Rapport D’analyse

Structure De Données Avancées

Lamine LAKHDARI 11709059
Oussama Youcef TERAA 11609625

TP1

1) Définition pour la fonction potentielle dans le cas ou α >= 1 :

$\Phi i = 2 \times nomi - taillei$

2)le coût amorti de l’opération Inserer-Table en fonction de α.

cout amorti = cout reél + Φap - Φav

cas1: pas d'extension (tailleap = tailleav), (nomap = nomav+1)

$$\begin{aligned} \text{cout amorti} &= \text{cout reél} + (\alpha \times nomap - tailleap) - (\alpha \times nomav - tailleav) \\ &= 1 + \alpha \times nomap - tailleap - \alpha \times nomav + tailleav \\ &= 1 + \alpha(nomap - nomav) \\ &= 1 + \alpha \end{aligned}$$

cas2: extension (tailleap = α × tailleav), (nomap = nomav+1), (tailleav)

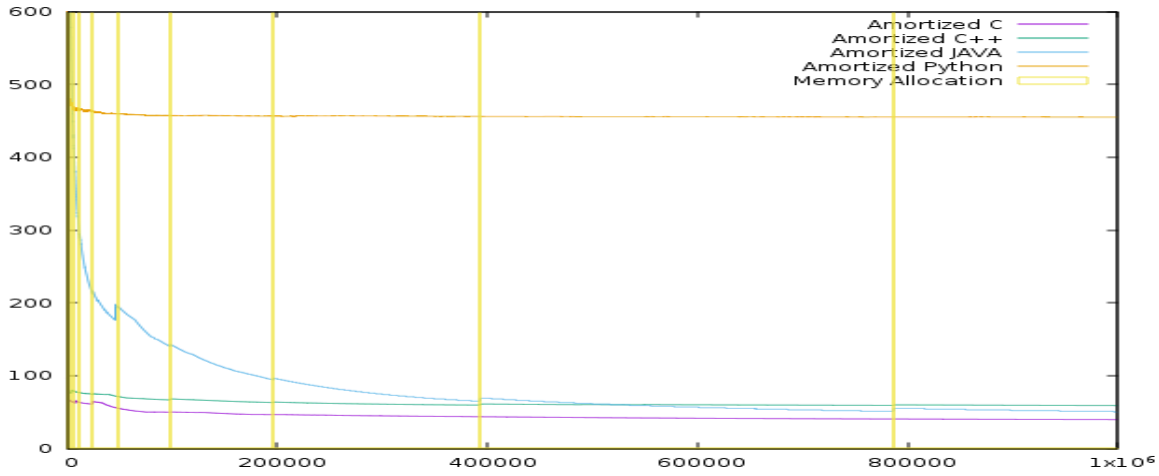
$$\begin{aligned} \text{cout amorti} &= \text{cout reél} + (\alpha \times nomap - tailleap) - (\alpha \times nomav - tailleav) \\ &= nomap + \alpha \times nomap - tailleap - \alpha \times nomav + tailleav \\ &= nomav + 1 + \alpha \times (nomav + 1) - \alpha \times tailleav - \alpha \times nomav + tailleav \\ &= nomav + 1 + \alpha - \alpha \times tailleav + tailleav \end{aligned}$$

3.a)

-Lors de l'exécution des programmes, le morceau du code qui semble prendre le plus de temps a s'exécuter est la sauvegarde des données de l'expérience.
Il est plus lent que le reste parce que les opérations sur le disque dur prennent plus de temps.

3.b)

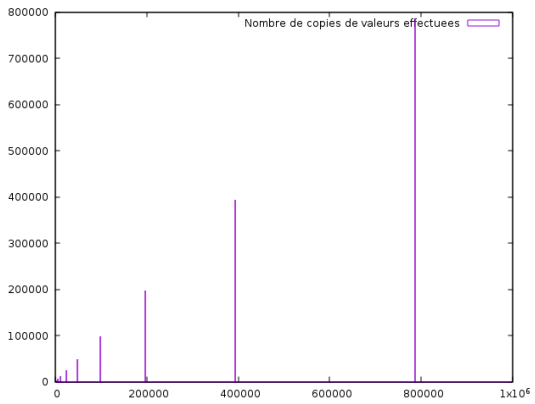
-Le coût amorti augmente lorsque de l'allocation de mémoire, parce que le coût amorti est en relation avec la taille du tableau et augmente lorsque l'augmentation de la taille du tableau (c'est bien remarquable en java).



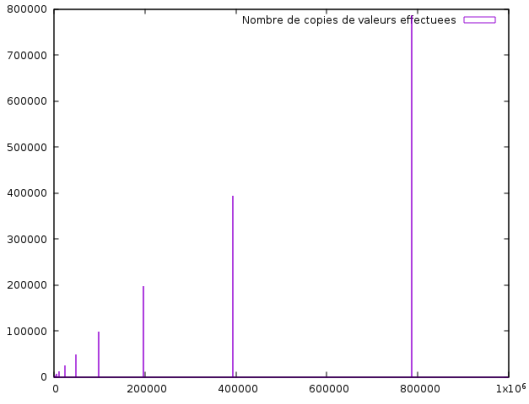
3.c)

-On remarque que lorsque l'allocation de mémoire le nombre de copies effectuées augmente, c'est du au fait que le tableau sera plein on doit procéder a une extension du tableau, on alloue la mémoire pour le nouveau tableau et on doit recopier toute les valeurs qui ils était dans l'ancien tableau dans le nouveau.

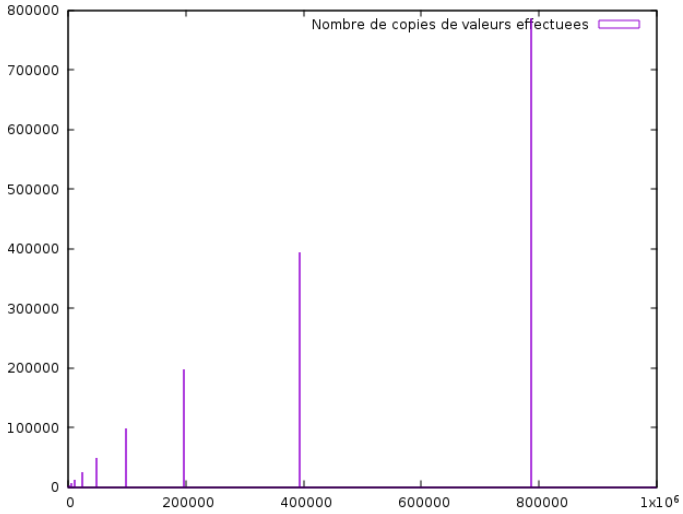
-La différence avec le temps réel mesuré l'allocation de la mémoire et le nombre de copies effectués ce passe au même moment.



C



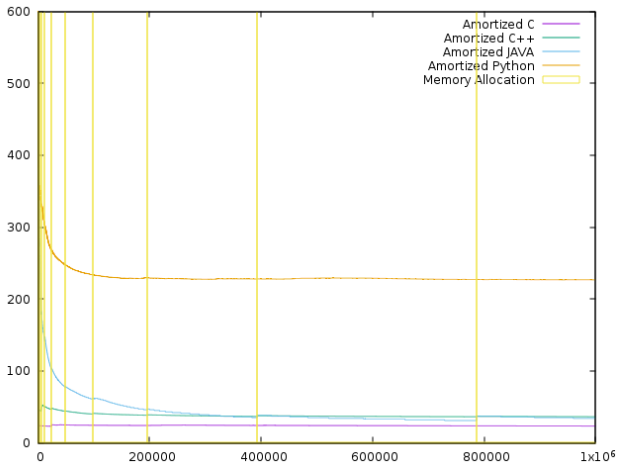
CPP

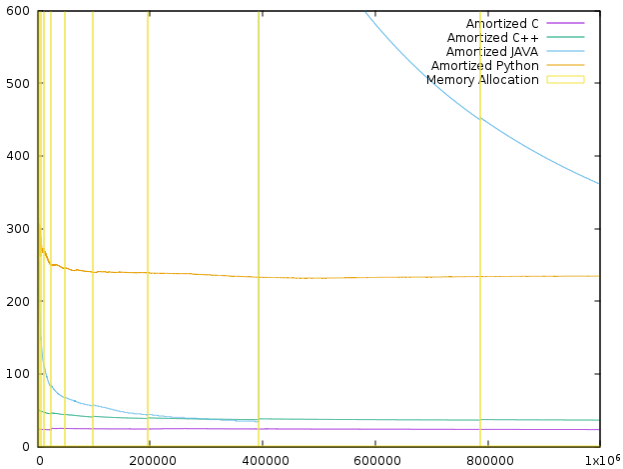


JAVA

3.d)

- Le coût amortie change d'une expérience a une autre ainsi que le total coast (le temps prix par l'opération), average coast(qui change avec le changement du total coast), la variance, et la standard déviation.
- Ce qui ne change pas c'est le nombre de copies, le moment de l'allocation de mémoire.



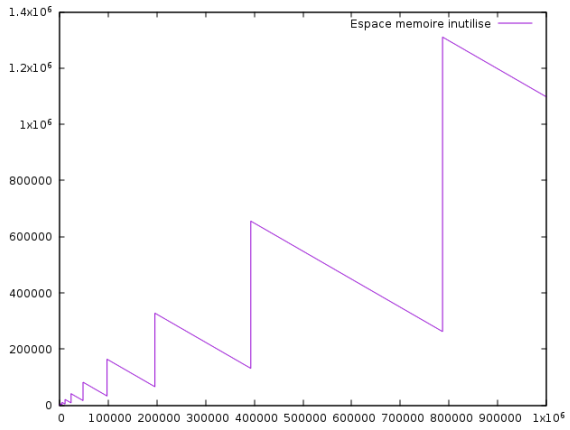


3.e)

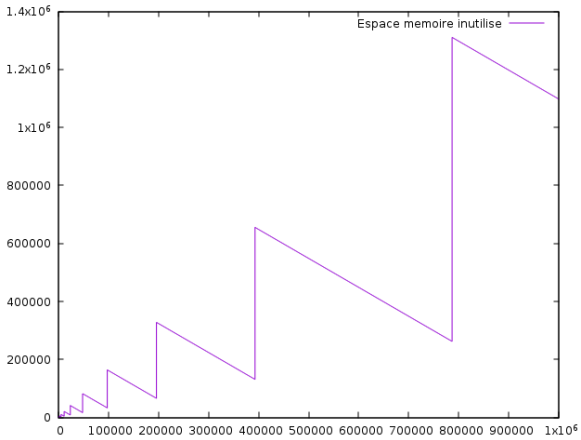
- On distingue deux deux types de langages : les langages interprétés et les langages compilés, le C et le C++ sont plus rapide parce que ce sont des langages interprétés qui sont d'abord compilé pars un logiciel qu'on appelle compilateur et c'est le système d'exploitation qui va utiliser le code binaire(résultat de la compilation) et les données d'entrée pour calculer les données de sortie. Contrairement au PYTHON et JAVA qui sont des langage interprétés ce qui veut dire qu'un interpréteur va utiliser le code source et les données d'entrée pour calculer les données de sortie.

3.f)

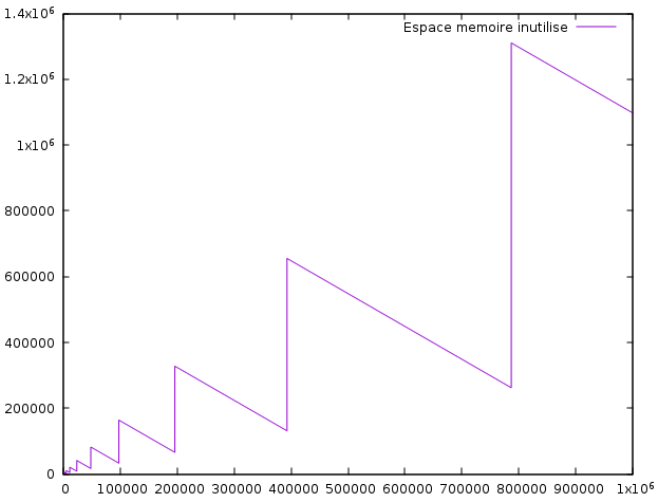
- L'espace mémoire inutilisé est le méme dans le C, JAVA ET C++, Il ne descend pas jusqu'au zéro c'est du a la condition d’extension du tableau dans la fonction do_we_need_to_enlarge_capacity() qui est si trois quart de la table est plein renvoie true donc on procede a l'extension du tableau.
- Donc l'espace mémoire inutilisé au moment de la réallocation de mémoire est de (1/4)ancien_table+ancien_table = (5/4)ancien_table.
- Le scénario dans lequel cela pourrait poser problème c'est dans le cas ou on insère un élément au tableau on fait la réallocation et on aura plus d’insertion donc la mémoire initulisé sera grande, on a aussi le problème qu'on utilise jamais la totalité de la mémoire donc on aura toujours de l'espace mémoire inutilisé.



C



CPP



JAVA

4)

Dans la figure 1, on a gauche le résultat après modification de la fonction do_we_need_to_enlarge_capacity, a droite l'ancien résultat.
- On remarque que lorsque la modification de la fonction do_we_need_to_enlarge_capacity pour ne déclencher que lorsque le tableau est plein on va économiser de l'espace mémoire, qui veux dire que l'espace mémoire inutilisé diminue par raport a la première expérience.
- L'utilisation de presque toute la mémoire alloue est aussi remarquable.
- On a aussi plus d’opération de réallocation de mémoire qu'avant.

