30/12/2019

# Rapport TP structure de données avancées

Hariss MOHAMMAD

Steave SUN

# *TP1*

## Question 1 :

Méthode du potentiel : O(Di) >= O(Do) = 0
O(i) = 0 après une extension
O(i) = ni = ti juste avant une extension
O(i) = 2ni - ti = 2i - ti
Ti: Capacité, 2ni : nombre d'élément

O (i) = xni - yti
Avant extension : xni - yti = ni
xni - yni = ni
y = 1 + y

Après extension : xni - yti = 0
xni - yαni = 0
x= yα

1 + y = yα
Y = 1 / (α-1) x = α / (α-1)
=> O(i) = (α / (α-1)) ni - (1 / (α-1)) ti
α = 2 => O(i) = 2ni -ti

On obtient la fonction potentielle : p = 1+2+...+2^n

## Question 2 :

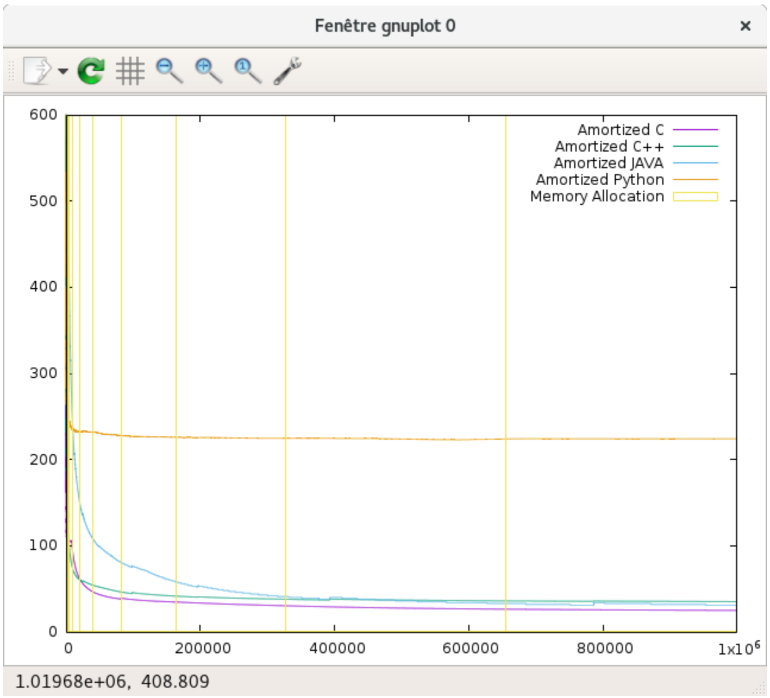
Ci = ci + O(i) - O(i-1)
Cas 1 : pas d'extension
Ci = 1 + ( (α / (α-1))ni - (1 / α-1)ti ) - ( (α / α-1)ni-1 - (1 / (α-1))ti-1 )
Ci = 1 + (α / (α-1))ni - (1 / α-1)ti - (α / (α-1))ni + (α / (α-1)) + (1 / (α-1))ti
Ci = 1 + (α / (α-1)) = θ(α)

Cas 2 : extension
Ci = ci + θ(i) - θ(i-1)
Ci = 1 + (α / (α -1)) = θ(α)

Composé de n éléments de coûts θ(α) = θ(1) = n θ(α) = θ(αn) = θ(n) avec α constant

