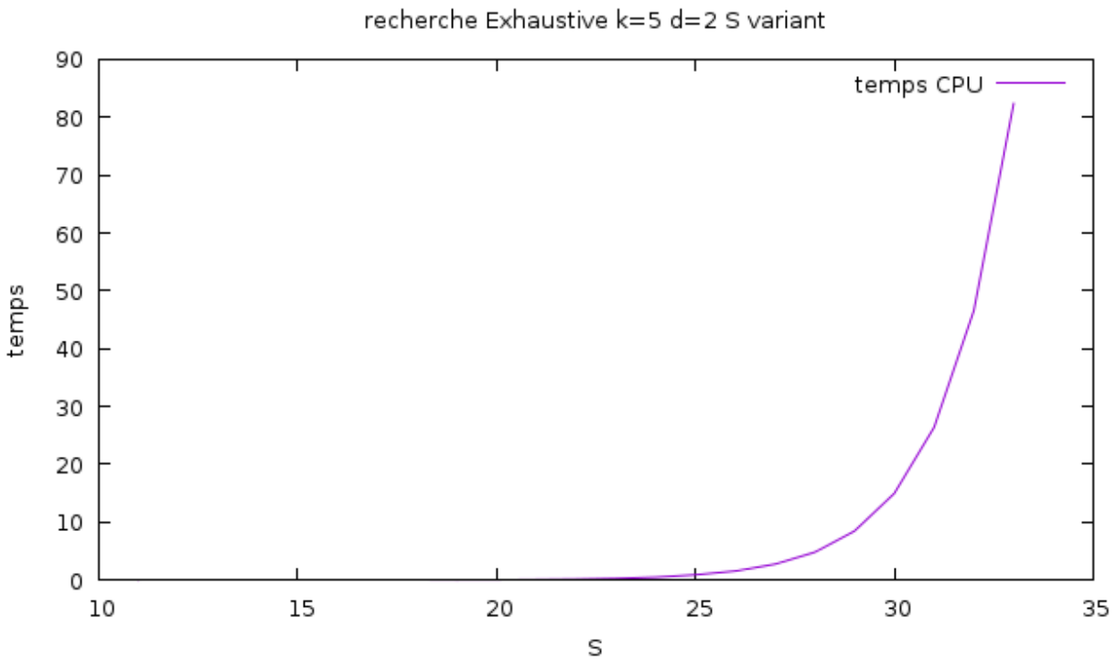
l'implémentation a été réalisée en python et le code source ainsi que les fichiers utilisés sont fournis en annexe.

2.2 analyse de complexité expérimentale

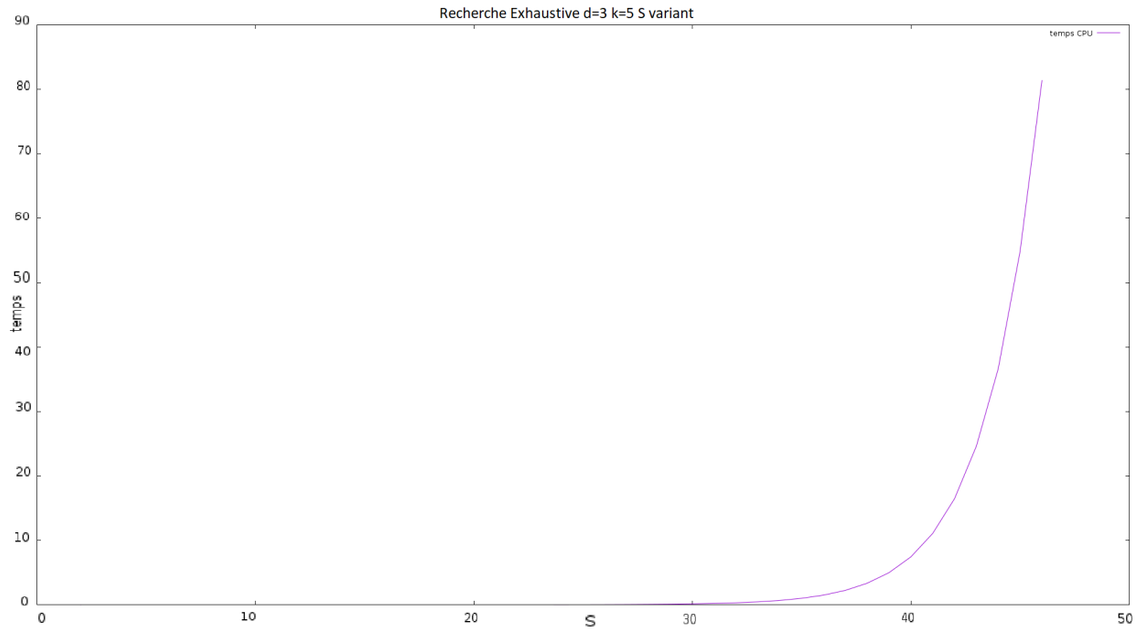
2.2.1 Question 12

Analyse de l'algorithme RechercheExhaustive

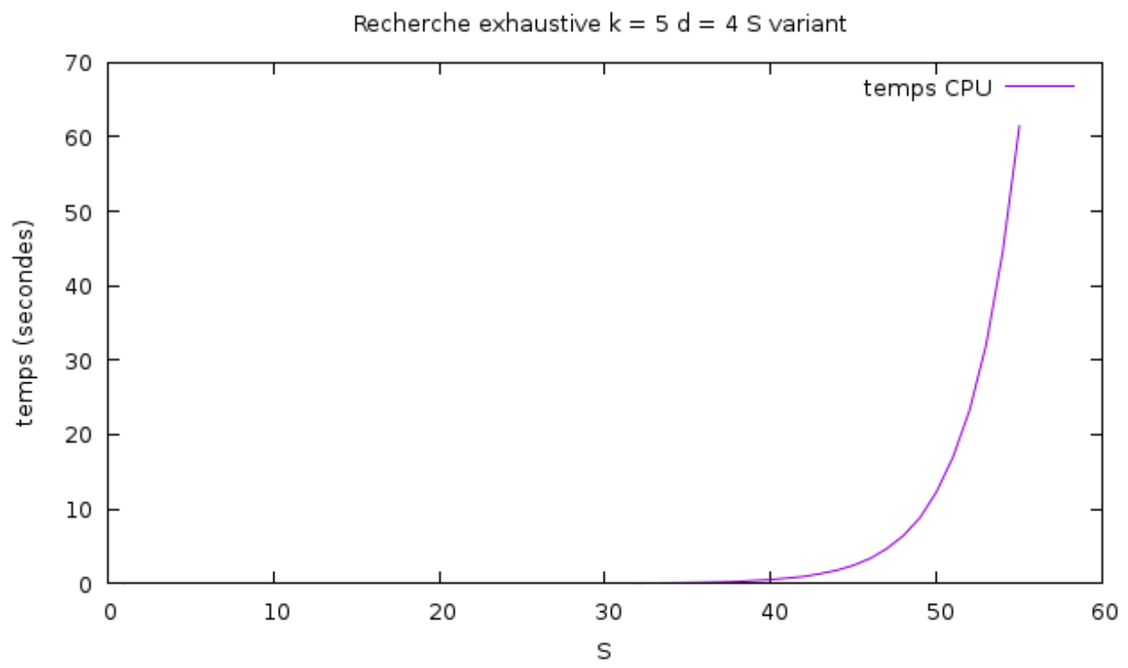
Tout les tests ont été réalisés sur une machine équipée d'un processeur intel core I7, 16 GB de ram et sous l'OS Linux. Nous avons fait varier les paramétrés S et K sur les 3 systèmes Expo.
les courbes des temps d'exécution sont présentées ci-dessous :
Temps d'exécution en variant S :
Pour d = 2 :