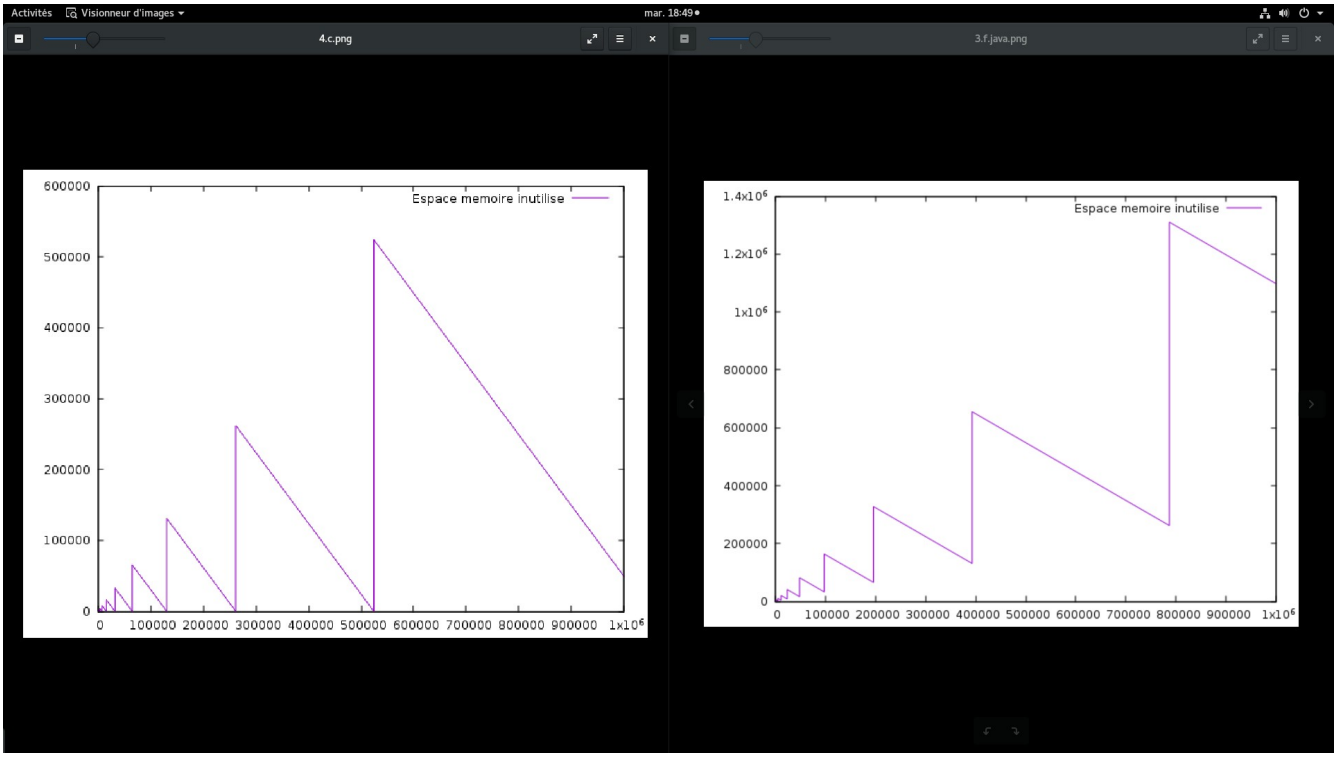
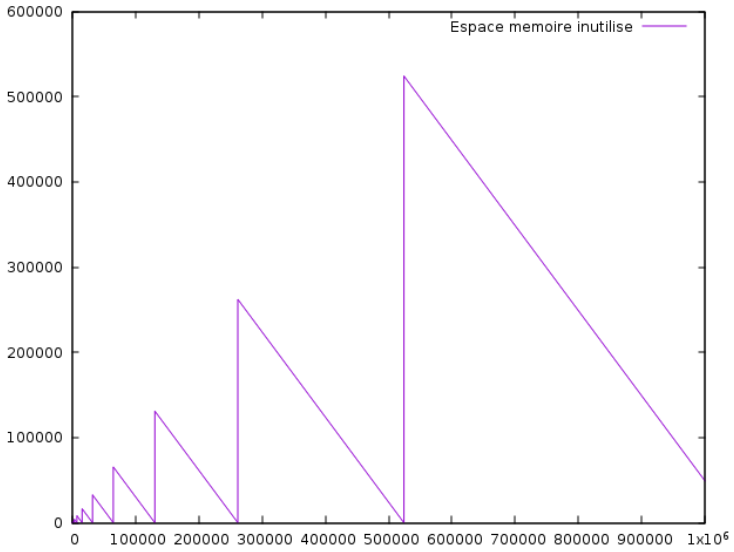
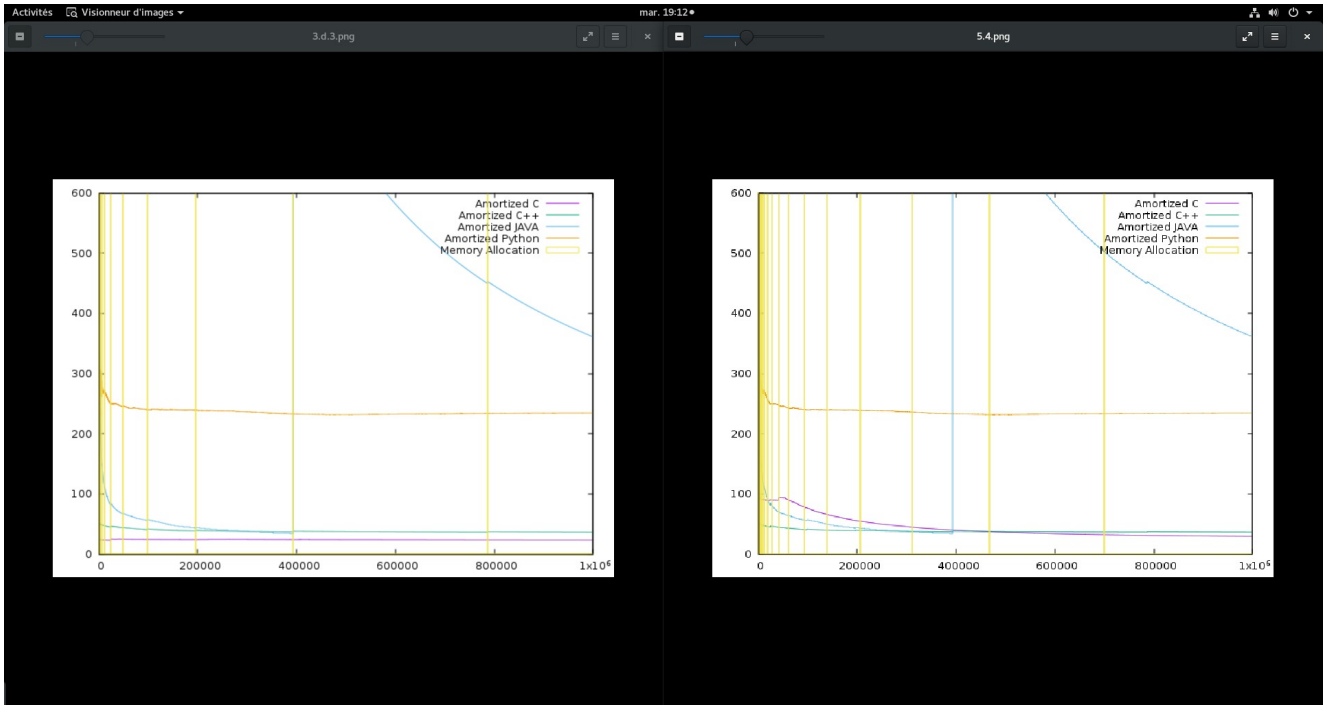


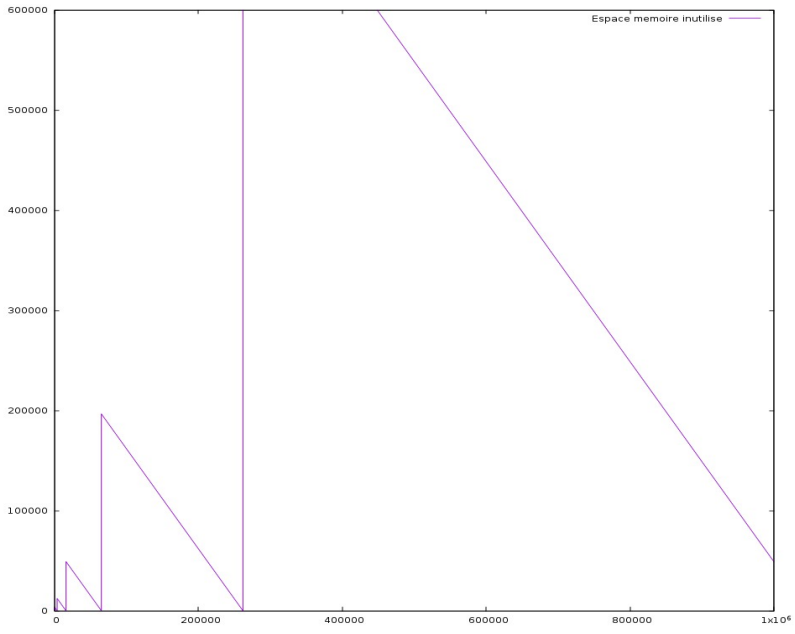
Figure-1

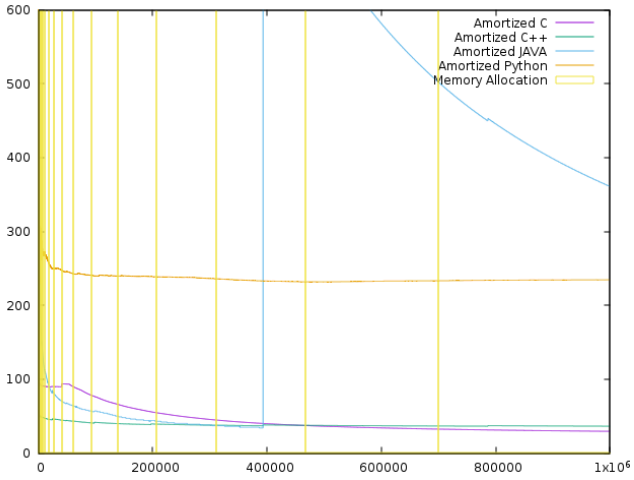


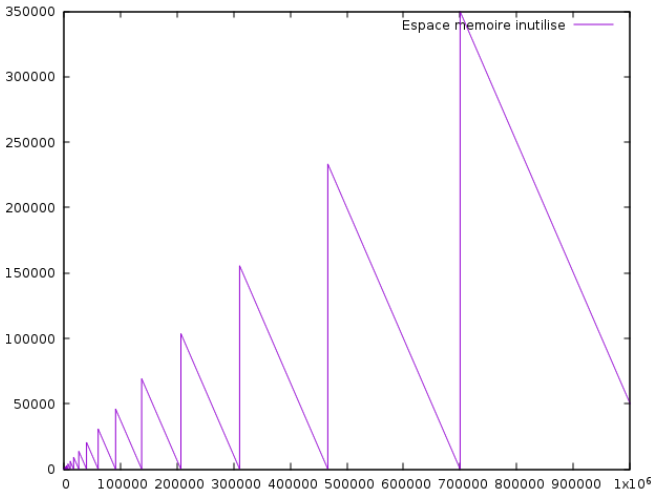
5)

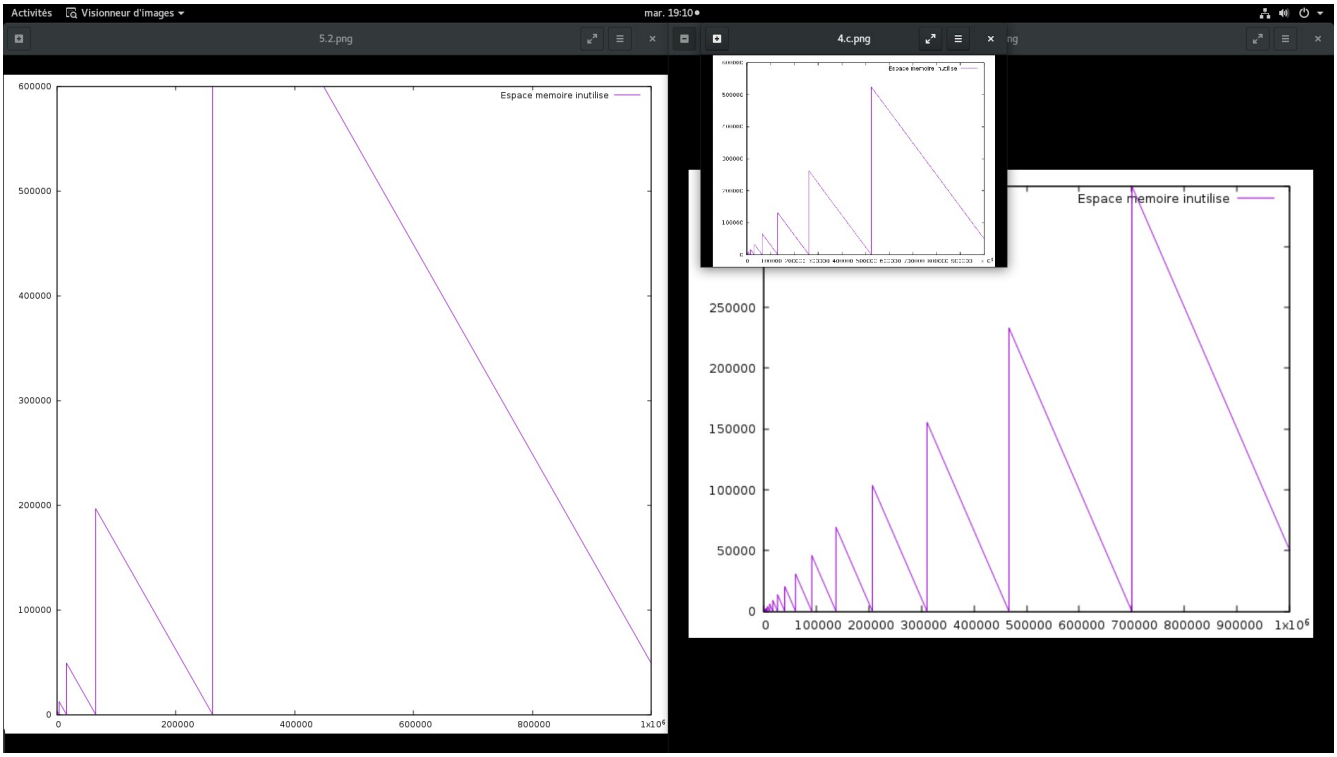
-Premièrement la réallocation de mémoire dépend aussi de la variation du facteur multiplicatif α, on augmentant ce dernier on fait moins de réallocation (ce qui est logique) et on a le contraire aussi, ce que veux dire quand on diminue ce facteur multiplicatif on fait moins de réallocation.
-L'espace mémoire inutilisé diminue aussi lorsqu'on diminue ce facteur multiplicatif l'espace mémoire inutilisé diminue mais le cout on temps augmente, le contraire est vrais aussi qui veux dire que lorsqu'on augmente le facteur multiplicatif, l'espace mémoire inutilisé augmente et le coût on temps diminue.
-le coût en temps et le coût en espace sont on corrélation inverse.