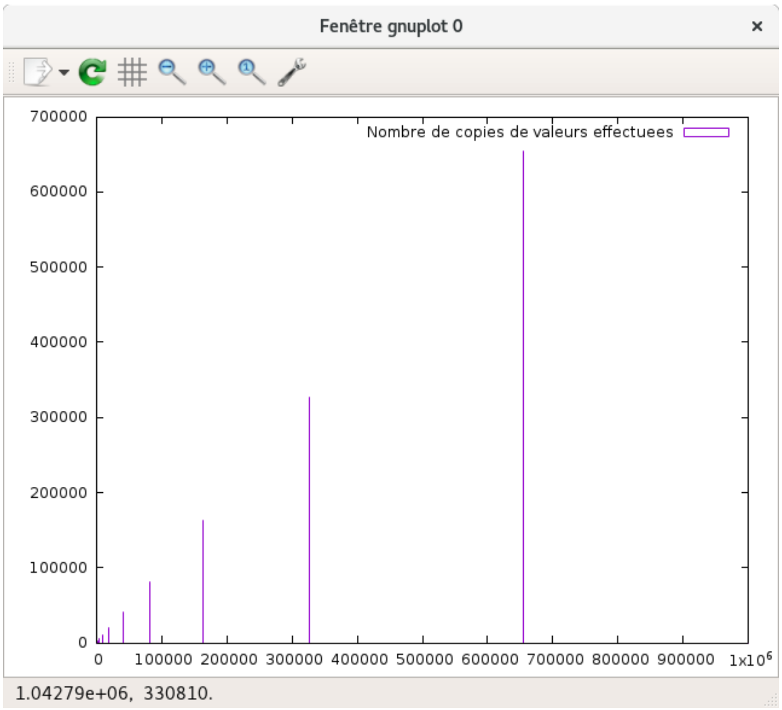
## Question 3 :

a) Le programme le plus long à s'exécuter est le celui du code écrit en Python car il y a un facteur d'agrandissement et par le fait que le programme sauvegarde sur le disque dur. La complexité de ces fonctions est linéaire car il faut écrire dans le fichier.



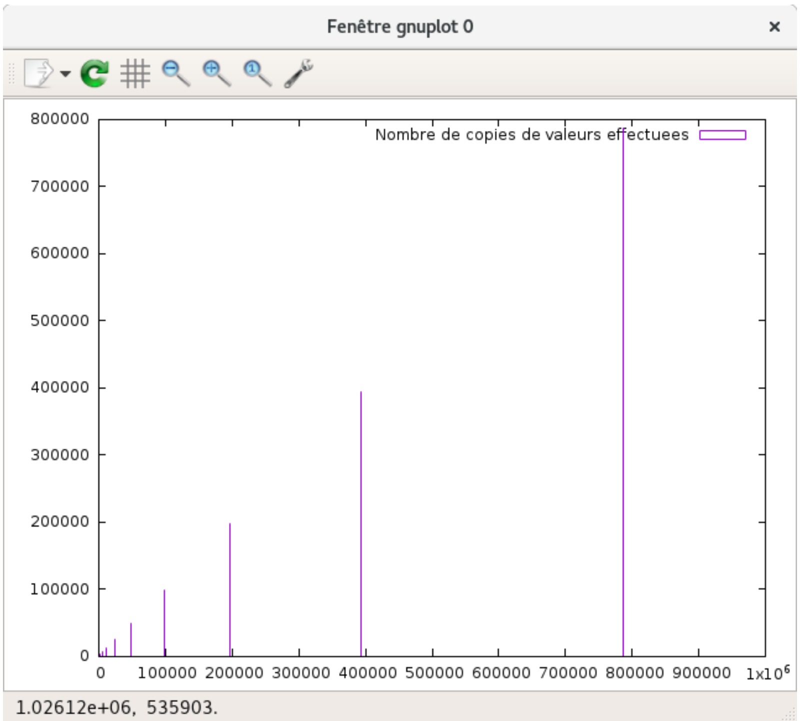

b)

c) On remarque que l'augmentation du coût amorti est lié au moment où les copies sont effectués. Coût amorti voir q3a
C (copies) :

