Gestion d'une bibliothèque

Sommaire :

- Contexte
- Description des structures et organisation du code
- Réponses aux questions
- Méthode de validation et performance

Contexte :

Dans ce mini-projet, nous avons implémenté la gestion d'une bibliothèque. Nous avons modélisé la bibliothèque comme un ensemble de livre. Un livre est représenté par son titre, le nom de son auteur et un numéro d'enregistrement.

```
int num ;
```

```
char * titre ;
```

```
char * auteur ;
```

L'objectif de ce mini-projet est de comparer l'utilisation des tables de hachage et des listes chaînées.

Par différents algorithmes voir laquelle de ses structures est la plus efficace dans le cas de cette modélisation sur plusieurs critères : rapidité d'exécution, facilité de manipulation, complexité, terminaison.

Nous avions le fichier 'GdeBiblio.txt' fourni sur moodle afin d'évaluer nos algorithmes.

Pour simplifier la lecture et l'écriture de fichiers, nous supposons dans ce projet que les titres et les

auteurs de livres ne contiennent pas d'espaces.

Description des structures et organisations du code:

Nous avons défini 4 principales structures pour modéliser un livre et une bibliothèque dans le cas d'une table de hachage et de liste chaînée :

```c
typedef struct livre {
 int num ;
 char * titre ;
 char * auteur ;
 struct livre * suiv ;
 } Livre ;

 typedef struct { /* Tete fictive */
 Livre * L ; /* Premier element */
 } Biblio ;
```

```c
typedef struct livreh {
int clef ;
int num; /*... toutes les donnees permettant de representer un livre */
char * auteur;
char * titre;
struct livreh * suivant ;
} LivreH ;

typedef struct table {
int nE ; /*nombre d'elements contenus dans la table de hachage */
int m ; /*taille de la table de hachage */
LivreH ** T ; /*table de hachage avec resolution des collisions par chainage */
} BiblioH ;
```

Afin de factoriser le code nous sommes passées par des fichiers headers et utiliser l'outil du makefile afin de faciliter la compilation et avoir une vision globale des dépendances :

Nous avons 4 fichier.c: main, entreeSortieLC,biblioLC,biblioH

```
CFLAGS= -Wall -Werror
CC=gcc

all: main

main: entreeSortieLC.o biblioLC.o biblioH.o main.o
    $(CC) $(CFLAGS) -ggdb -o main entreeSortieLC.o biblioLC.o biblioH.o main.o -lm

entreeSortieLC.o: entreeSortieLC.c entreeSortieLC.h biblioLC.h biblioH.h
    $(CC) $(CFLAGS) -c entreeSortieLC.c

biblioLC.o: biblioLC.c biblioLC.h
    $(CC) $(CFLAGS) -c biblioLC.c

biblioH.o: biblioH.c biblioH.h
    $(CC) $(CFLAGS) -c biblioH.c

main.o: main.c entreeSortieLC.h biblioLC.h biblioH.h
    $(CC) $(CFLAGS) -c main.c

clean:
    rm -f *.o main
```

Les fonctions sont basées sur la recherche d'ouvrage dans une bibliothèque par ses différentes composantes (titre, numéro, auteur) ainsi que la libération en mémoire, l'ajout d'ouvrage.

Il y'a aussi des fonctions de fusion de bibliothèque et de recherche des livres en plusieurs exemplaire.

Toutes ces fonctions ont été programmer de manière itérative que ça soit dans le cas des tables de hachage ou des listes chainées afin d'avoir une comparaison concrète.

Les fichiers biblioLC.c et biblioLc.h sont dédiées à la représentation d'une bibliothèque par une liste chainée.

Les fichiers biblioHc et biblioH.h sont dédiées à la représentation d'une bibliothèque par table de hachage.

Les fichiers entreSortieLC.c entreeSortieLC.h sont dédiées à la lecture et l'écriture des fichiers .txt afin de récupérer les données des bibliothèques.

Le fichier main.c est dédiée à la simulation de la gestion et l'évaluation de la performance.

Nous avons aussi créé d'autre ficher .txt afin de tester nos algorithmes avec différentes bibliothèques.