Pour d = 3 :



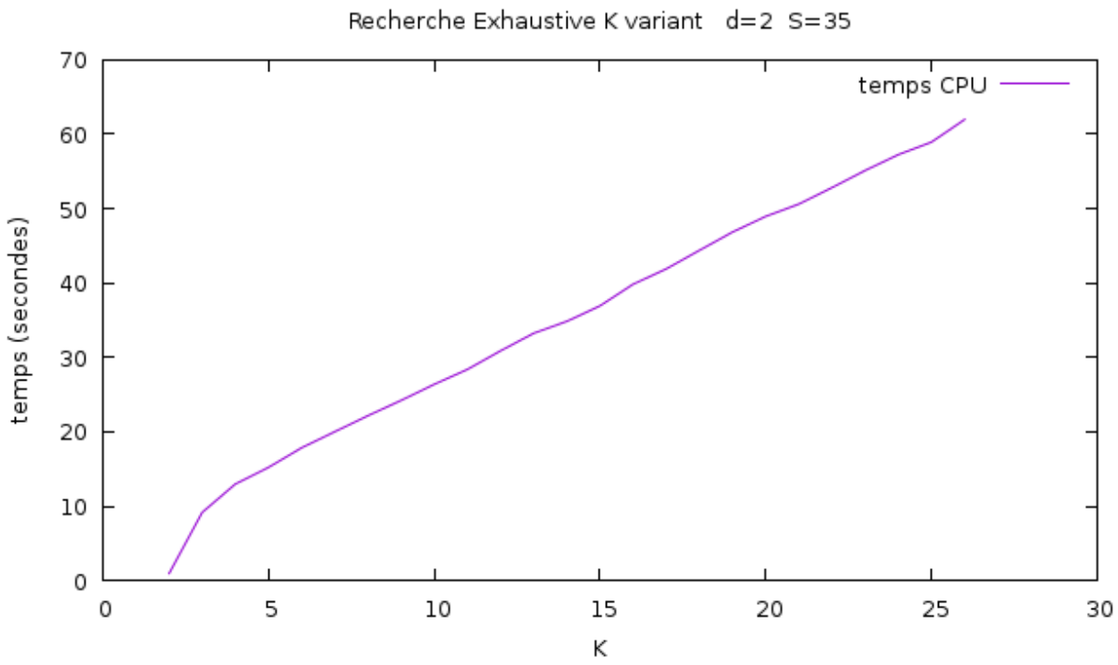
Pour d = 4 :



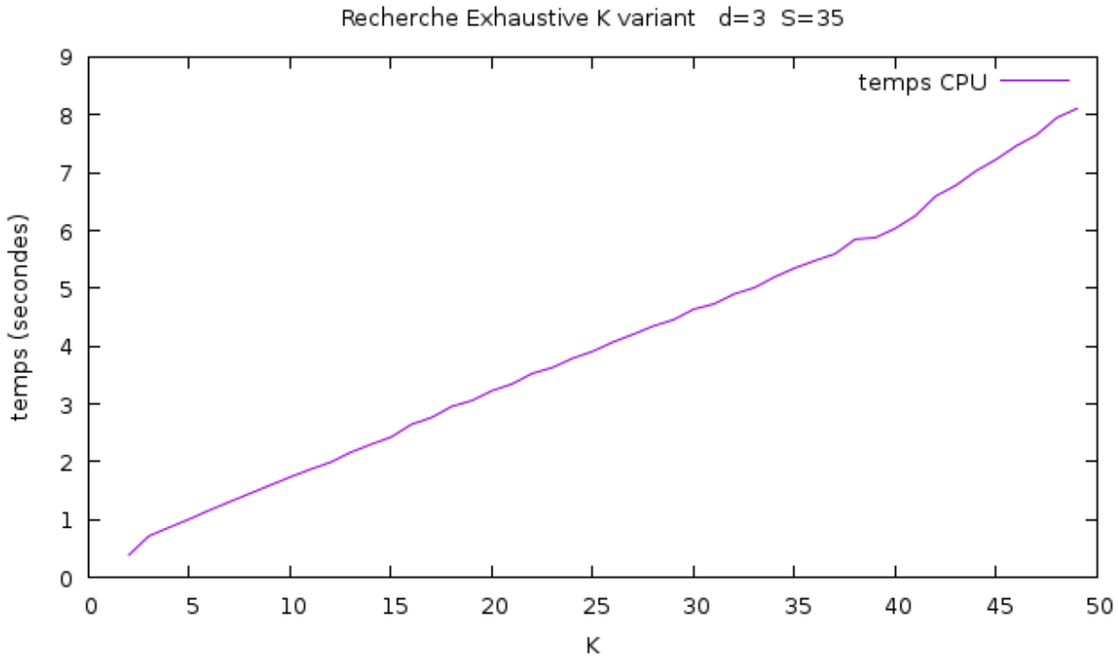
analyse : On remarque que les 3 courbes démontrent bien le résultat de l'analyse théorique de cet algorithme qui est exponentiel en faisant varier S.
Pour $k = 5$ les temps d'exécution deviennent très vite trop grands pour une utiliser l'algorithme.