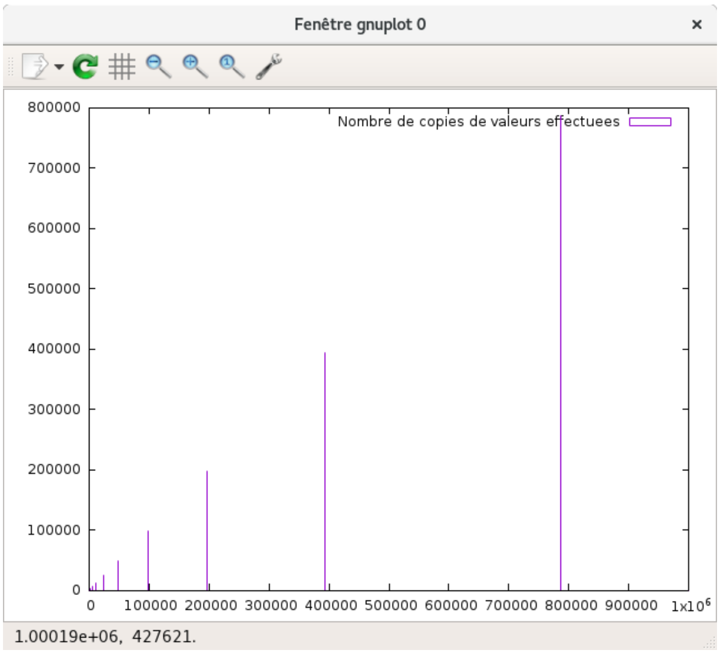



d) Le temps d'exécution change d’une expérience à l’autre. Les copies effectuées ne changent pas.

JAVA :

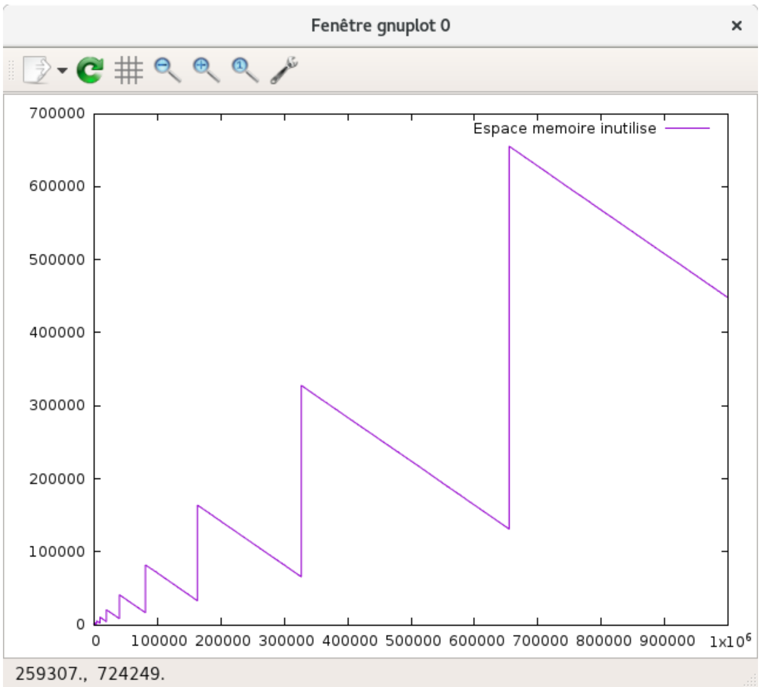


C++ :




e) Certains langages sont plus rapides comme le C ou le C++ car ils sont totalement compilés. Le JAVA et le Python sont plus lents car ils sont interprétés.

f) L'agrandissement de la taille du tableau s'effectue lorsqu'il est remplis au ¾, donc il y a ¼ qui sont pas utilisé. Elle augmente au fil de l'exécution, ce qui pourrait poser un problème car d'autre processus pourront avoir besoin de mémoire.



## Question 4 :

Nous avons modifié la condition : la taille est égale à la capacité. Cela permet d’utiliser toute la mémoire alloué au programme au lieu de 3/4.

## Question 5 :

Lorsqu’on fait varier le facteur multiplicatif :

- A 75%, nous avons une moyenne d’environ 23,6 ; 23,8
- A 100% nous avons une moyenne d’environ 23,2 ; 23,5

On observe donc une baisse lorsque le tableau est rempli totalement.