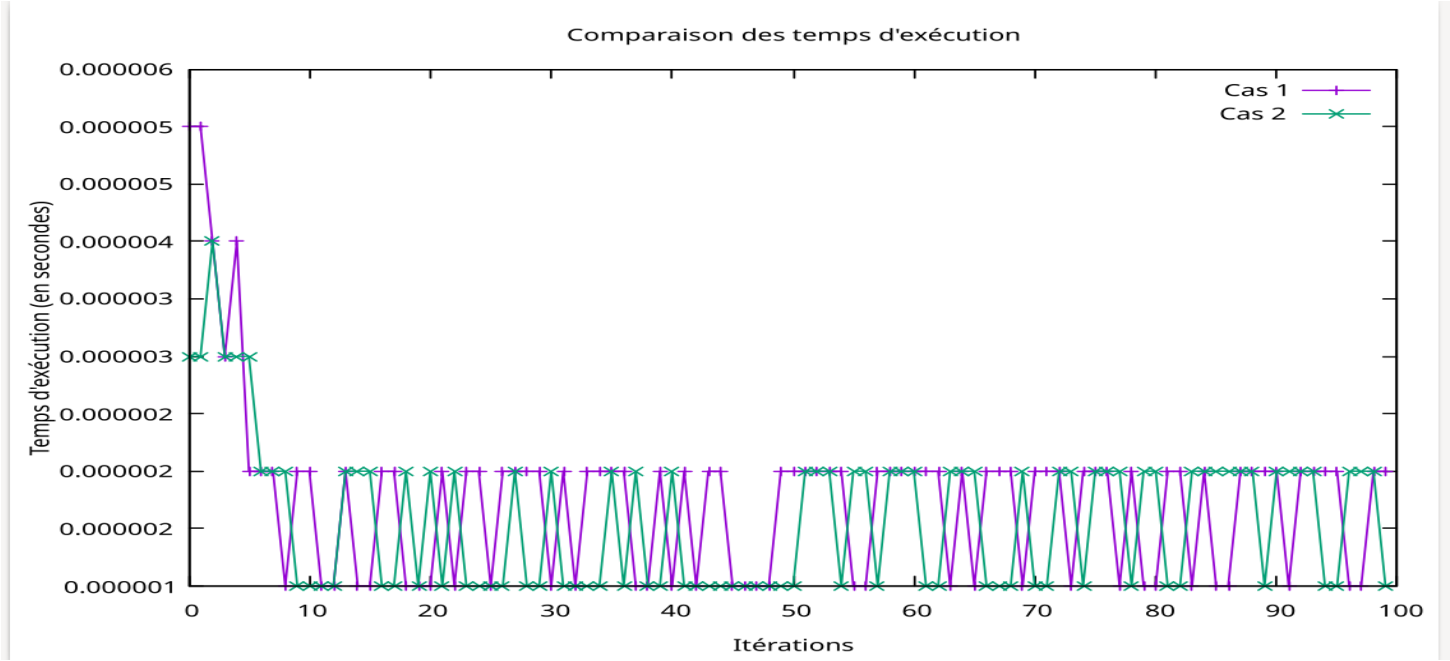
Réponses aux questions:

Question 3.1(Dans cette question la courbe verte représente les listes chainées et les courbes violette les tables de hachage)