Temps d'exécution en variant K :

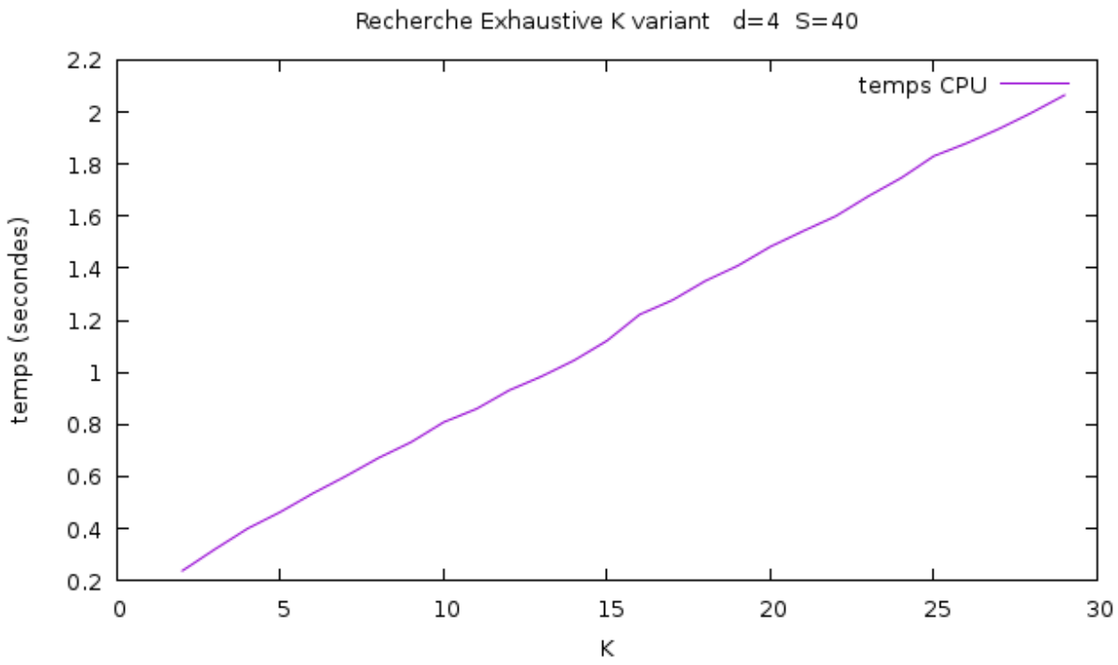
Pour d = 2 :



Pour d = 3 :



Pour d = 4 :



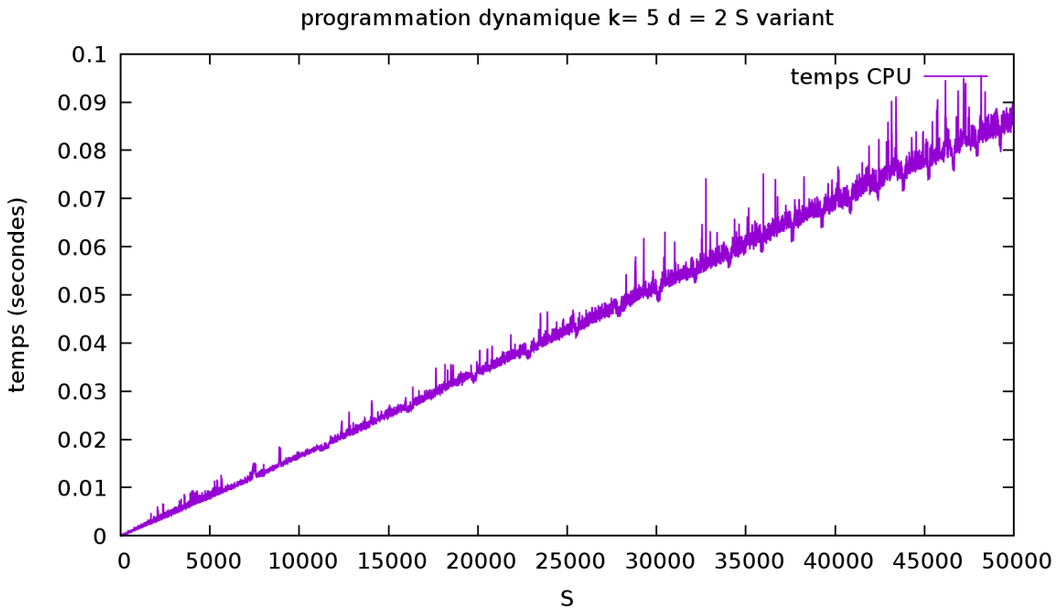
analyse : L'algorithme est exponentiel sur S, Avec un S fixé et un K variant, L'algorithme devient polynomial, mais avec une valeur S=35 ou S=40 les temps d'exécution atteignent vite des valeurs très grandes, comme on peut le constater sur les 3 courbes.

Analyse de l'algorithme AlgoProgDyn

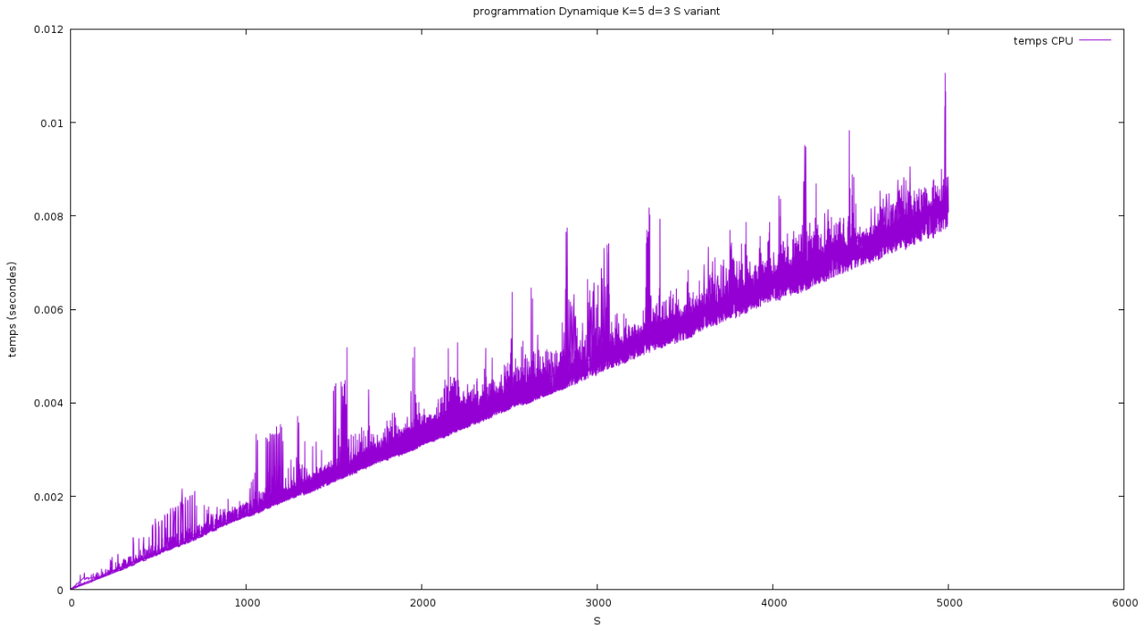
Nous avons fait varier les paramétrés S et K sur les 3 systèmes Expo.
les courbes des temps d'exécution sont présentées ci-dessous :