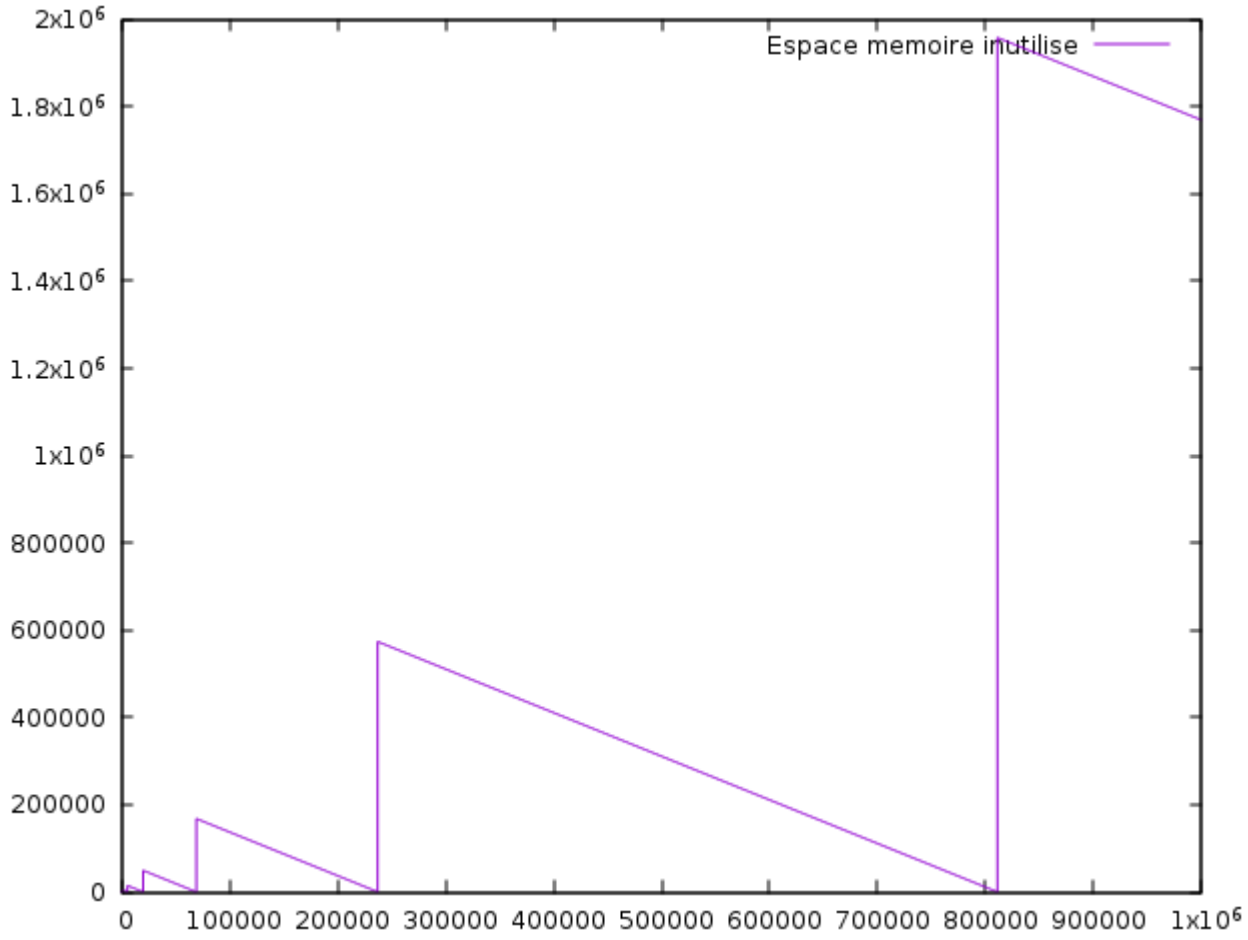

Amortized C
Amortized C++
Amortized JAVA
Amortized Python
Memory Allocation
600
500
400
300
200
100
0
0
200000
400000
600000
800000
1x10⁶


Espace memoire inutilise
350000
300000
250000
200000
150000
100000
50000
0
0
100000
200000
300000
400000
500000
600000
700000
800000
900000
1x10⁶


Activités
Visionneur d'images
mar. 19:10
5.2.png
4.c.png
Espace memoire inutilise
600000
500000
400000
300000
200000
100000
0
0
200000
400000
600000
800000
1x10⁶
Espace memoire inutilise
250000
200000
150000
100000
50000
0
0
100000
200000
300000
400000
500000
600000
700000
800000
900000
1x10⁶

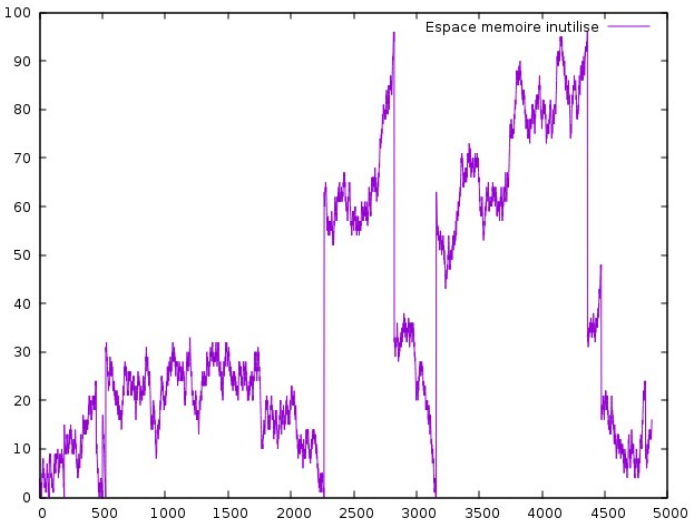
6)

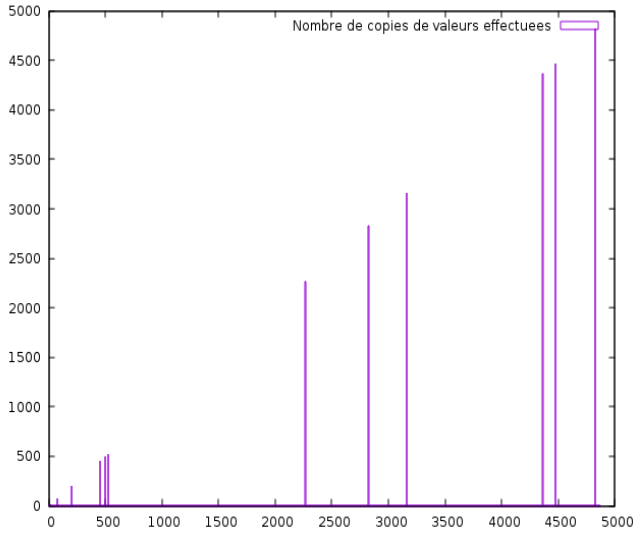
-Lorsqu'on varie la capacité n vers une capacité n+sqrt(n) on remarque que l'espace mémoire lorsque de la réallocation mémoire, l'espace mémoire alloue s'utilise moins rapidement que lorsqu’on a une capacité n.

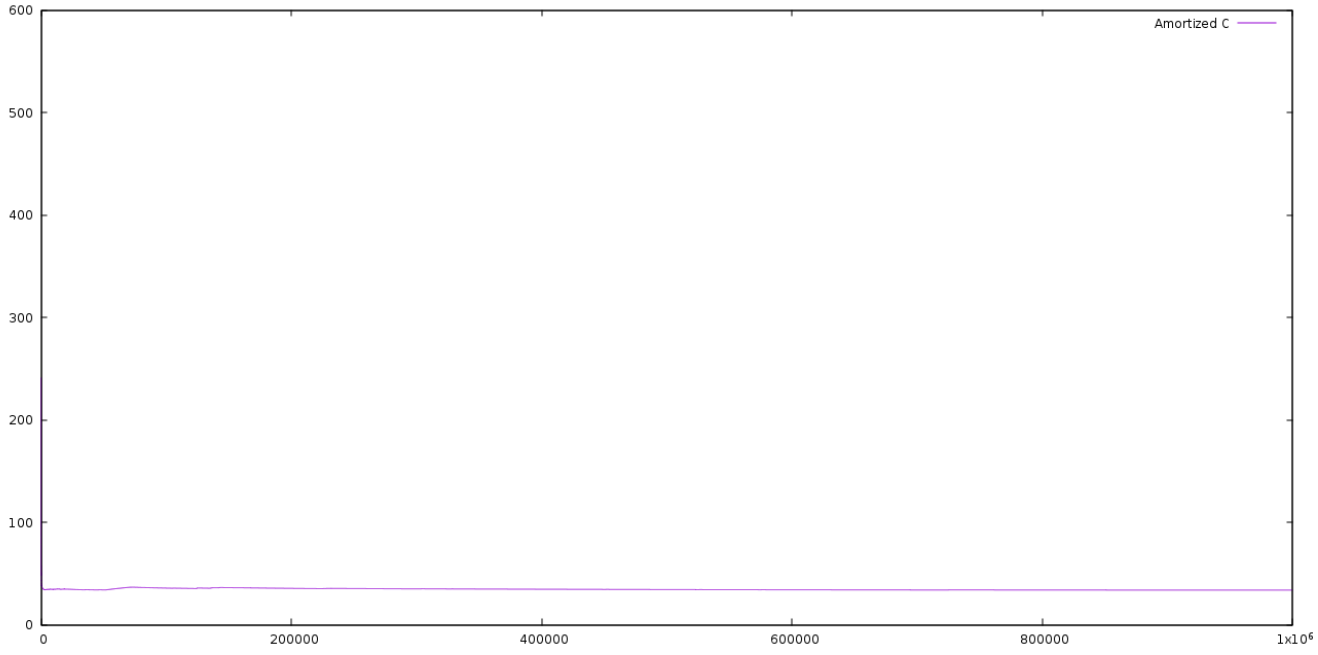


TP2

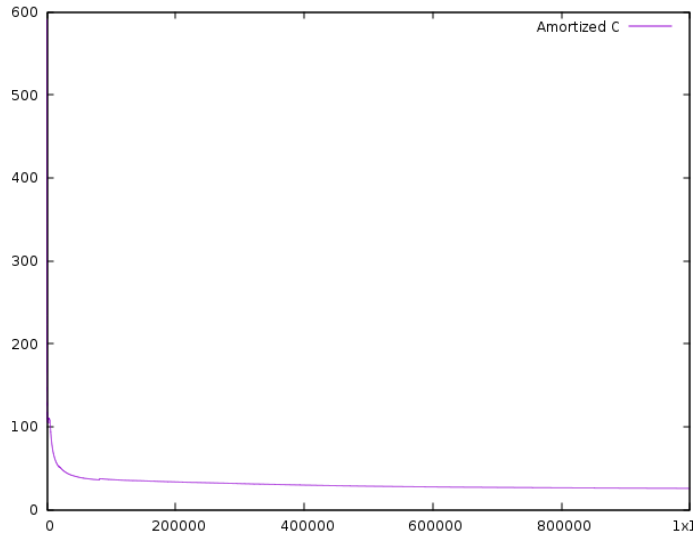
-après avoir changé le code pour effectuer un mélange de n opérations d'insertion si la probabilité est supérieur à 0,5 et une suppression sinon on a obtenu le coût amortie, le coût réel et l'espace mémoire non-utilisé ci-dessous :