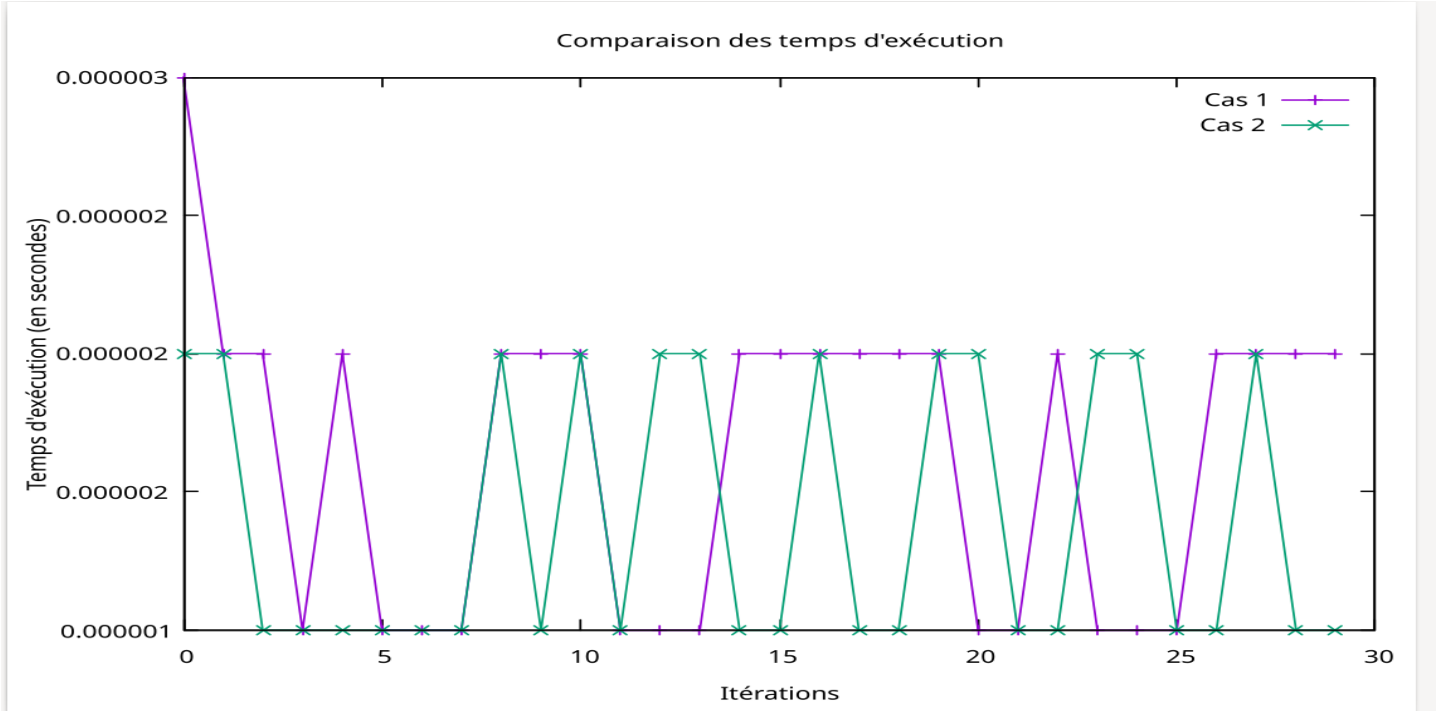
Pour la recherche par titre :

Nous avons testé avec des tables de hachage et des listes chainées de taille variant de 0 à 50, avec 30 et 100 recherches d'affilé.

Nous avons aussi évalué le score à travers des fonctions qui renvoie si la table de hachage est meilleure et 0 sinon. Par cela nous avons pu conclure que les listes chainées étaient plus efficaces pour la recherche par titre.




```
Score hachage par titre: 25 Score liste par titre: 75
```



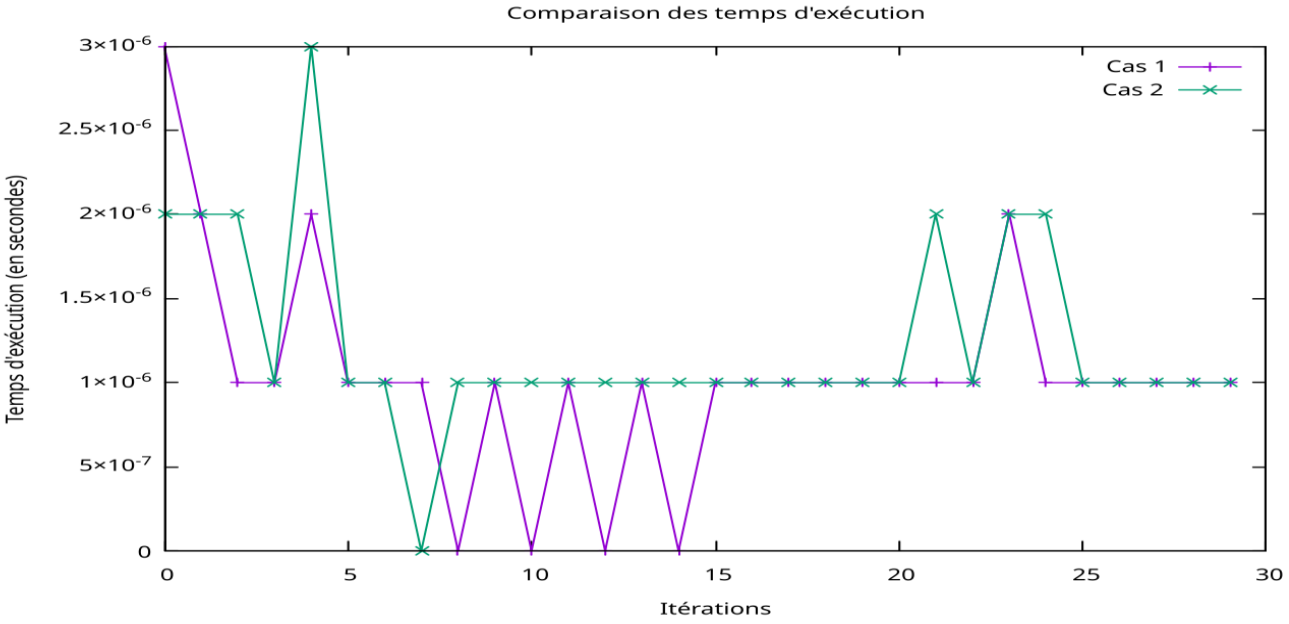
```
Score hachage par titre: 5 Score liste par titre: 25
```