Temps d'exécution en variant S :

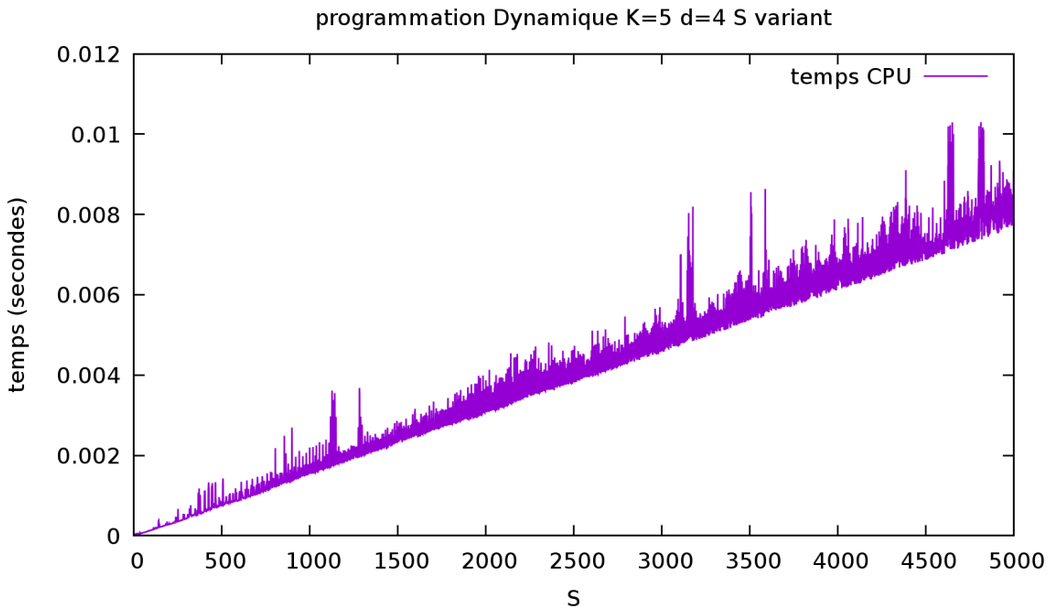
Pour d = 2 :



Pour d = 3 :



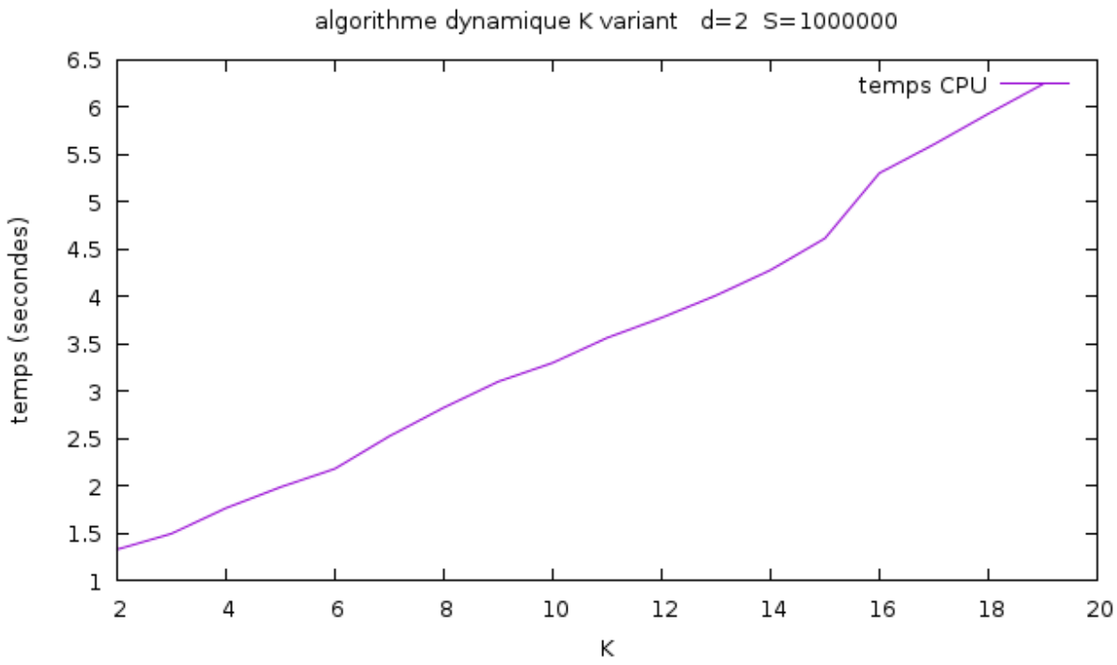
Pour d = 4 :