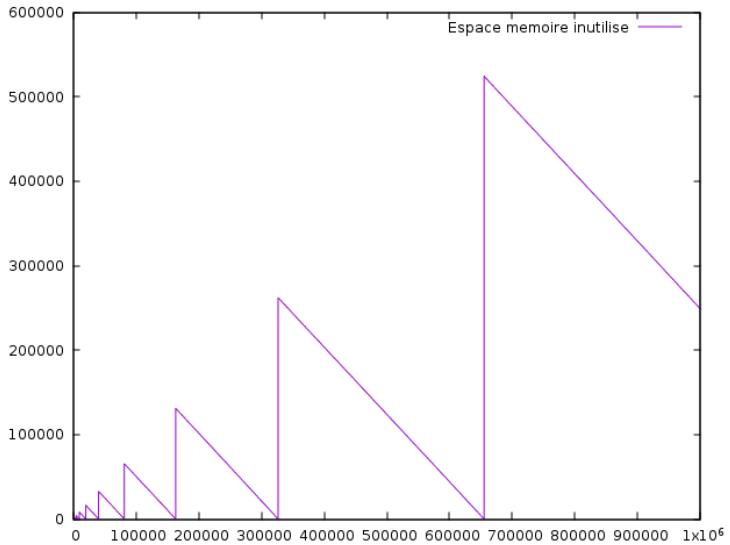


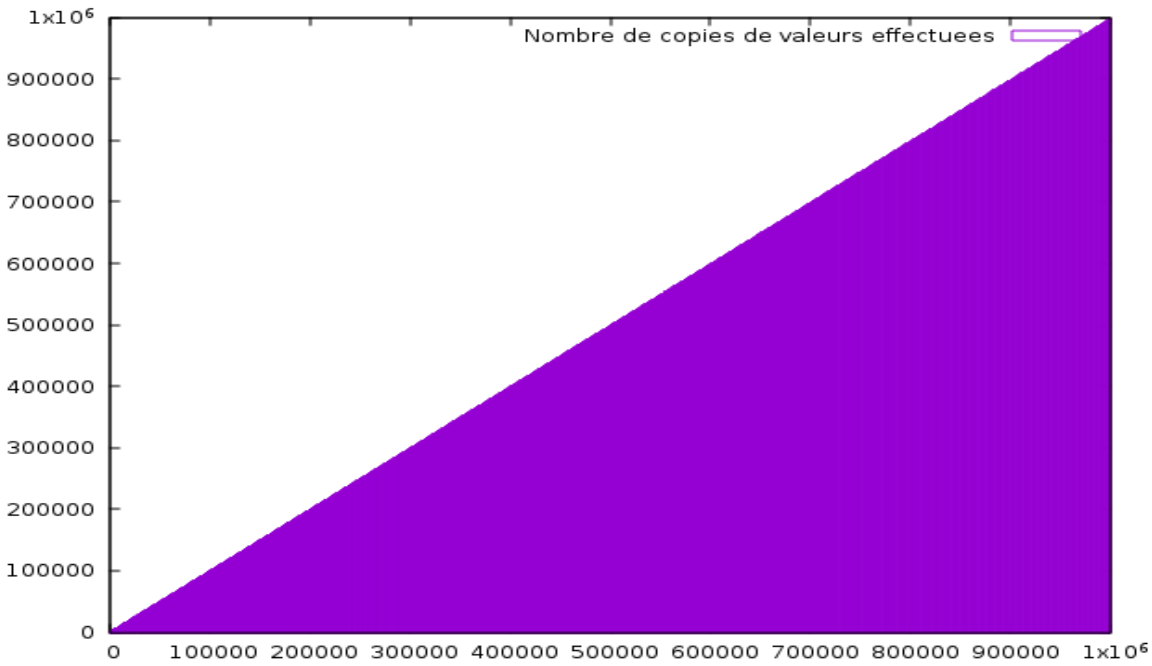




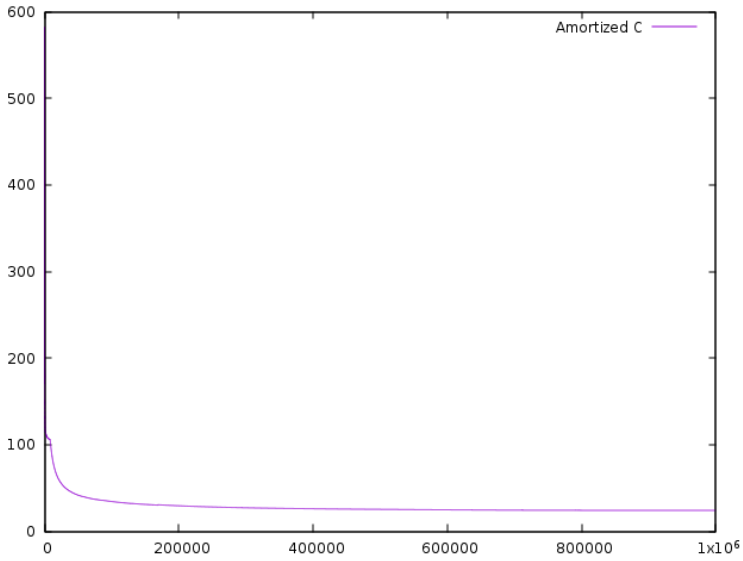
-maintenant on va tester différentes valeurs de P et on va enregistrer les coûts amortis et réel ainsi l'espace mémoire non-utilisé :
pour P= 0,1