On a bien le score par liste chainées qui est plus élevés que le score par hachage et que même si les graphes sont assez proches les point les plus haut sont ceux du cas 1(courbe violette) qui représente le temps d'exécution avec table de hachage. Nous avons testé avec des titres qui était présents ou non et les résultats étaient équivalent.

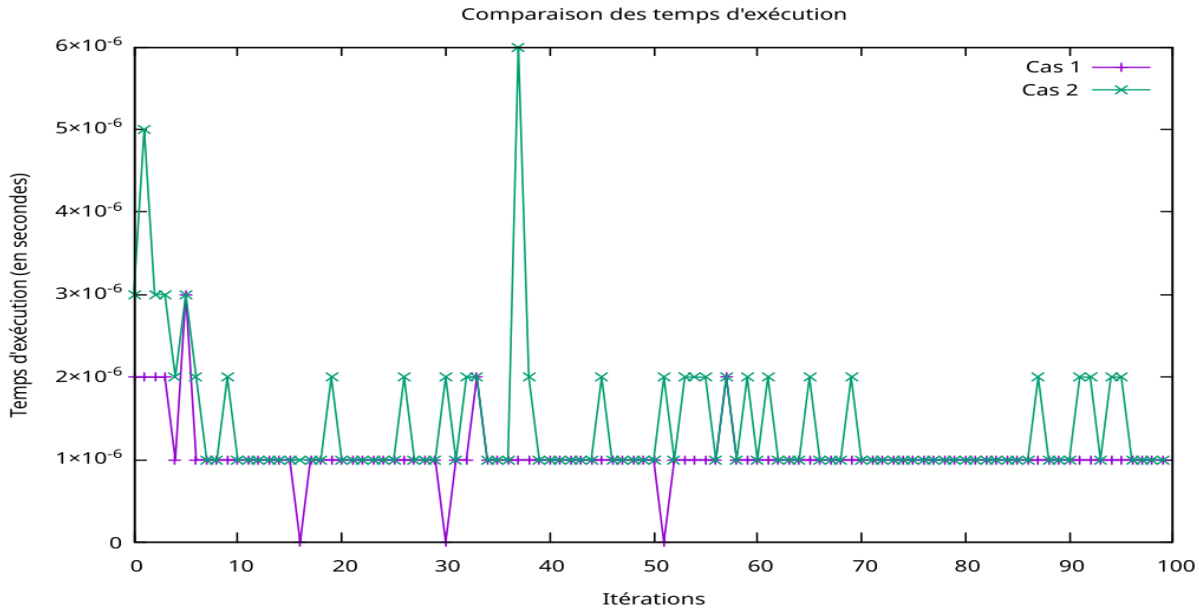
Pour la recherche par numéro:

Nous avons testé avec des tables de hachage et des listes chainées de taille variant de 0 à 50, avec 30 et 100 recherches d'affilé.

Nous avons aussi évalué le score à travers des fonctions qui renvoie si la table de hachage est meilleure et 0 sinon. Par cela nous avons pu conclure que les listes chainées étaient plus efficaces pour la recherche par numéro.




```
Score hachage par numéro : 8 Score liste par numéro: 22
```




```
Score hachage par numéro : 28 Score liste par numéro: 72
```