## Question 6 :

ti = ti-1 + sqrt (ti-1)
ti-1 + sqrt (ti-1) = α ti-1
α = (ti-1+sqrt (ti-1)) / ti-1 = 1 + 1/sqrt (ti-1)
Pas constant

Le nombre de copie effectué à diminué

# *TP2*

## Question 1 :

$\alpha i = ni / ty <= ⅓$
$ti = ⅔\ ti-1$

Cas 1 : Avant extension
$\hat{C}i = ci + \Phi i - \Phi i$
$\hat{C}i = 1 + |2xni-ti| - |2*ni-1 - ti-1 |$
On sait que :
$ni-1 = ni+1$
$Ti-1 = ti$
$\alpha i-1 = ni-1 / ti-1 > ⅓$
$\Leftrightarrow ni-1 / ti > ⅓$
$\Leftrightarrow ti > 3(ni+1)$
$\Leftrightarrow ti > 3ni + 3$
$\Leftrightarrow 3ni + 3 -ti < 0$
$2ni - ti < 0$
$\hat{C}i = 1 + (ti - 2ni) - |2 (ni + 1) - ti|$
$\hat{C}i = 1 + ti - 2ni - (ti - 2(ni+1))$
$\hat{C}i = 1 + ti - 2ni - ti + 2ni + 2$
$\hat{C}i = 3$

Cas 2 : Apres extension
$\hat{C}i = ni-1 - 1 + | 2ni - ti | - |2ni-1 - ti-1|$
On sait que :
$ni-1 = ni+1$
$ti-1 = 3/2ti$
$\alpha i = ni / ti <= 1 / 3$
$\Leftrightarrow ti-1 >= 3ni-1$
$\Leftrightarrow 3/2\ ti >= 3(ni +1)$
$\Leftrightarrow ti >= 2(ni + 1)$
$\hat{C}i = ni + (ti - 2\ ni) - |2 (ni + 2 - 3/2 * 2(ni +1)|$
$\hat{C}i = ni + 2ni + 2 - 2ni - | 2ni + 2 - 3/2 * 2(ni + 1)|$
$\hat{C}i = ni + 2 - |2ni + 2 - 3ni -3 |$
$\hat{C}i = ni + 2 - |-ni - 1|$
$\hat{C}i = ni + 2 - (ni+1)$
$\hat{C}i = ni + 2 - ni - 1$
$\hat{C}i = 1$

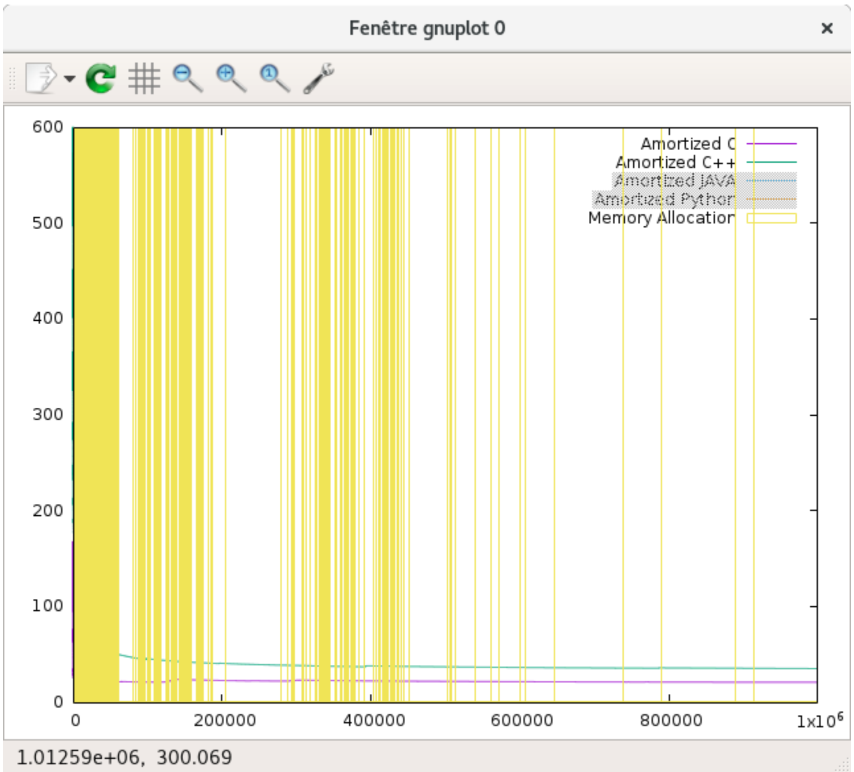
## Question 2 :