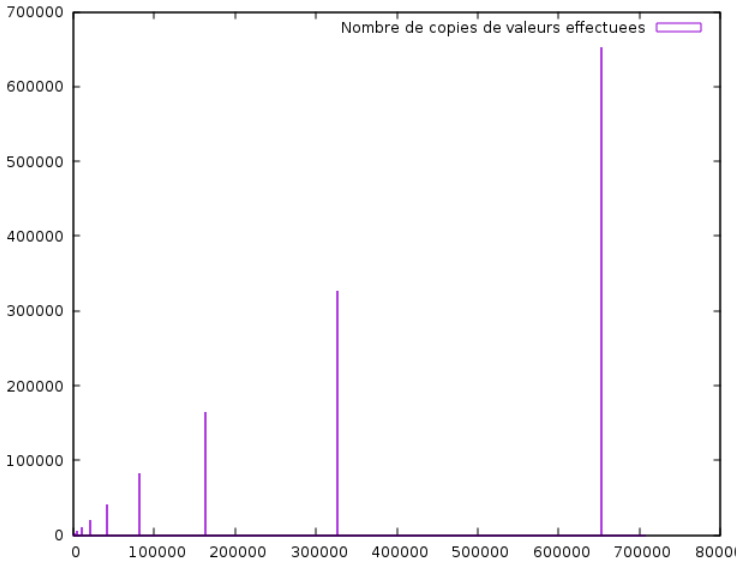


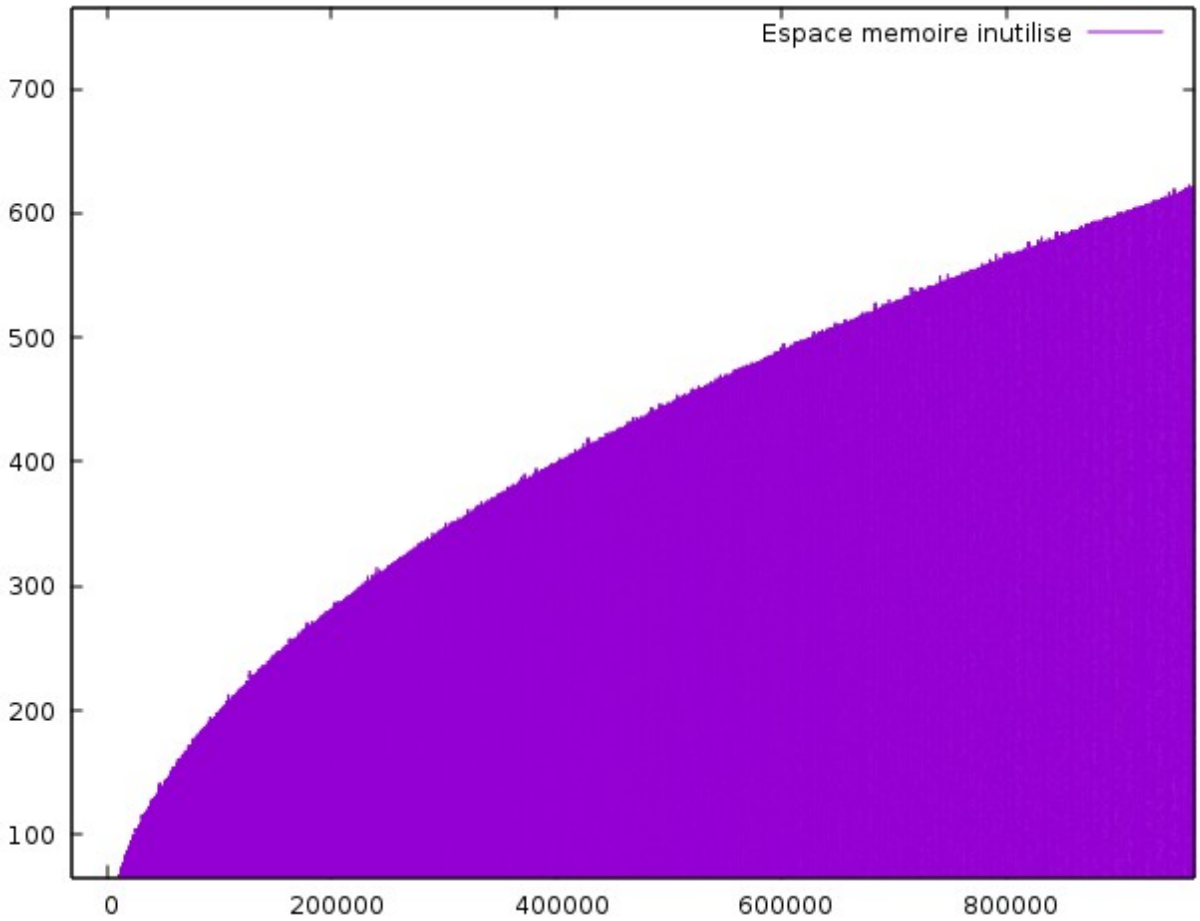




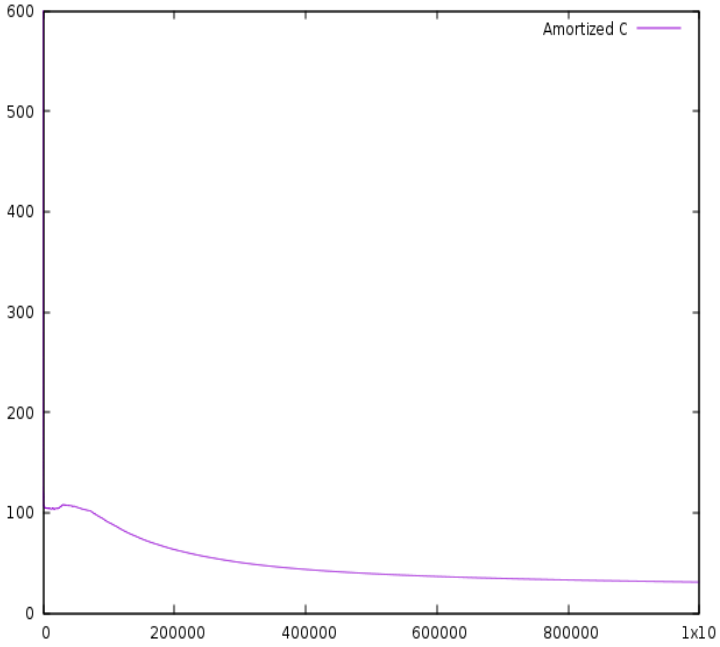
pour P=0,3

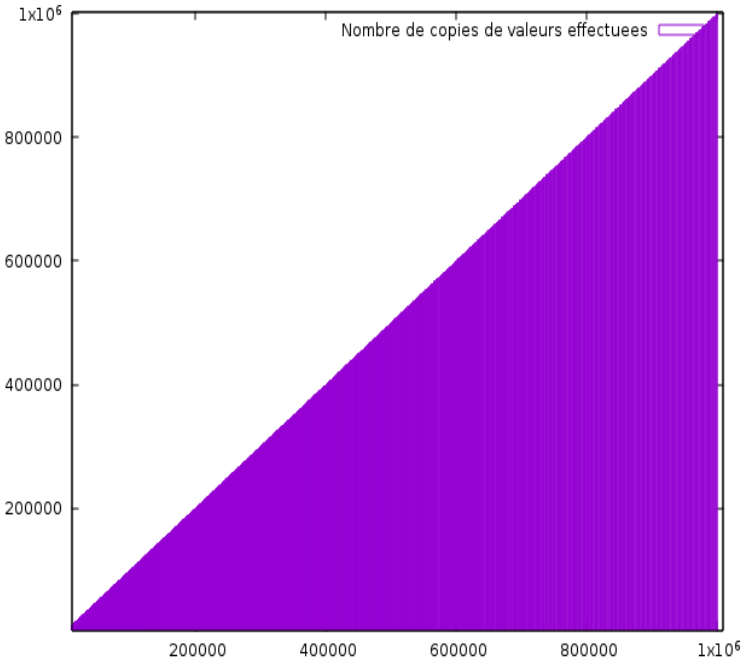

Amortized C
600
500
400
300
200
100
0
0
200000
400000
600000
800000
1x10^6

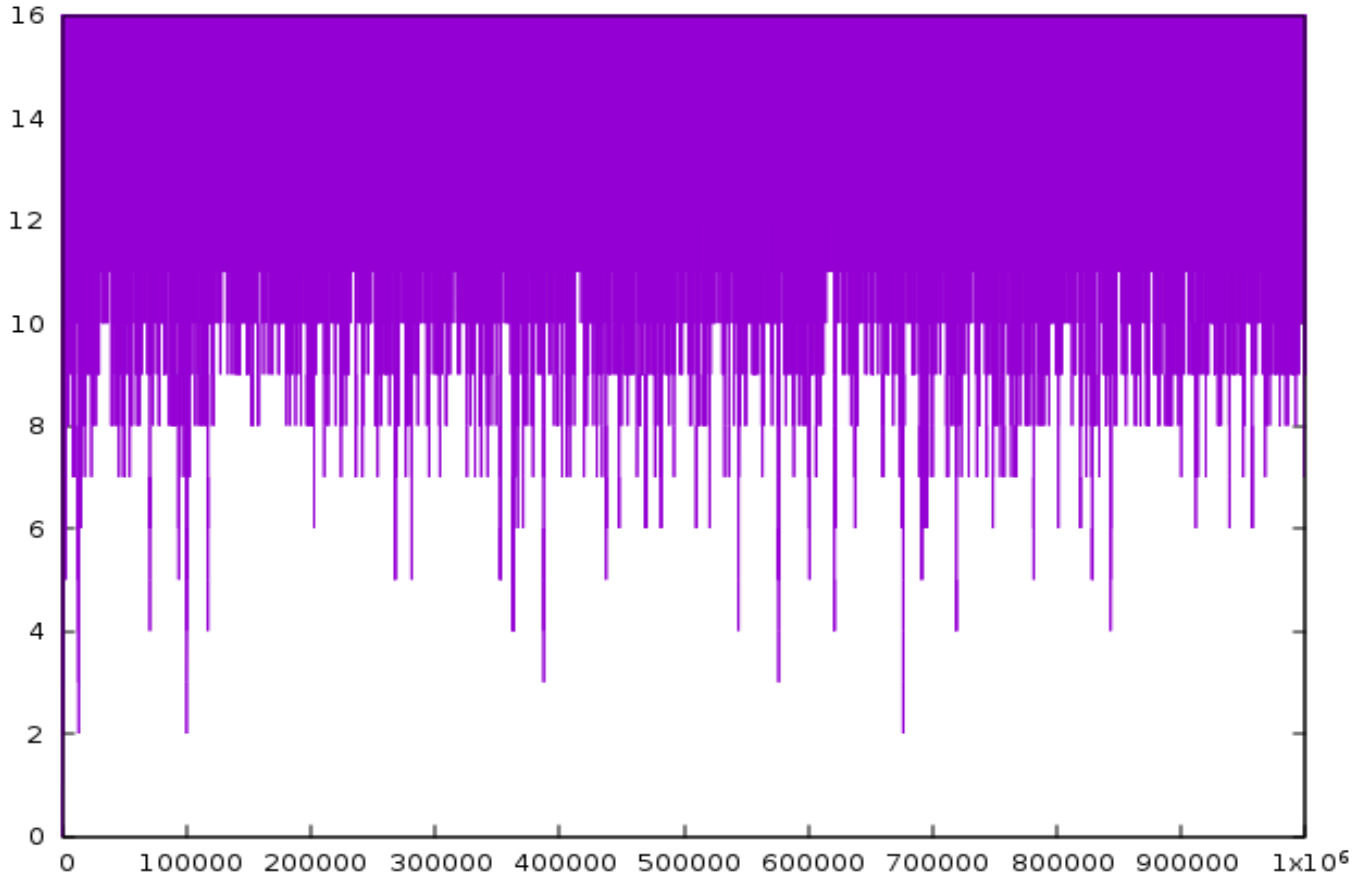

Nombre de copies de valeurs effectuees
700000
600000
500000
400000
300000
200000
100000
0
0
100000
200000
300000
400000
500000
600000
700000
800000


Espace memoire inutilise
700
600
500
400
300
200
100
0
200000
400000
600000
800000

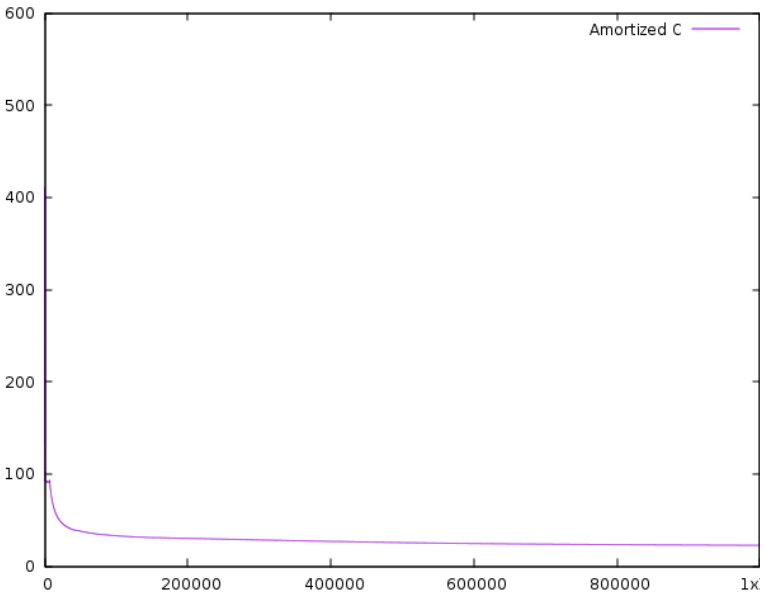
pour P=0,7

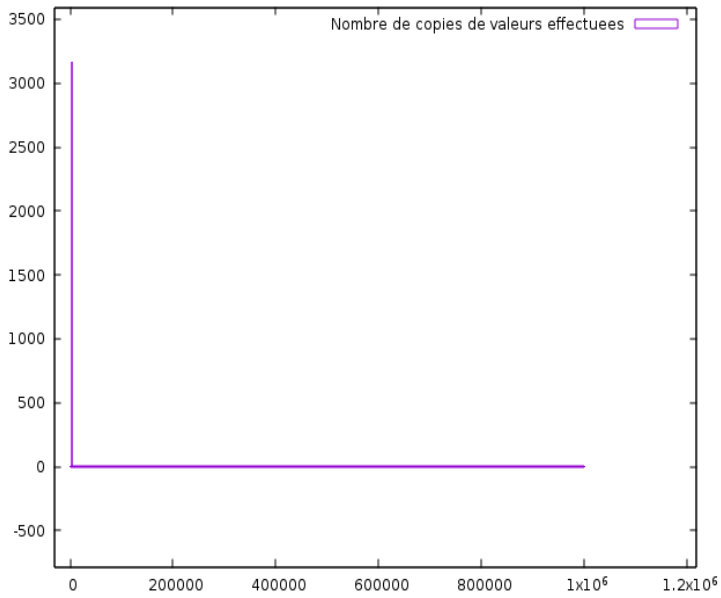

Amortized C
600
500
400
300
200
100
0
0
200000
400000
600000
800000
1x10

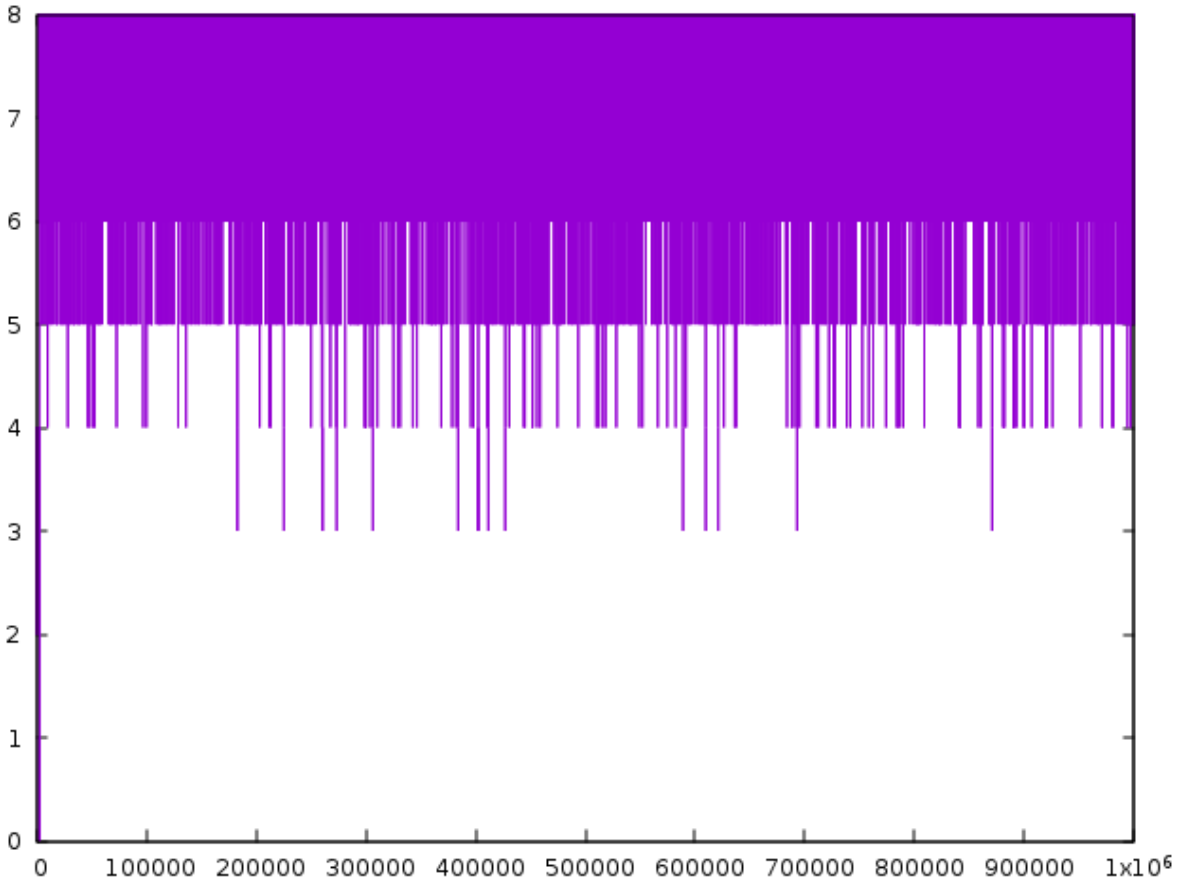

Nombre de copies de valeurs effectuees
1x10⁶
800000
600000
400000
200000
200000
400000
600000
800000
1x10⁶


16
14
12
10
8
6
4
2
0
0
100000
200000
300000
400000
500000
600000
700000
800000
900000
1x10⁶

pour P=0,9

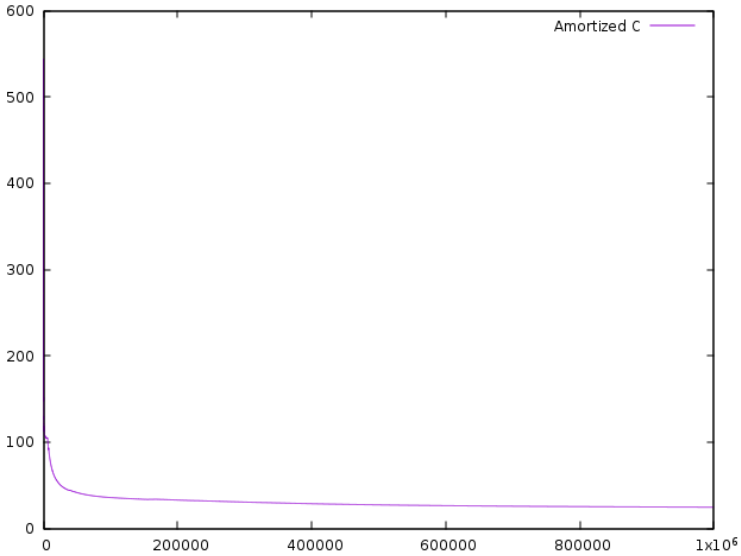

Amortized C
600
500
400
300
200
100
0
0
200000
400000
600000
800000

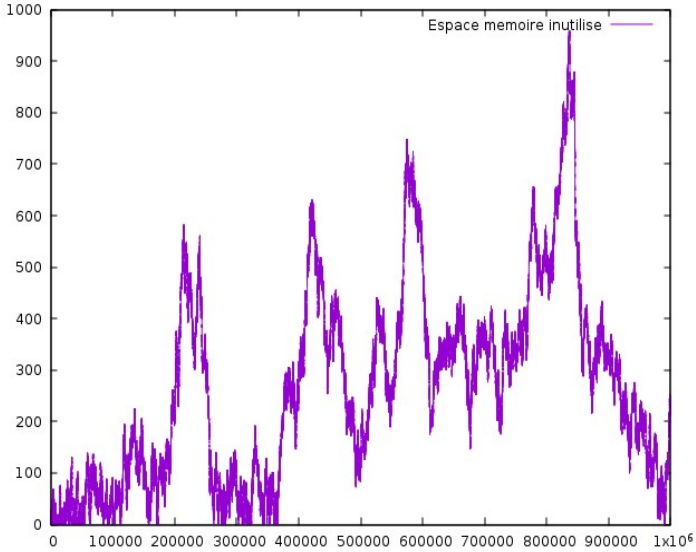

Nombre de copies de valeurs effectuees
3500
3000
2500
2000
1500
1000
500
0
-500
0
200000
400000
600000
800000
1x10⁶
1.2x10⁶


8
7
6
5
4
3
2
1
0
0
100000
200000
300000
400000
500000
600000
700000
800000
900000
1x10⁶

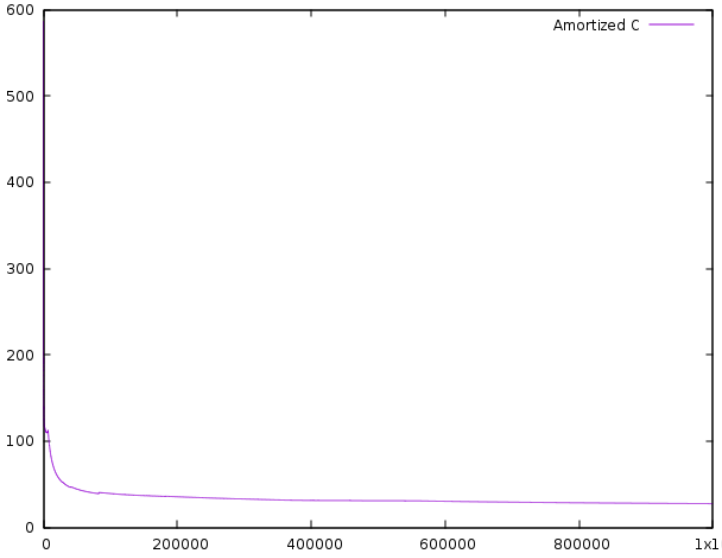
-on remarque que si p est grand alors on a moins d'insertion ce qui implique moins de mémoire inutilisé et un coût amortie moins que si p est petit alors on a beaucoup d'ajout et plus de mémoire inutilisé.