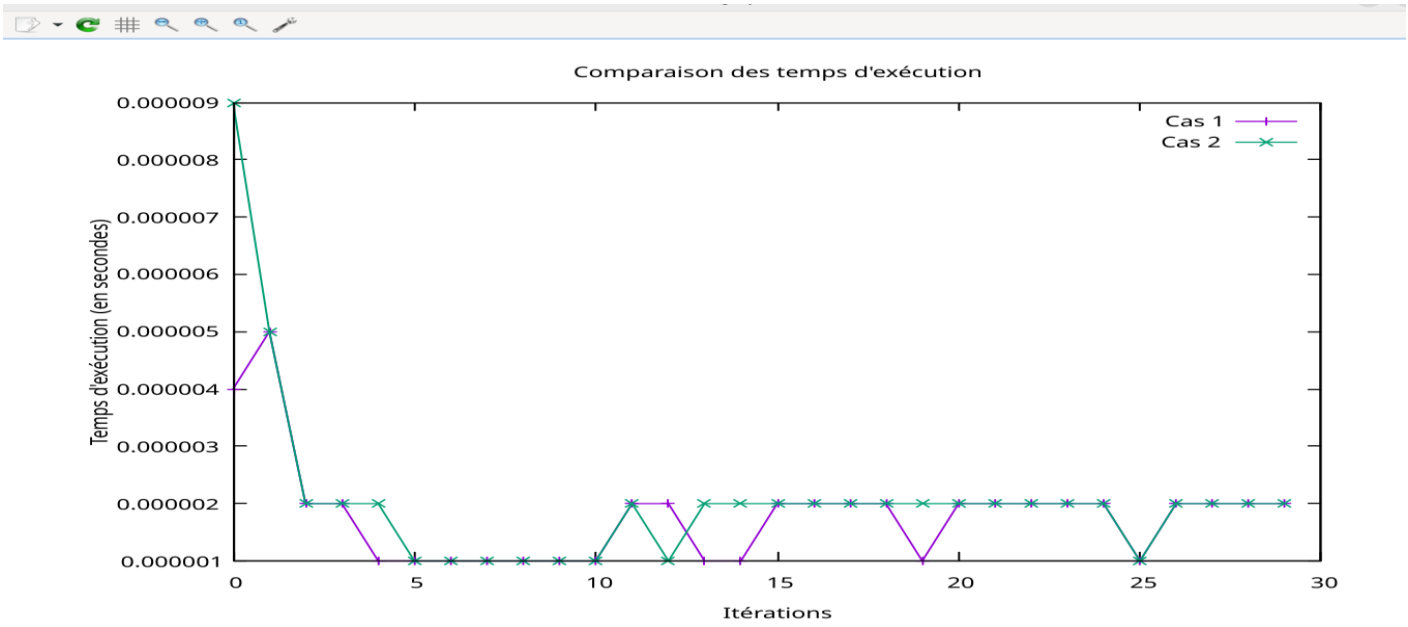
On a bien le score par liste chainées qui est plus élevés que le score par hachage, mais les courbes sont assez proches donc le score nous aide à conclure même si les piques les plus haut sont ceux avec liste chainée. (Courbe verte)

Nous avons testé avec des titres qui était présents ou non et les résultats étaient équivalent.

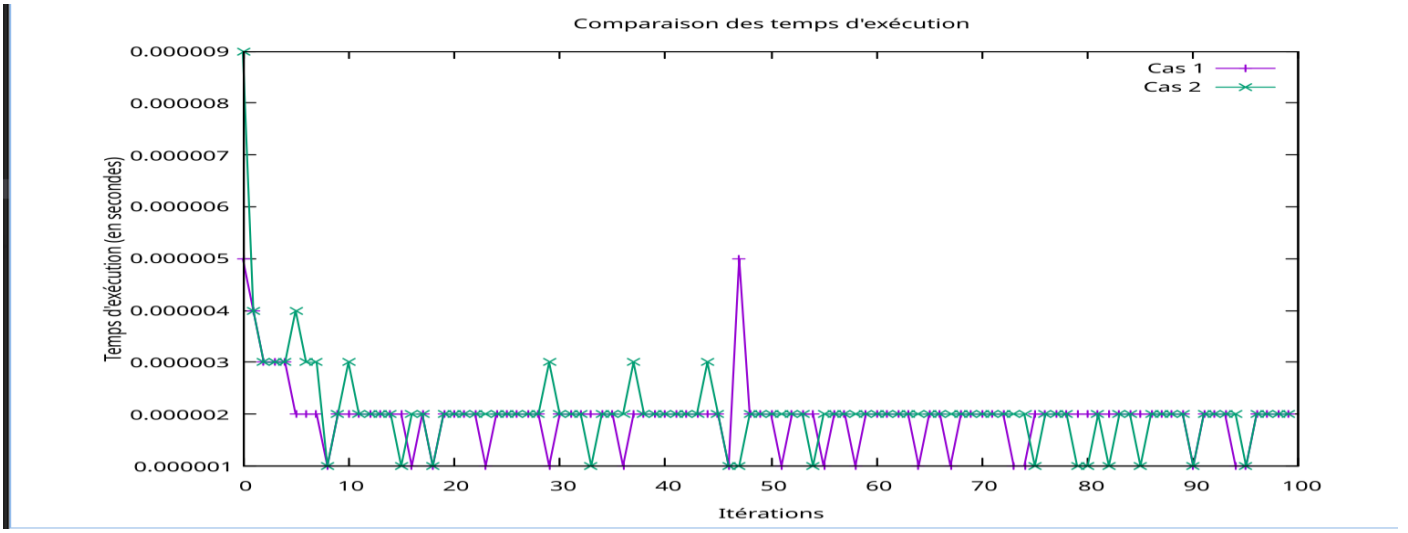
Pour la recherche par auteur :

Nous avons testé avec des tables de hachage et des listes chainées de taille variant de 0 à 50, avec 30 et 100 recherches d'affilé.