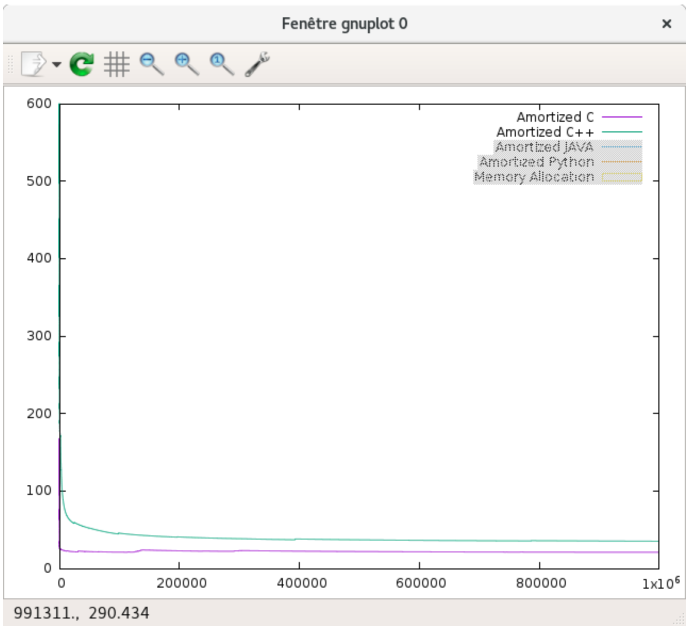
```
for(i = 0; i < 1000000 ; i++){
        if(rand()%3){
         // Ajout d'un élément et mesure du temps pris par l'opération.
         clock_gettime(clk_id, &before);
         memory_allocation = arraylist_append(a, i);
         clock_gettime(clk_id, &after);
        }
        else{
  timespec_get(&before,TIME_UTC);
  memory_allocation = arraylist_pop_back(a);
  timespec_get(&after,TIME_UTC);
        }
        // Enregistrement du temps pris par l'opération
        analyzer_append(time_analysis, after.tv_nsec - before.tv_nsec);
        // Enregistrement du nombre de copies efféctuées par l'opération.
        // S'il y a eu réallocation de mémoire, il a fallu recopier tout le tableau.
        analyzer_append(copy_analysis, (memory_allocation)? i:1 );
        // Enregistrement de l'espace mémoire non-utilisé.
        analyzer_append(memory_analysis,arraylist_capacity(a)-arraylist_size(a));

 }
```

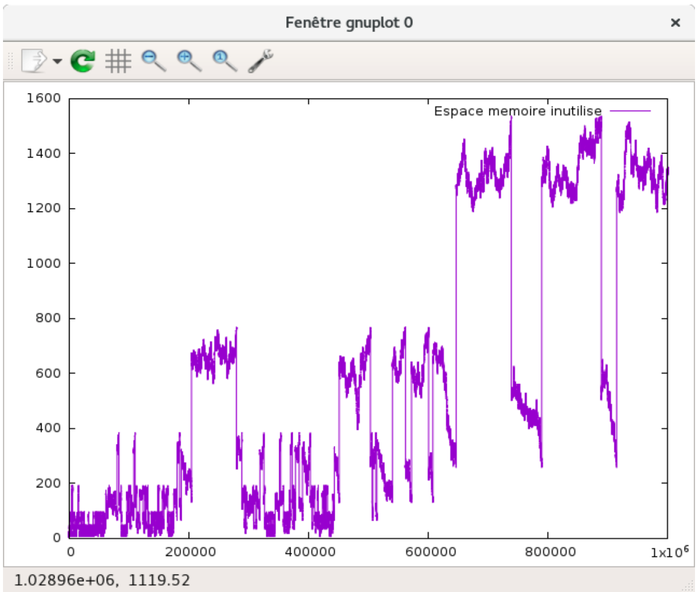
## Question 3 :