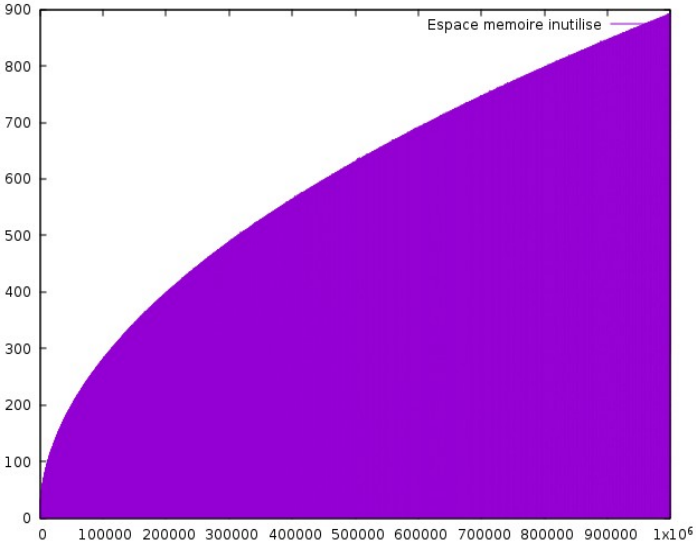
-on choisit p=0,5 et on modifie la stratégie de redimensionnement de la table.
il faut contracter la table quand le facteur de remplissage de la table $\alpha i = nomi/taillei = 1/2$
car $taillei+1 = taillei - sqrt(taillei)$ lors d'une contraction



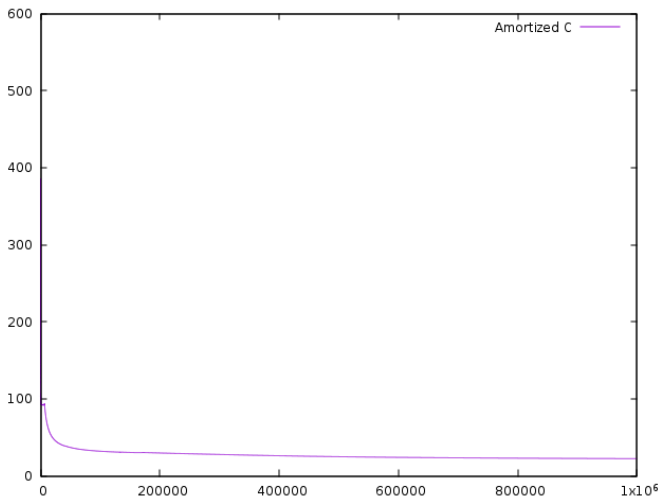


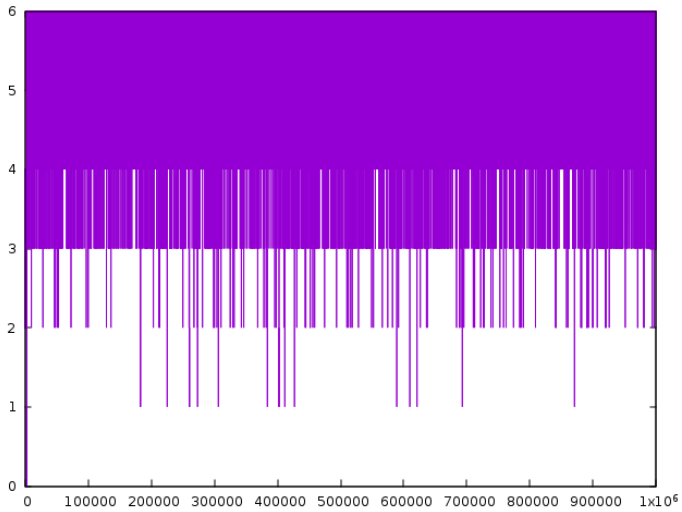
-on teste à nouveau les différentes valeurs de P :
pour P=0,1





pour P=0,9



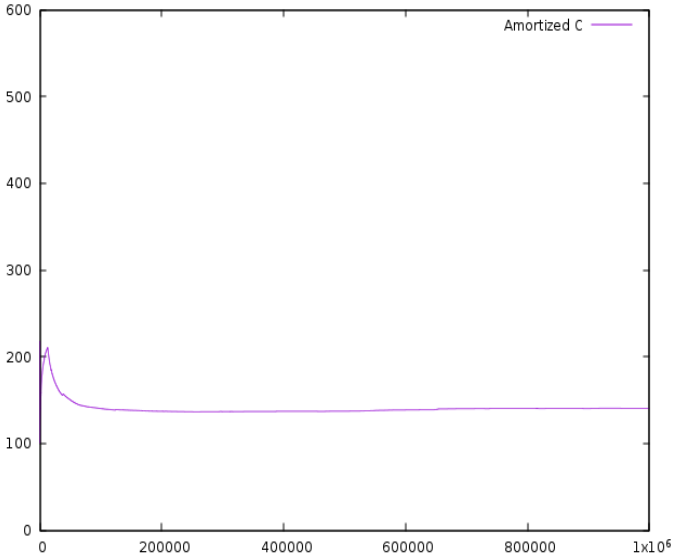


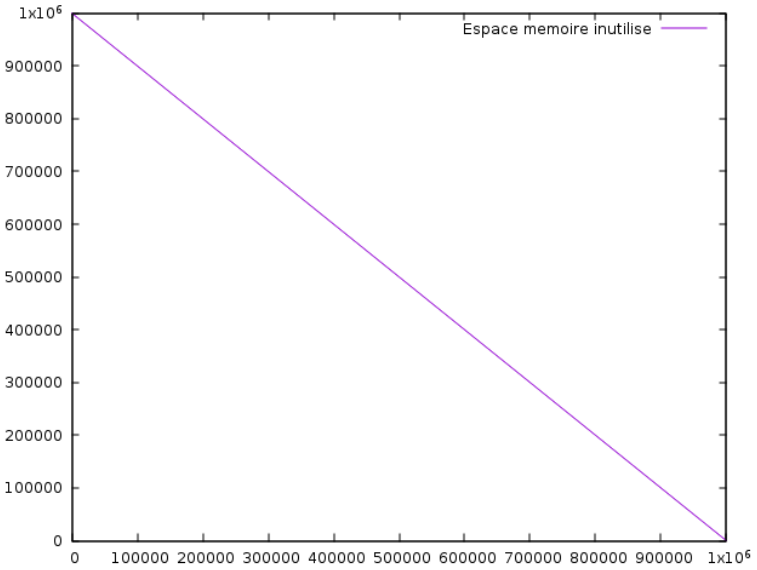
la valeur de p pour laquelle cette stratégie semble mieux fonctionner est P=0,5
parce que pour cette valeur on a des résultats sur lesquelles on peut baser nos expériences
pour P plus grand que 0,9 on fait beaucoup d'opération d'ajouts et pour moins de 0,5 on fait plus de suppression que d'ajouts.

TP3

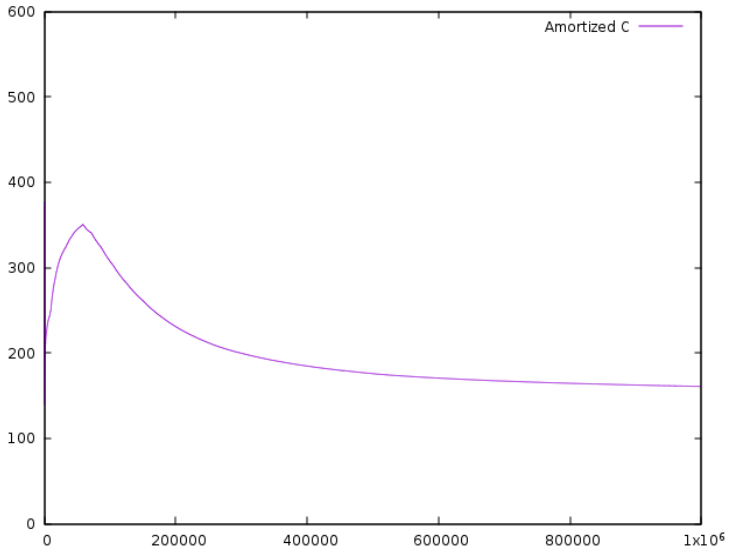
-les expériences effectués en temps et en mémoire de la structure de tas binaire statique :

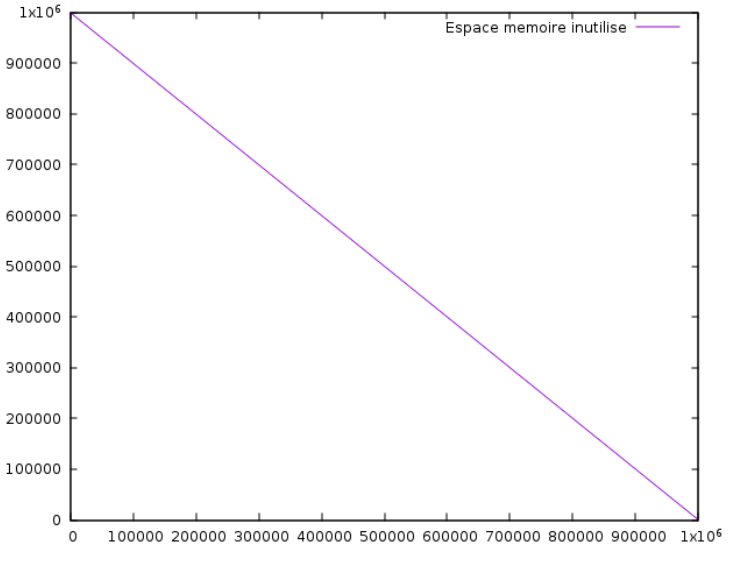
-dans le cas où l'on fait qu'ajouter des clés dans l'ordre croissant :dans ce cas on insert directement dans le tas sans besoin de faire d'équilibration vu que toute valeurs va étre plus grande que la précedente



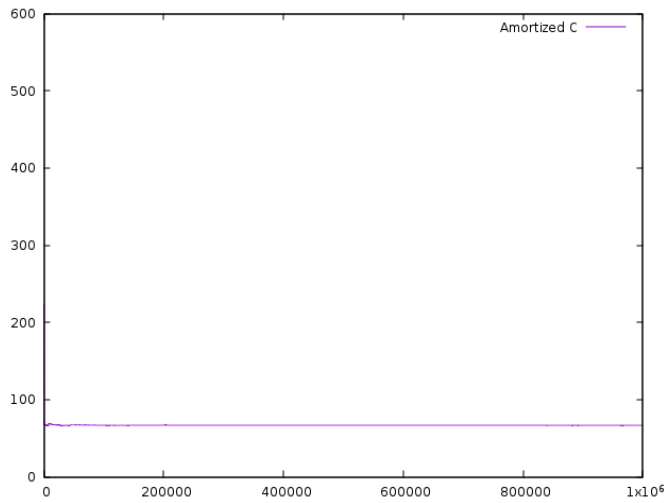


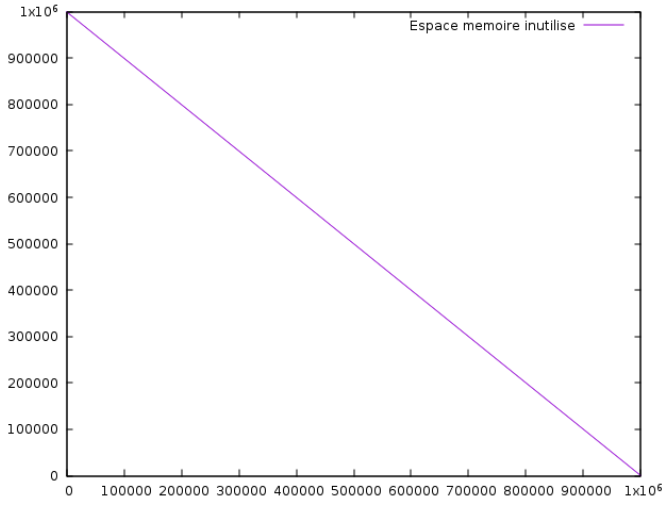
-dans le cas où l'on fait qu'ajouter des clés dans l'ordre décroissant :dans ce cas le coût amortie est augmenter parce que on dois faire une équilibration plusieurs fois après les insertions