Nous avons aussi évalué le score à travers des fonctions qui renvoie si la table de hachage est meilleure et 0 sinon. Par cela nous avons pu conclure que les listes chainées étaient plus efficaces pour la recherche par auteur.




```
Score hachage par auteur: 5 Score liste par auteur: 25
```




```
Score hachage par auteur: 19 Score liste par auteur: 81
```

On a bien le score par liste chainées qui est plus élevés que le score par hachage, mais les courbes sont assez proches donc le score nous aide à conclure.

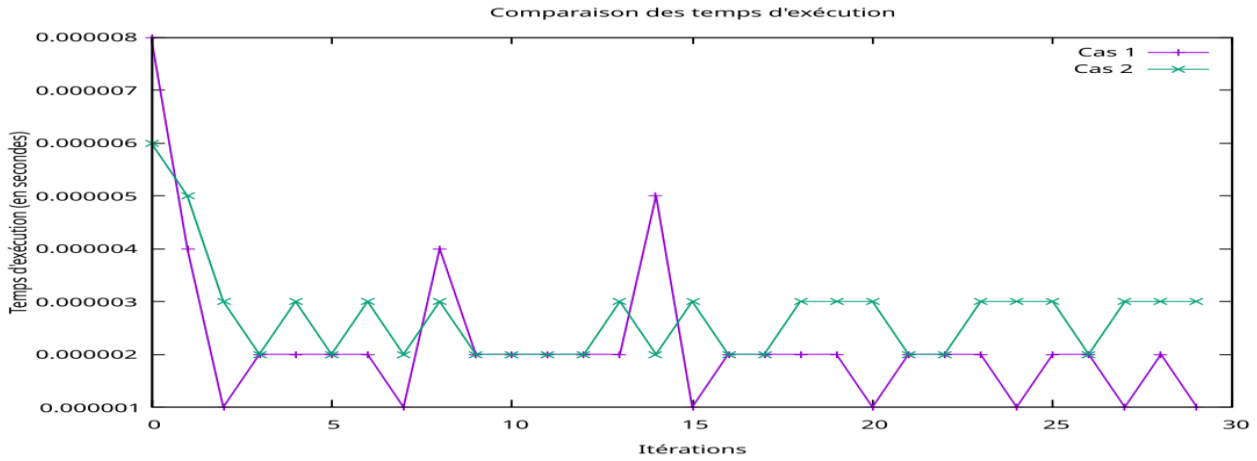
Nous avons testé avec des titres qui était présents ou non et les résultats étaient équivalent.

Question 3.2 :

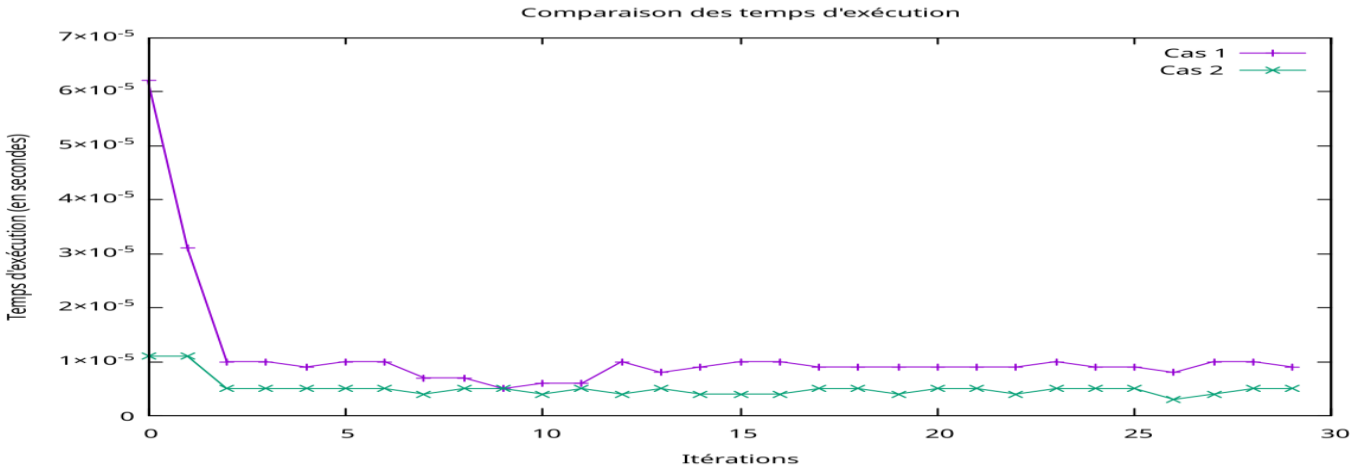
Lorsque l'on augmente la taille de la table de hachage elle devient nettement plus efficace pour la recherche par auteur et numéro mais pas pour la recherche par titre :

```
Score hachage par numéro : 16 Score liste par numéro: 14
Score hachage par auteur: 28 Score liste par auteur: 2
Score hachage par titre: 0 Score liste par titre: 30
```