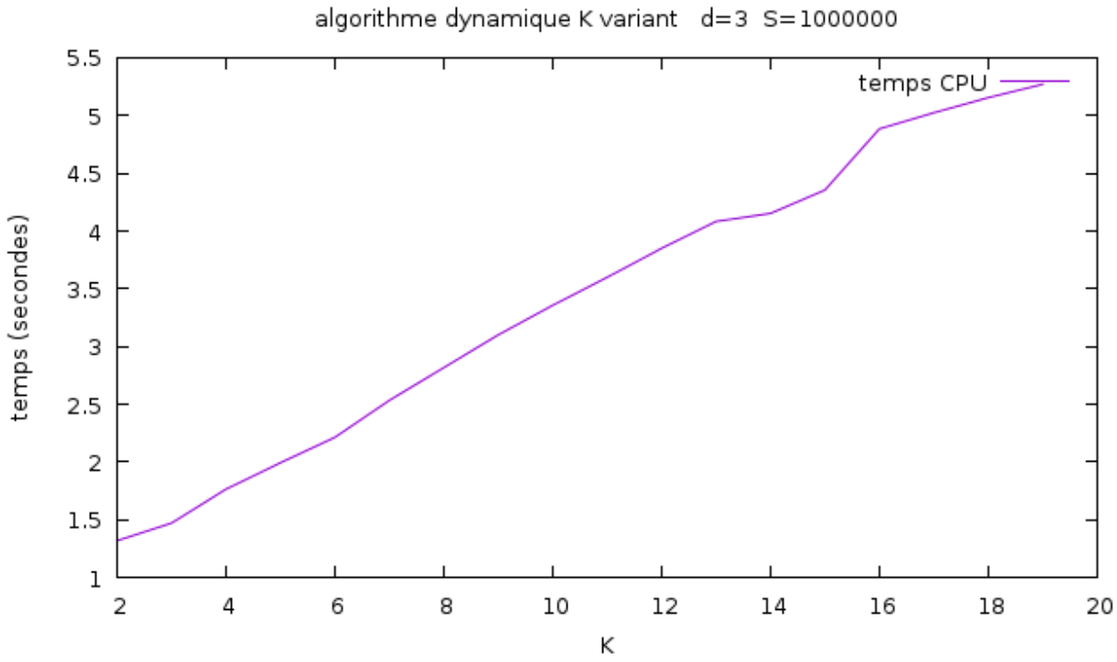
analyse : les 3 courbes suivent un tracé linéaire,ce qui vérifie l'analyse théorique de l'algorithme qui est en $O(S \times K)$.

Temps d'exécution en variant K :

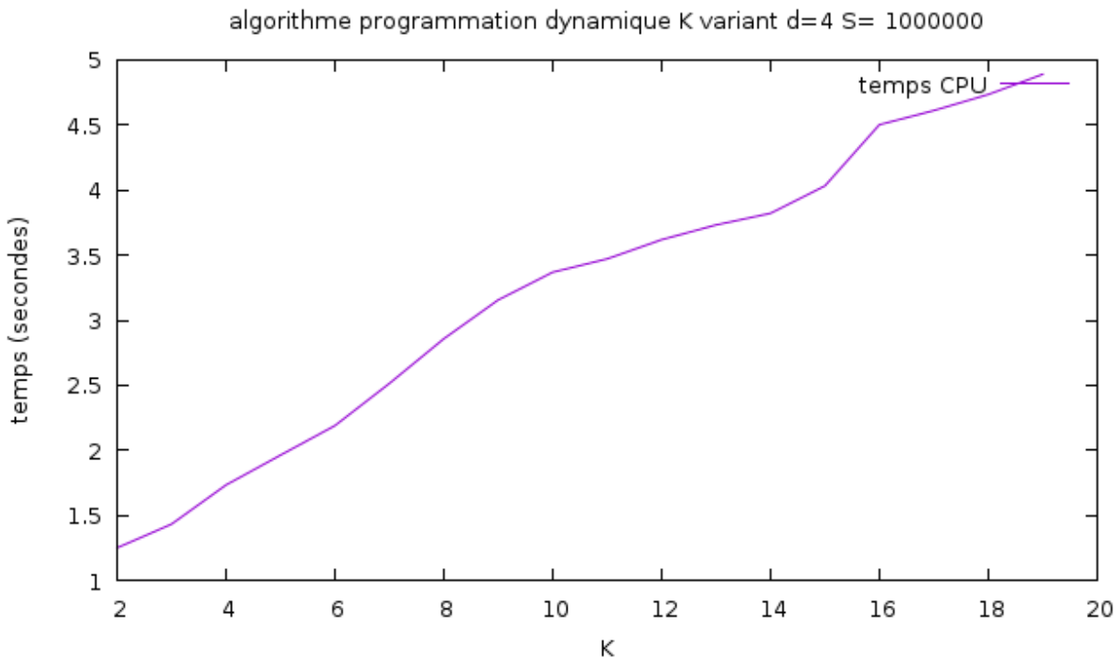
Pour d = 2 :



Pour d = 3 :



Pour d = 4 :



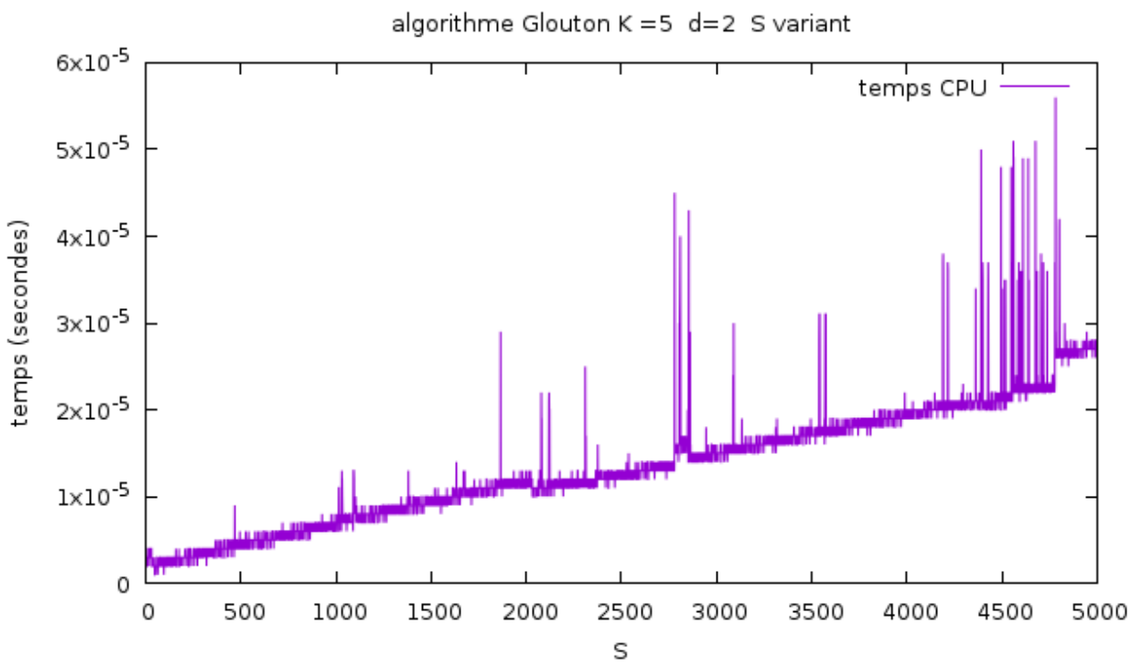
analyse : les 3 courbes suivent un tracé linéaire pour un S fixé et un k variant,ce qui vérifie l'analyse théorique de l'algorithme qui est en $O(S \times K)$.