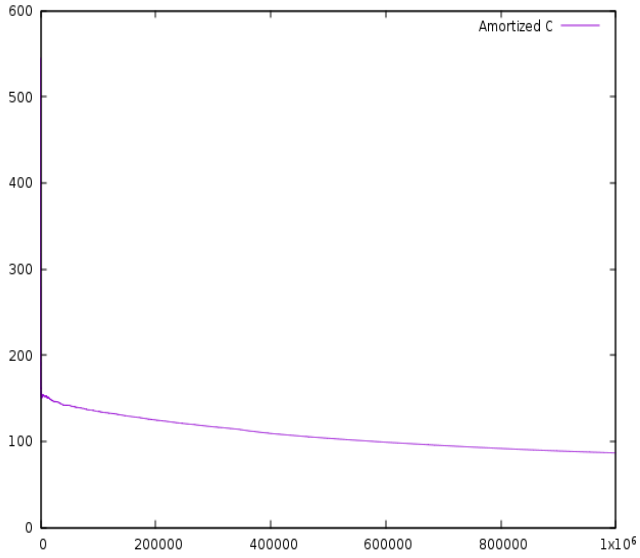


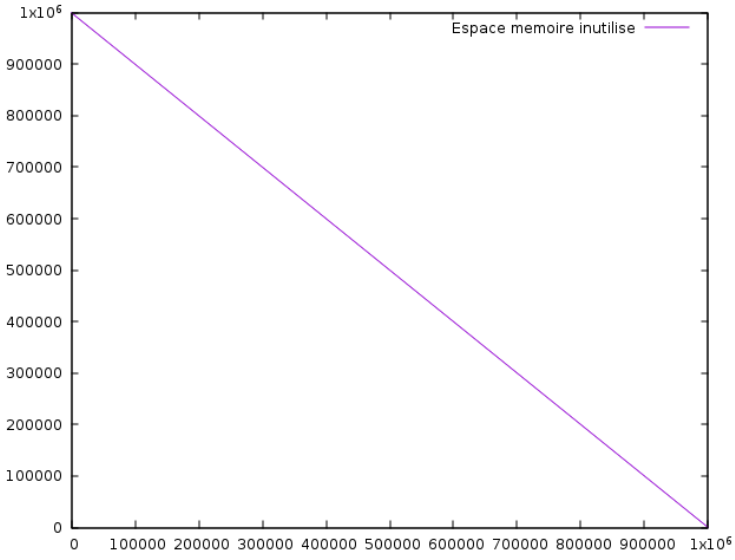
-dans le cas où l’on ne fait qu’ajouter des clés aléatoires :c’est la meilleure stratégie





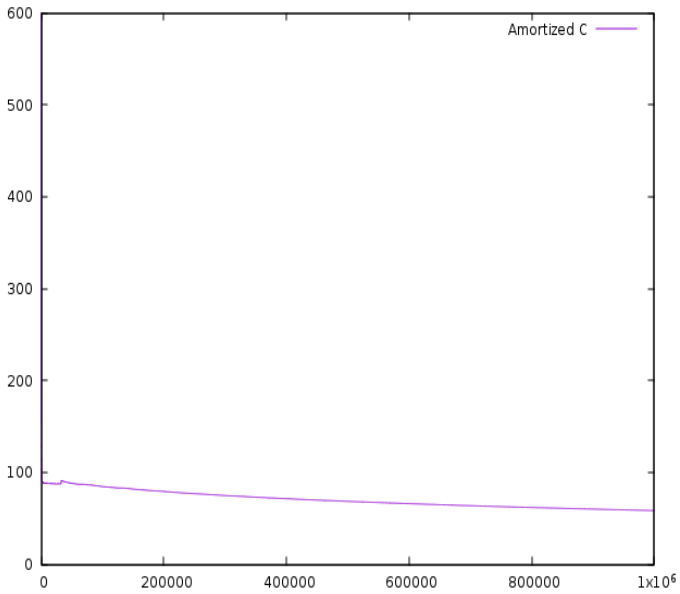
-dans le cas où l’on ajoute et on extrait des éléments: l’espace inutilisé ne change pas vu que c’est un tas avec capacité statique, alors que le coût amortie lui se diminue vu que le coût amortie d’une suppression est plus petit que le coût amortie d’une insertion dans la majorité des cas

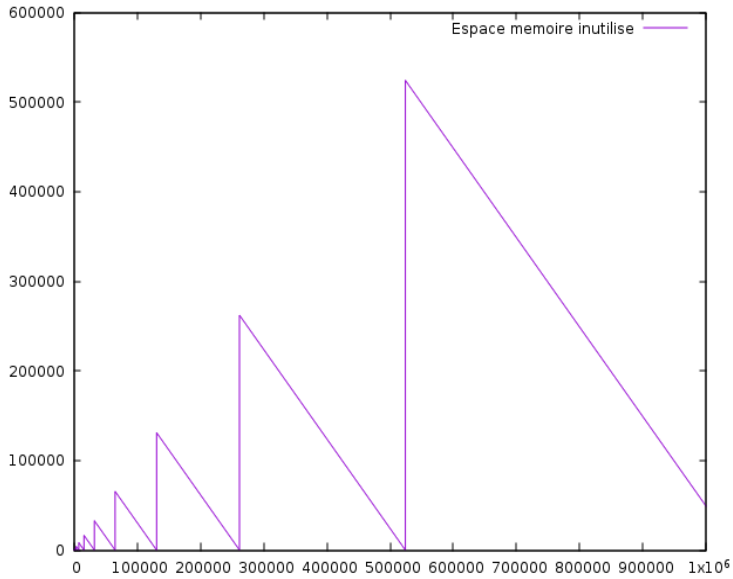




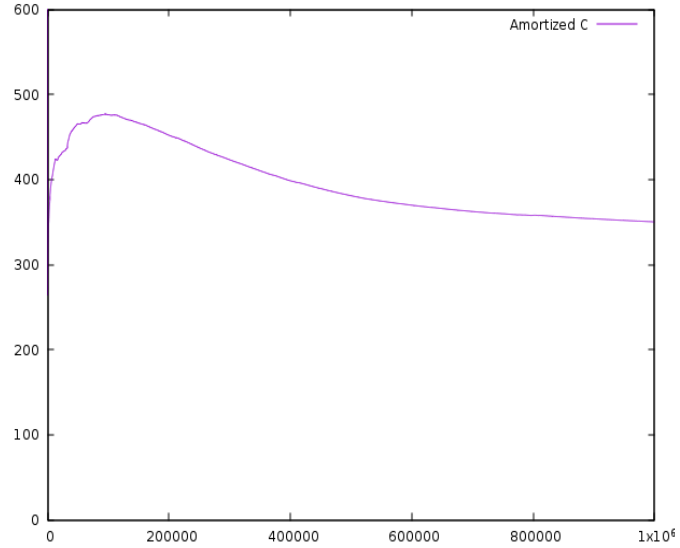
-les expériences effectués en temps et en mémoire de la structure de tas binaire statique :

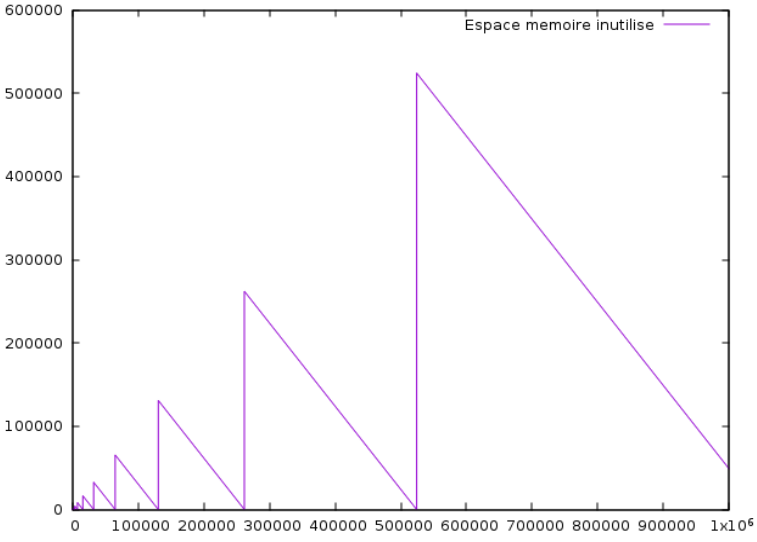
-dans le cas où l'on fait qu'ajouter des clés dans l'ordre croissant :



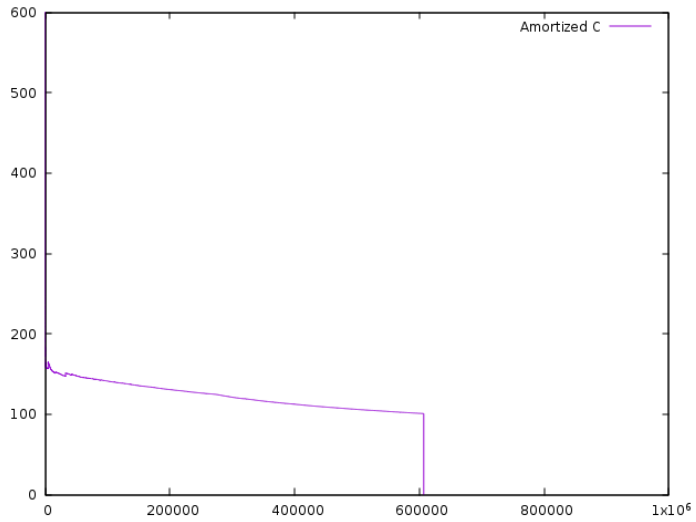


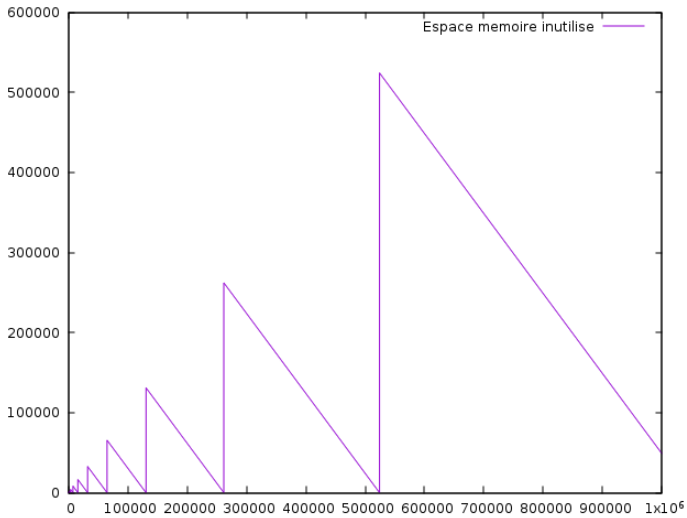
-dans le cas où l'on fait qu'ajouter des clés dans l'ordre décroissant :



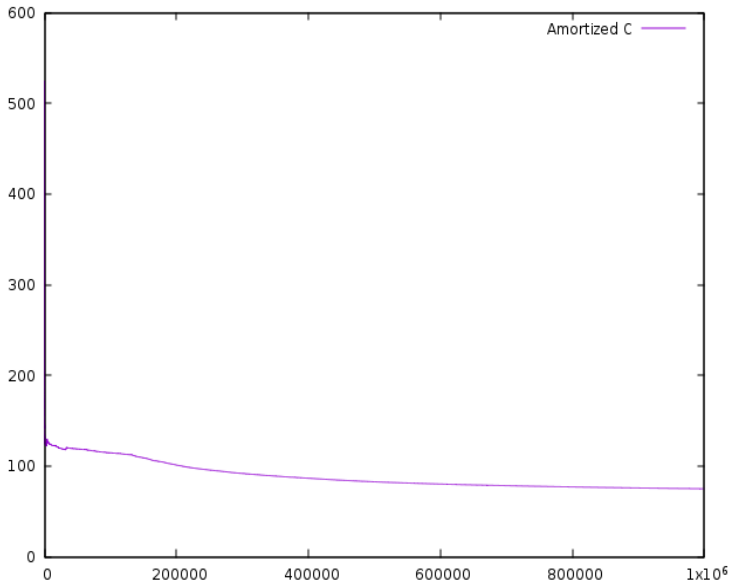


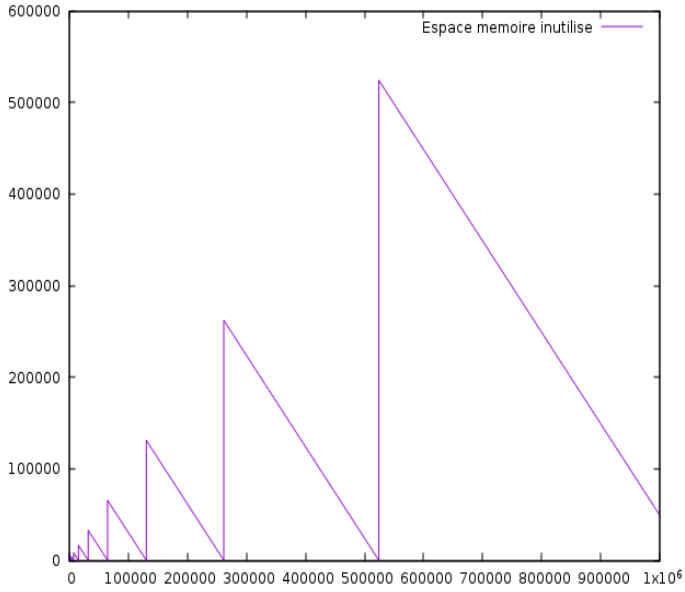
-dans le cas où l’on ne fait qu’ajouter des clés aléatoires :