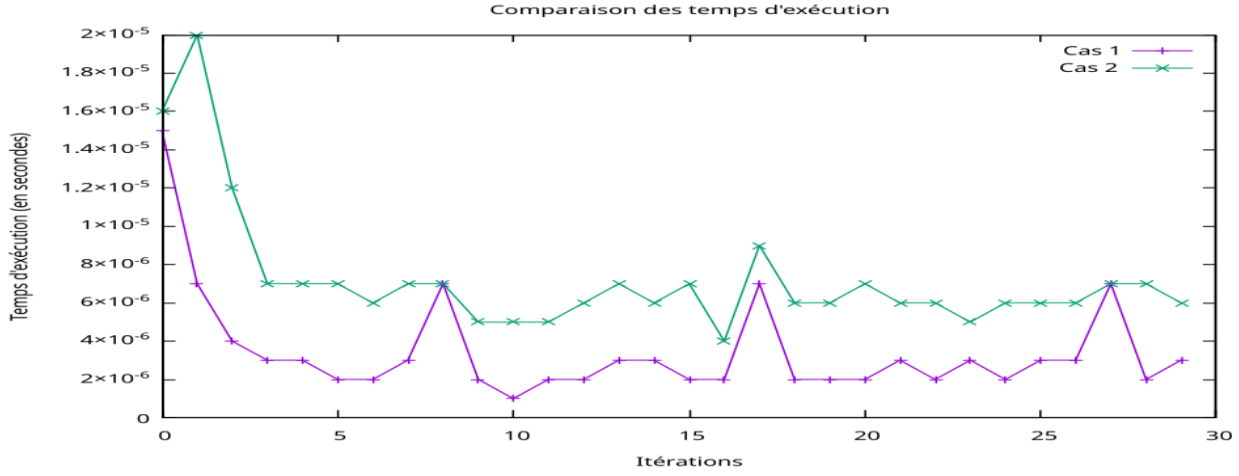
Recherche par numéro (courbe verte:LC/ courbe violette:Hachage)



Recherche par titre (courbe verte:LC/ courbe violette:Hachage)



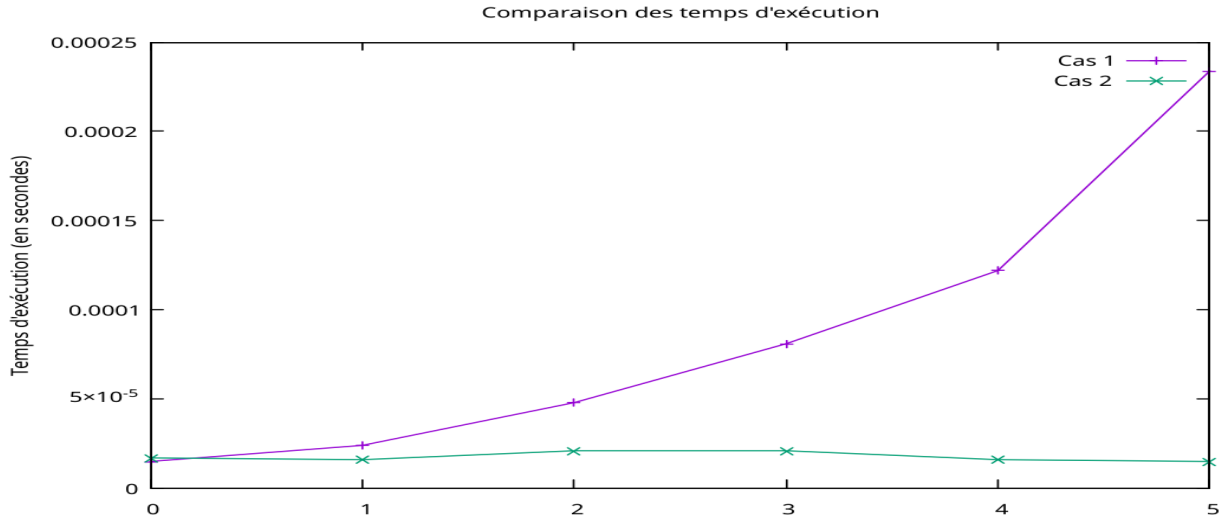
Recherche par auteur (courbe verte:LC/ courbe violette:Hachage)