Analyse de l'algorithme Glouton

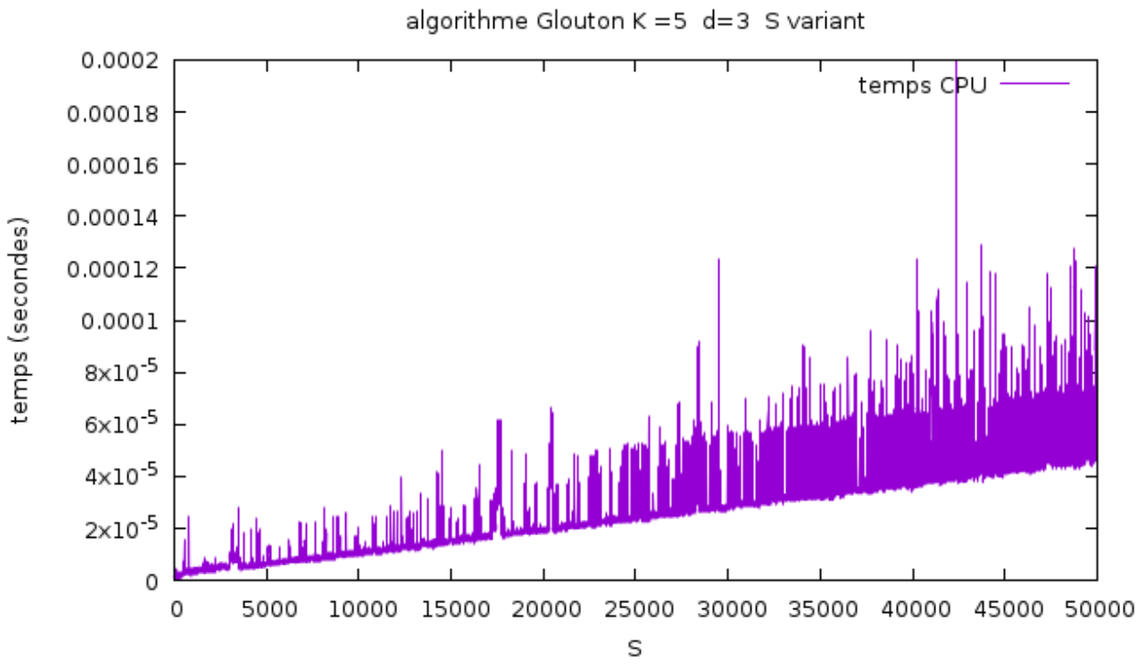
Nous avons fait varier les paramétrés S et K sur les 3 systèmes Expo.
les courbes des temps d'exécution sont présentées ci-dessous :

Temps d'exécution en variant S :

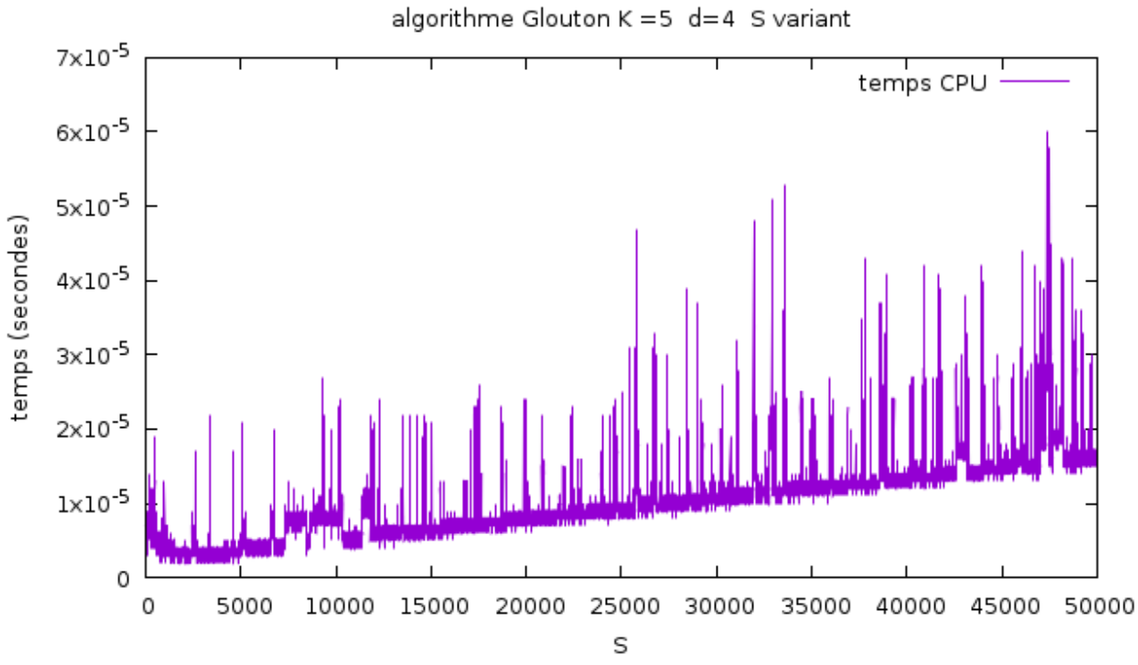
Pour d = 2 :



Pour d = 3 :