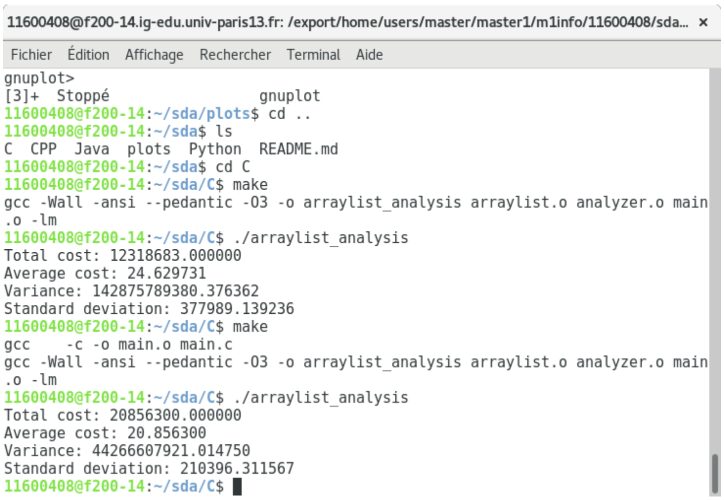
Coût amorti :






Espace mémoire utilisée :



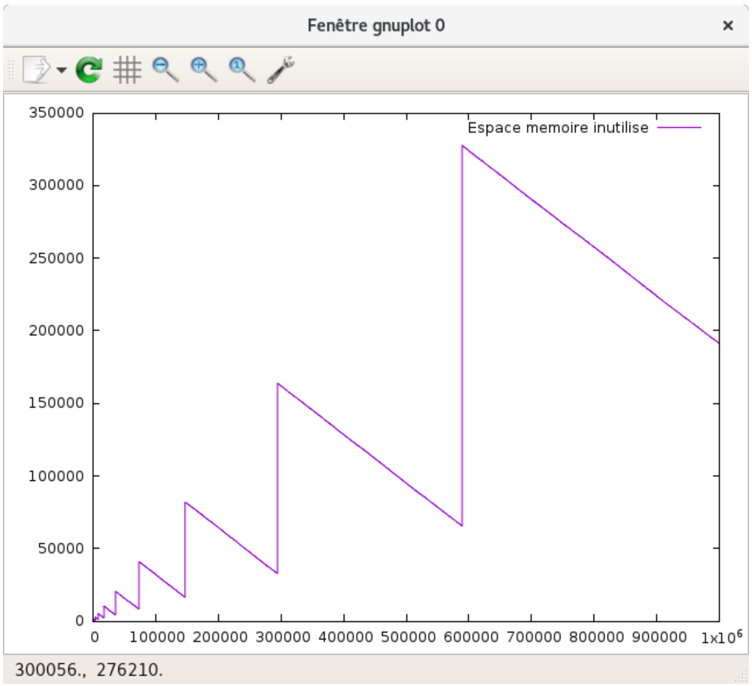
11600408@f200-14.ig-edu.univ-paris13.fr: /export/home/users/master/master1/m1info/11600408/sda...