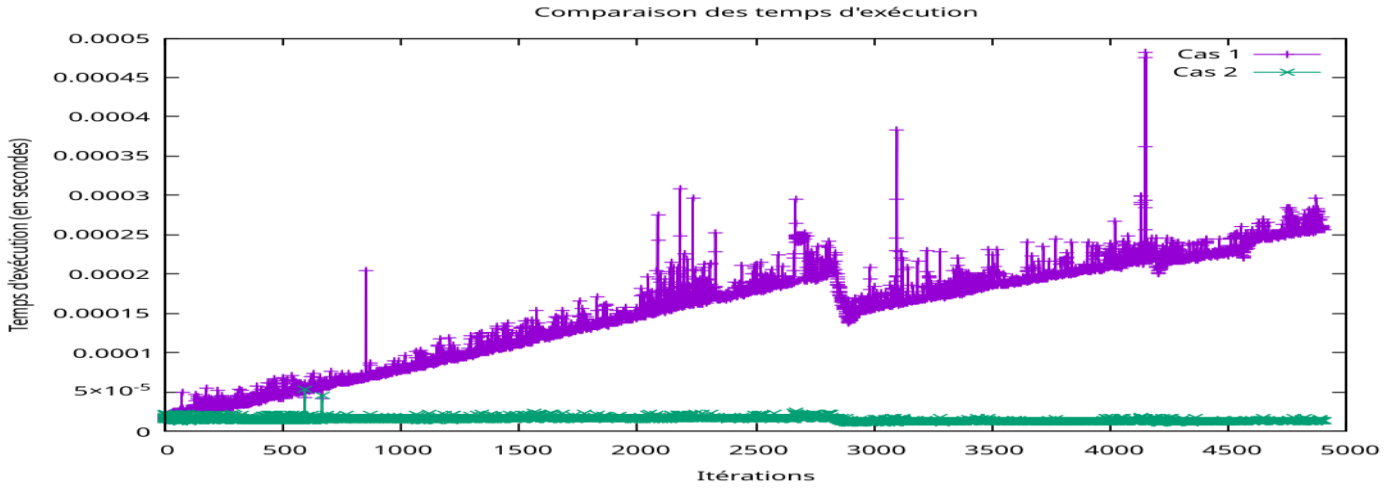
Question 3.4

Nous avons de même une fonction de comparaison pour les différentes fonctions de recherche d'exemplaire, nous avons donc tester avec différents pas pour n, et différentes tailles pour les tables et les listes.

Nous avons conclu que les listes restent plus efficaces que la table de hachage.




```
Score hachage par exmplaire: 1 Score liste par exmplaire 5
```




```
Score hachage par exmplaire: 29 Score liste par exmplaire 4871
```

Il faudrait prendre en compte aussi si la fonction de hachage est correctement distribuée pour nos données et nos éventuelles bibliothèque créés.

Méthode de validation et performance :

Pour évaluer notre programme nous avons d'abord crée d'autre fichier.txt pour représenter les diffèrent cas :

-le fichier de base GdeBiblio.txt

-le fichier bibliothèque.txt avec moins de livre

-le fichier bibliothèque1.txt qui est vide

-le fichier bibliothèquedoublons.txt qui contient plusieurs livres en plusieurs exemplaires

Pur évaluer la performance, nous avons créé 4 fonctions comparaion_auteur, comparaison_titre, comparaison_numéro,comparaion_exmplaire qui renvoie si le hachage est meilleur et 0 sinon. Ces dernières affiches aussi les temps d'exécution afin de créer des graphiques à l'aide gnuplot.Nous avons aussi implémenter un système de score pour les statistiques

La plupart de nos algorithmes sont au maximum en $O(n^2)$, donc pas plus de 2 boucles imbriqués. Nous considérons les opérations en dehors des boucles négligeable.