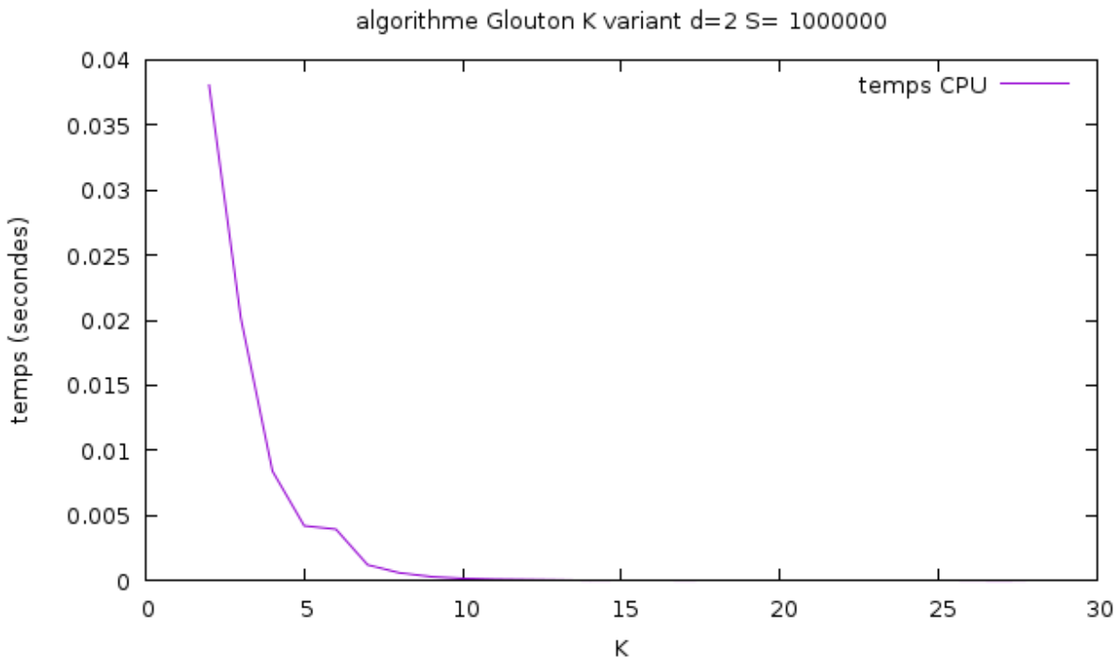
Pour d = 4 :



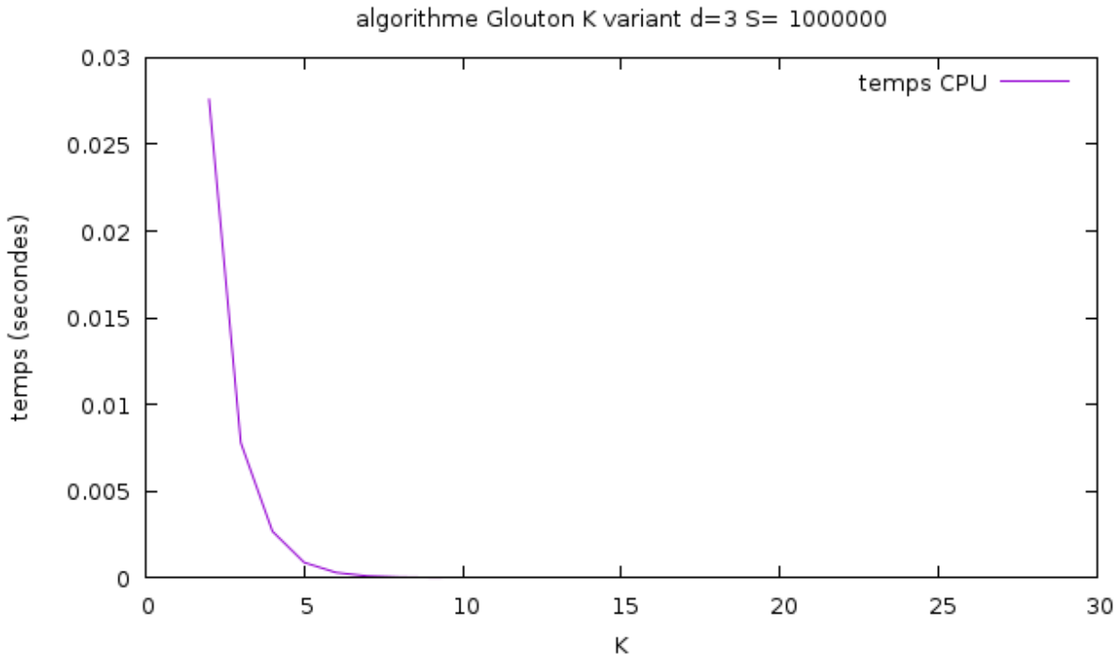
analyse : les 3 courbes suivent un tracé linéaire,ce qui vérifie l'analyse théorique de l'algorithme qui est en $O(S)$.

Temps d'exécution en variant K :

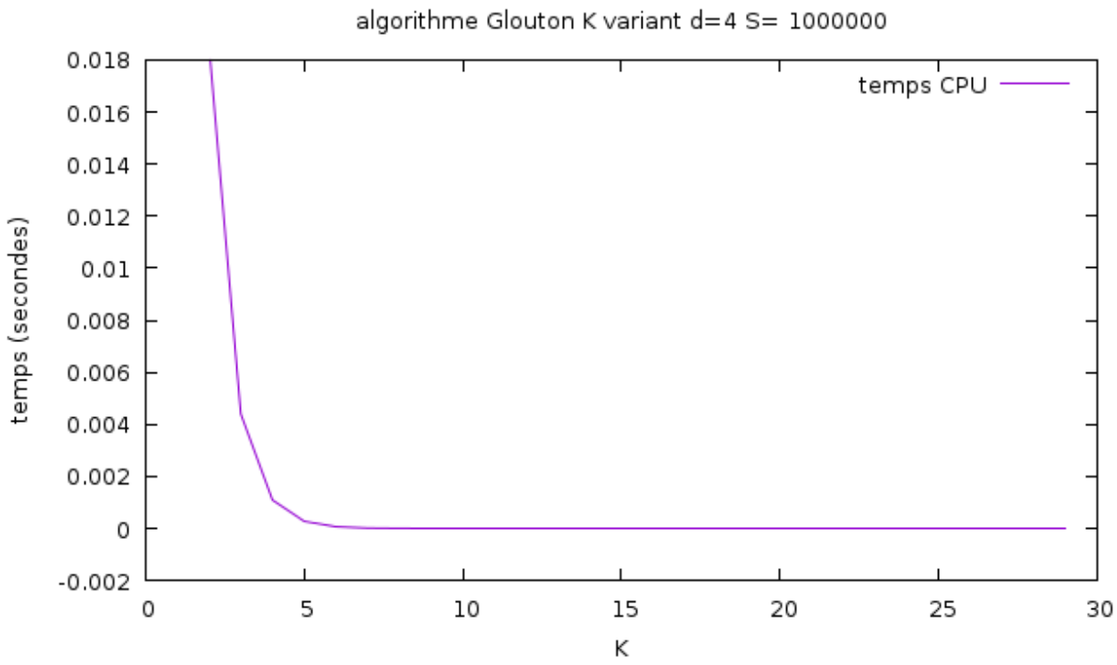
Pour d = 2 :



Pour d = 3 :



Pour d = 4 :



analyse : On constate d'abord que l'algorithme atteint sont plus grand temps d'exécution pour de petites valeurs de K, ce qui correspond au cas ou on ne prend que des bocaux de 1 (le pire cas) qui est en $O(S)$. Les valeurs diminuent ensuite pour approcher de 0 à l'infini quelque soit le K, ce qui démontre bien notre complexité de $O(S)$ qui est indépendante de K
Conclusion : Des trois algorithmes, l'algorithme glouton est le plus rapide suivit de l'algorithme Dynamique et enfin de l'algorithme Recherche Exhaustive.