Fichier Édition Affichage Rechercher Terminal Aide

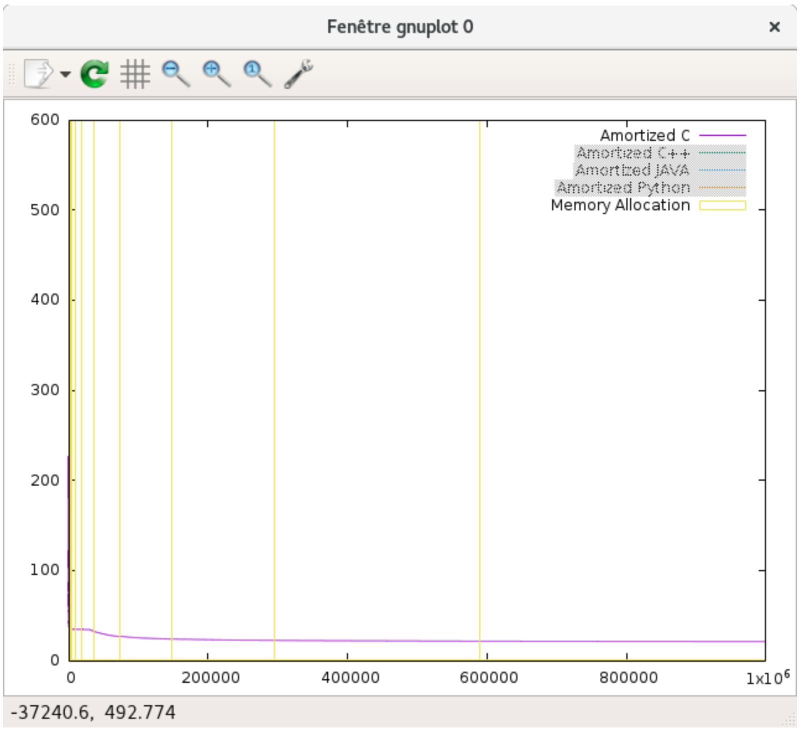
```
gnuplot>
[3]+  Stoppé                  gnuplot
11600408@f200-14:~/sda/plots$ cd ..
11600408@f200-14:~/sda$ ls
C  CPP  Java  plots  Python  README.md
11600408@f200-14:~/sda$ cd C
11600408@f200-14:~/sda/C$ make
gcc -Wall -ansi --pedantic -O3 -o arraylist_analysis arraylist.o analyzer.o main
.o -lm
11600408@f200-14:~/sda/C$ ./arraylist_analysis
Total cost: 12318683.000000
Average cost: 24.629731
Variance: 142875789380.376362
Standard deviation: 377989.139236
11600408@f200-14:~/sda/C$ make
gcc    -c -o main.o main.c
gcc -Wall -ansi --pedantic -O3 -o arraylist_analysis arraylist.o analyzer.o main
.o -lm
11600408@f200-14:~/sda/C$ ./arraylist_analysis
Total cost: 20856300.000000
Average cost: 20.856300
Variance: 44266607921.014750
Standard deviation: 210396.311567
11600408@f200-14:~/sda/C$
```

## Question 4 :

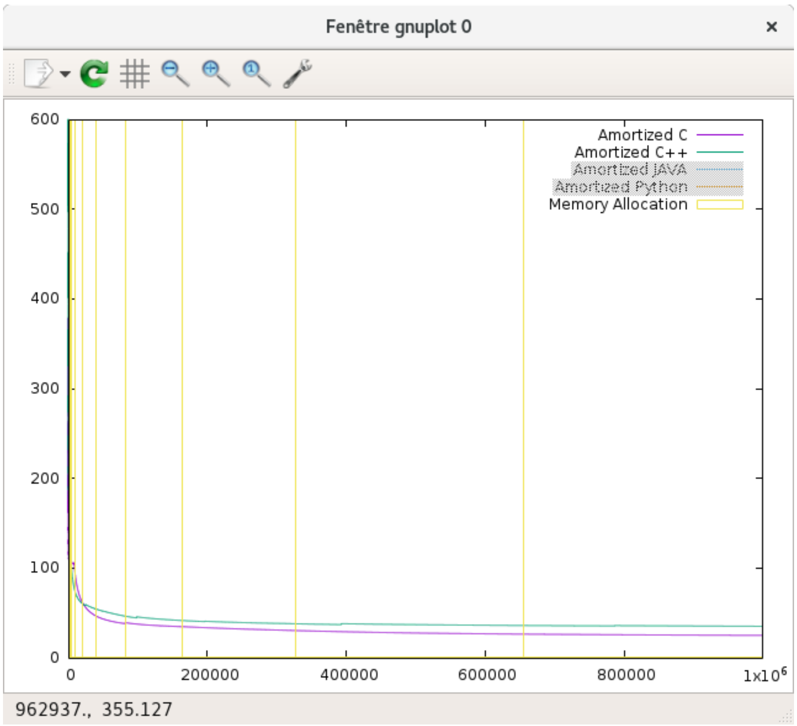
Pour p = 0.3
Espace mémoire inutilisé :