-dans le cas où l’on ajoute et on extrait des éléments:



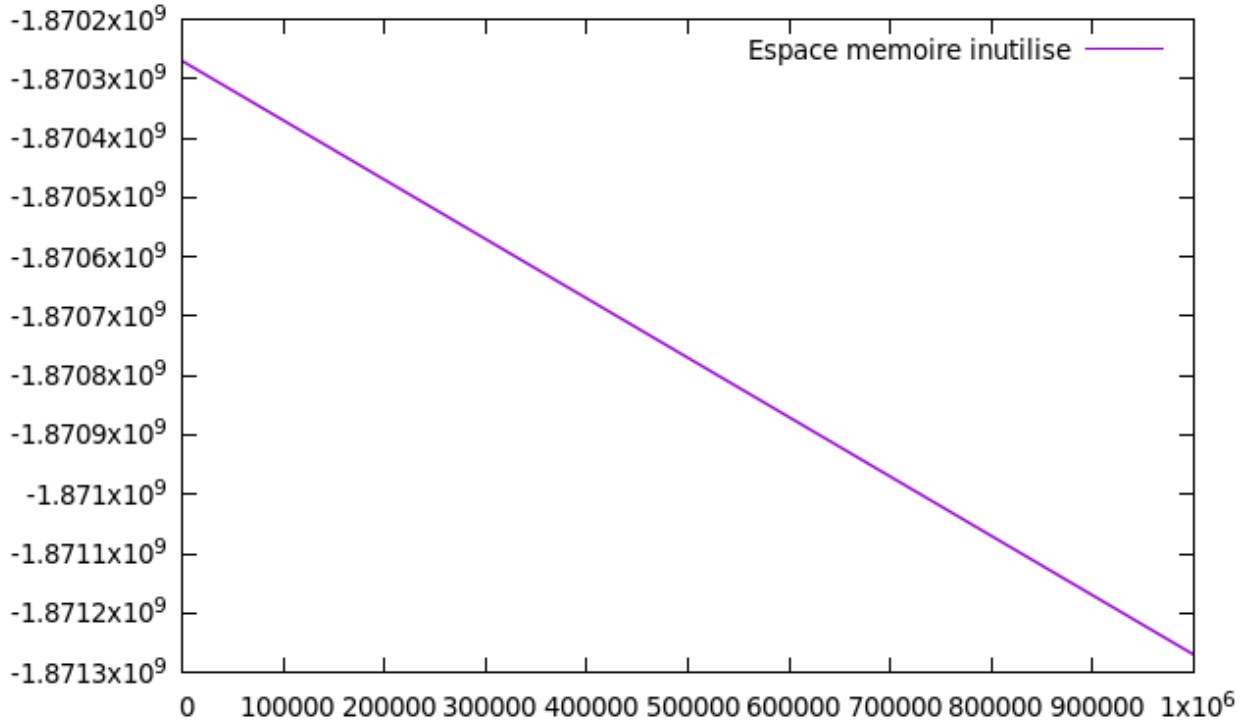


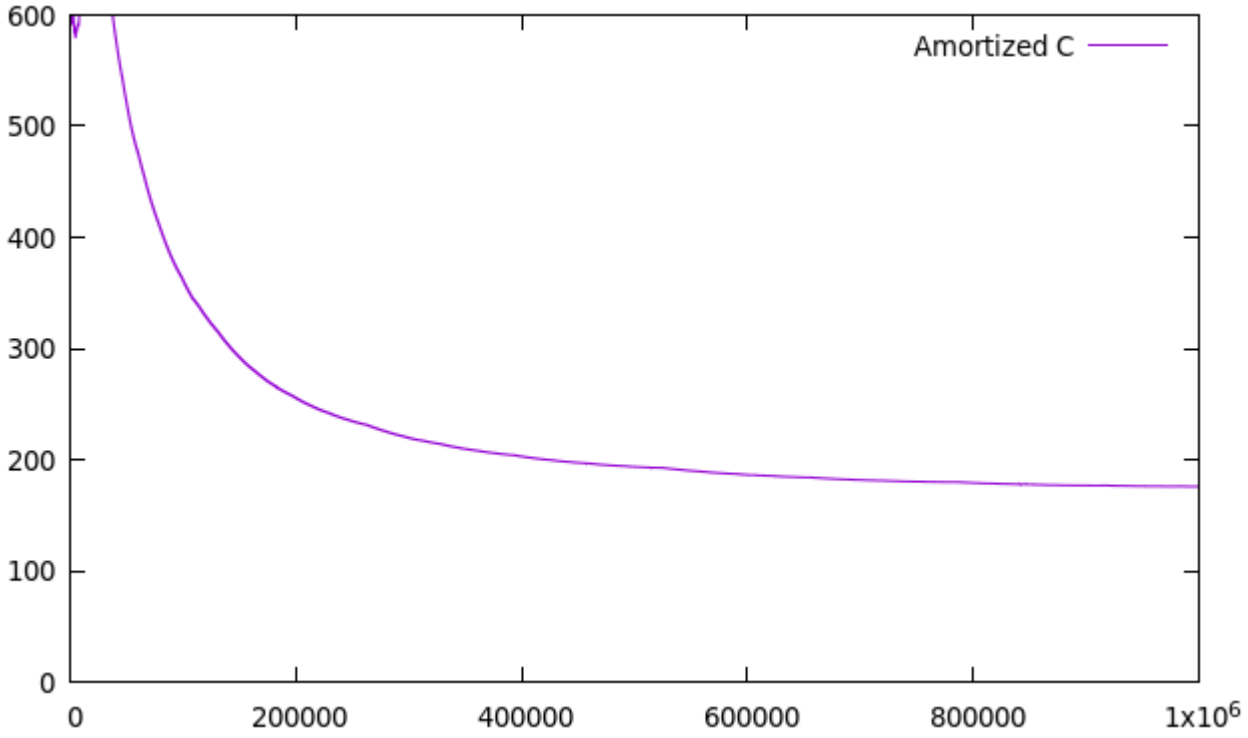
la comlexité amortie des operation d’jout et de supperession à changé parce que le cout amortie est relative au changement de la taille.

-les expériences effectués en temps et en mémoire de la structure de tas binomial :

-dans le cas où l'on ne fait qu'ajouter des clés **:**

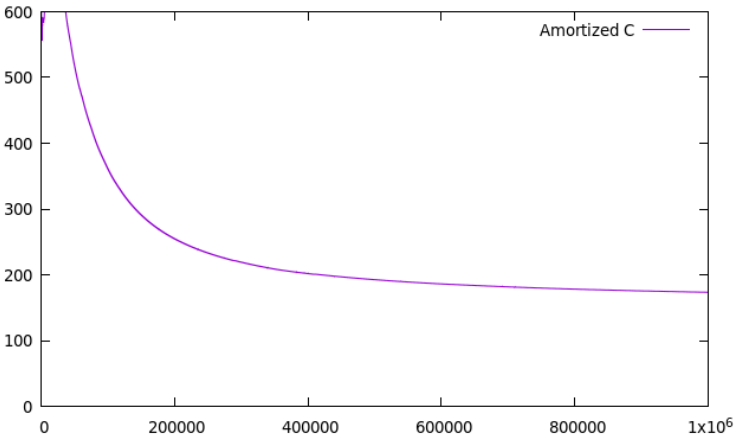
le coût amortie est élève au début de l'expérience lorsque qu'on fait qu'ajouter des clés dans l'ordre croissant, parce que au début y' trop d'opération pour équilibrer notre tas binomial.

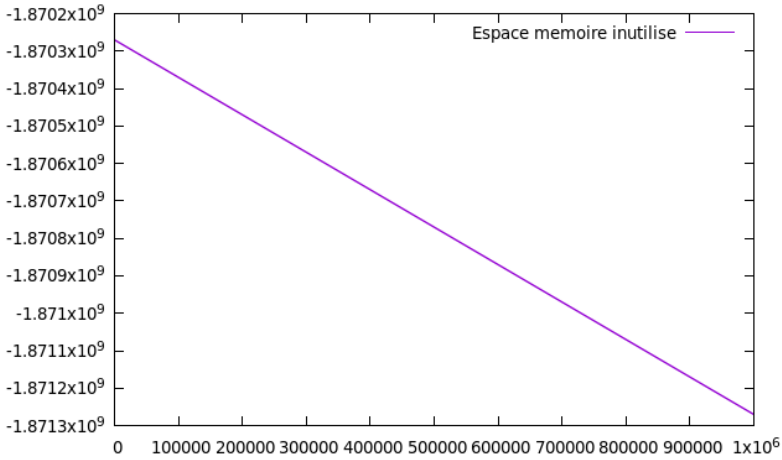




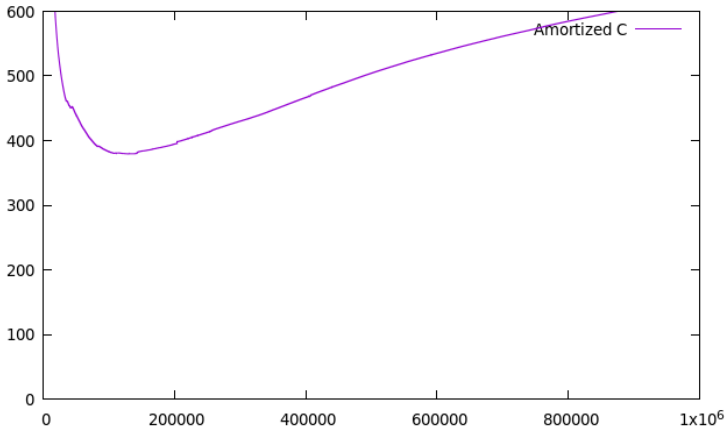
TP4

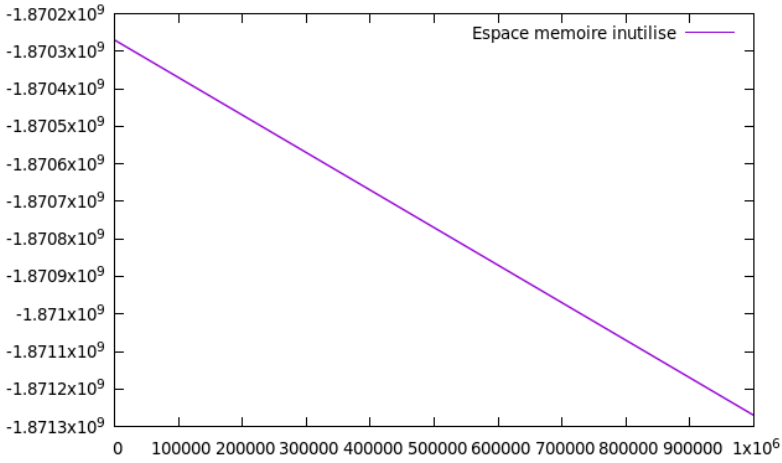
b-arbres : on a choisi pour chaque nœud une tableau des clés et un tableau des pointeurs pour ces fils, ainsi un boolean pour préciser es que c'est leaf ou pas.
-dans le cas où l'on fait qu'ajouter des clés dans l'ordre croissant :



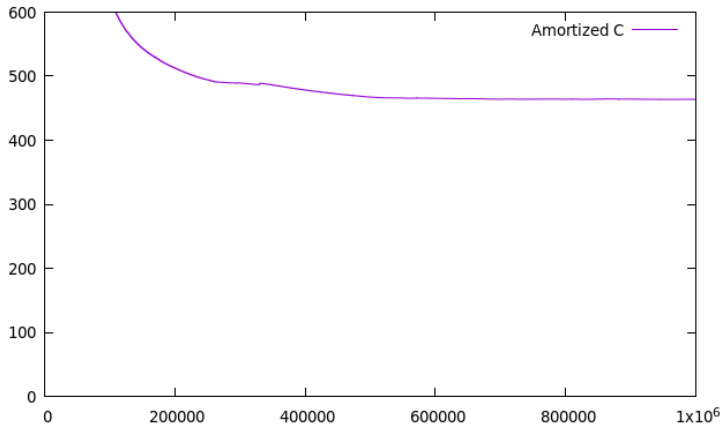


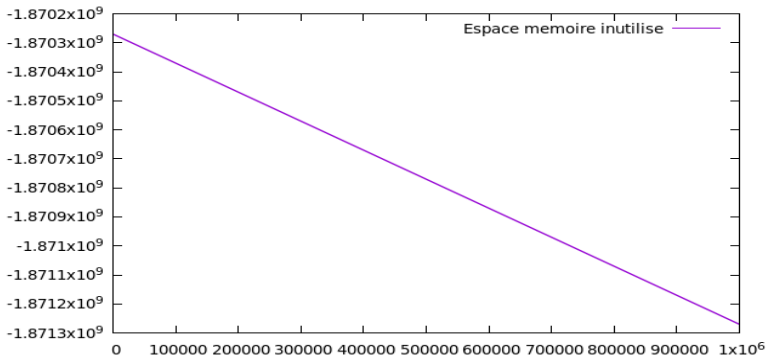
-dans le cas où l'on fait qu'ajouter des clés dans l'ordre aléatoire:





AVL
-dans le cas où l'on fait qu'ajouter des clés dans l'ordre croissant :





-dans le cas où l'on fait qu'ajouter des clés dans l'ordre aléatoires :