2.3 Utilisation de L'algorithme Glouton

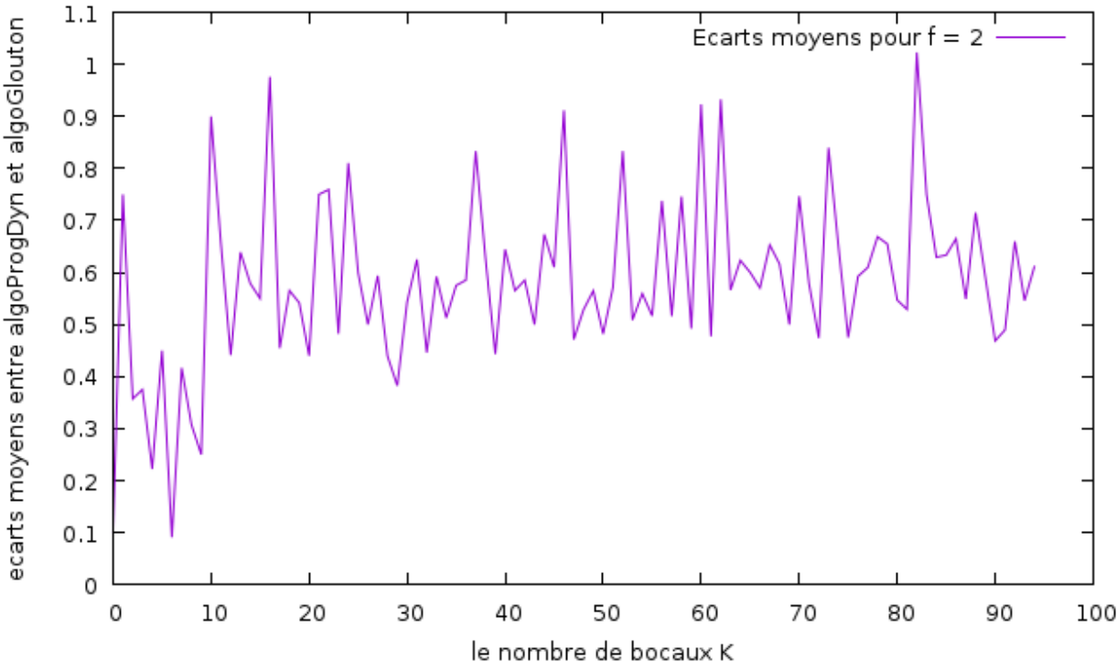
2.3.1 Question 13

Après implémentation avec python de la fonction qui génère des systèmes de capacité.
Nous avons généré $4 \times K$ systèmes pour chaque $K \in 1, \ldots, 100$ en variant le pmax.
Voici les ratios Glouton/tout les systèmes obtenus pour différentes exécutions : $4, 14\%, 3.93\%, 4.26\%$.

2.3.2 Question 14

Pour 100 valeurs de K, $pmax = 2 \times K$ et $f = 2$, voici les différentes Statistiques Obtenues :

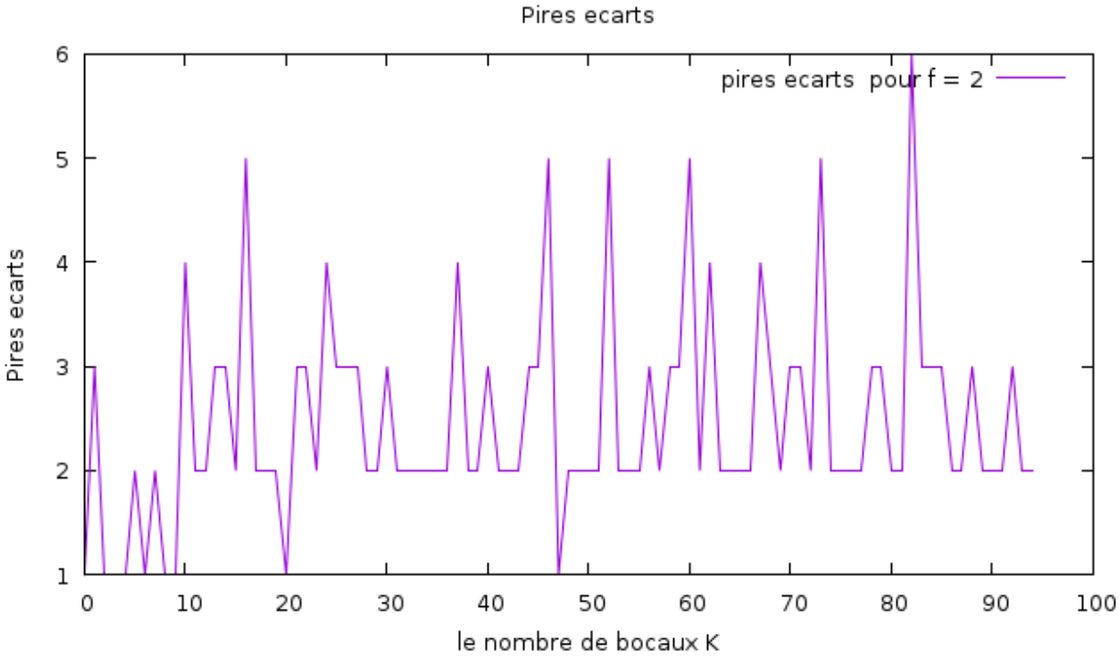
Écarts moyens :



analyse : Nous constatons que l'écart Moyen n'évolue pas avec K. et que la valeur maximale est d'environs 1 atteinte à $k = 82$

Écart moyen moyen : 0.582

Pires Écarts :



Analyse : On constate que les plus grand Écarts entre les deux Algorithmes ne sont pas dépendants de K et n'évoluent pas avec K.

le pire Écart : 6 bocaux.