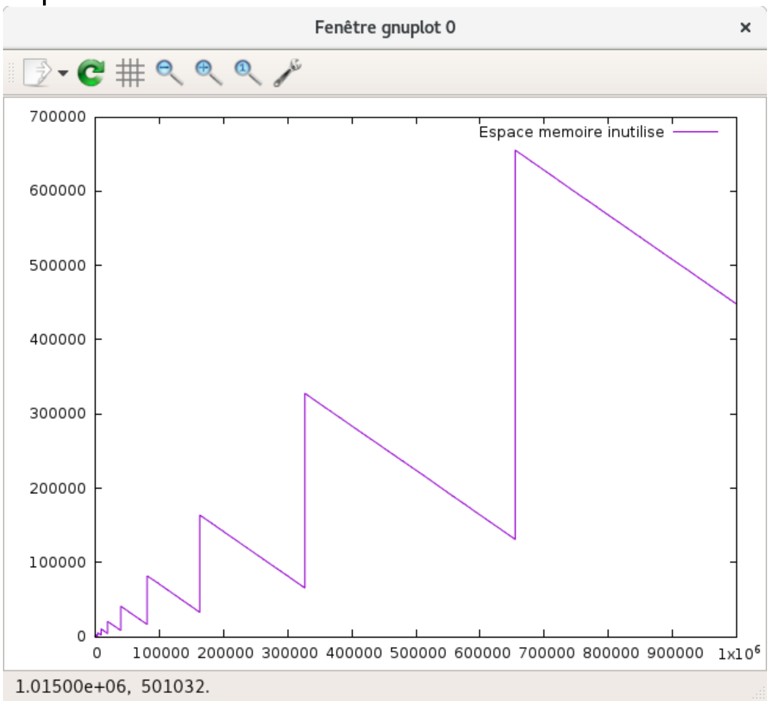

Coût amorti :



Pour p = 0.2
Coût amorti:




Espace mémoire non utilisé :




La relation qui existe entre p, le coût en temps et le gaspillage de mémoire est : lorsque p diminue, le coût en temps et la mémoire augmente.

## Question 5 :

Lors de l’extension, on observe une diminution du coût totale et lors d’une contraction, on observe une augmentation du coût.

## Question 6 :

La relation qui existe entre p, le coût en temps et le gaspillage de mémoire est identique à la question 4, si p diminue, le coût et la mémoire augmente.

# TP4

## Question 1 :

Voir code B-arbre

## Question 2 :

Pour représenter le nœud de l'arbre, une structure composée de 3 variables a été créé :

- **Int n :** correspond au nombre de clefs contenus dans le nœud
- **Int key[M-1] :** tableau qui contient les clefs de l'arbre
- **Struct _node *p[M] :** structure, pointeur qui permet de représenter les fils, une feuille ne contient pas de pointeur

La liste de ses clés est située dans le tableau Key, celles de ses enfants sont situées dans la structure p.

L'effet que cela a sur les opérations de fusion et de scindage des nœuds :
Notre code n'effectue pas les opérations de fusion et de scindage. Ces fonctions n'ont pas été définies.

## Question 3 :

Voir code AVL