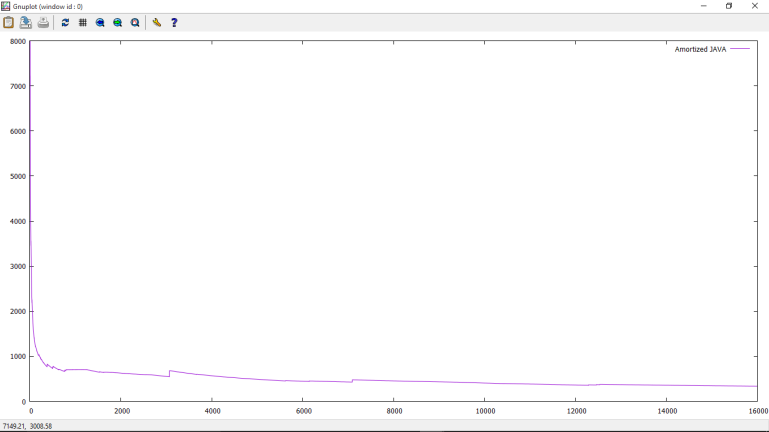
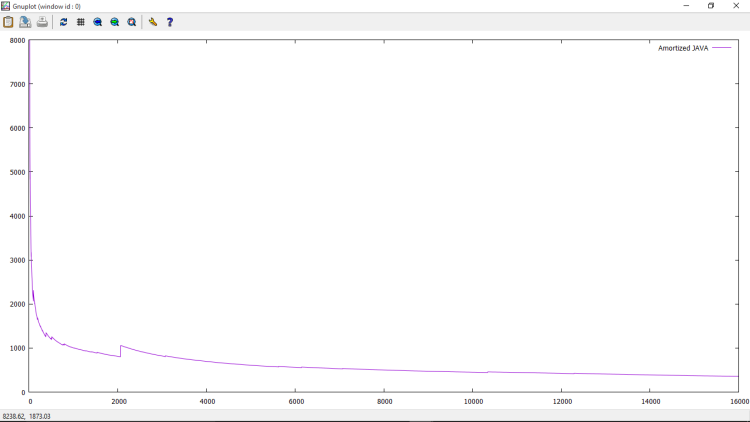
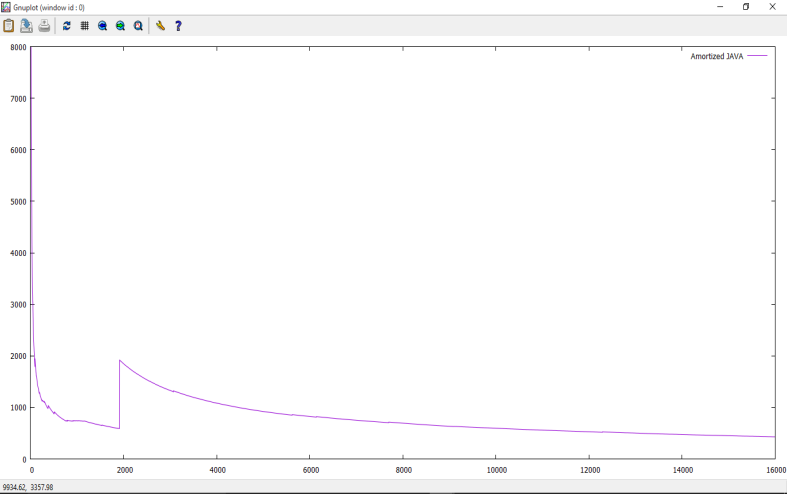
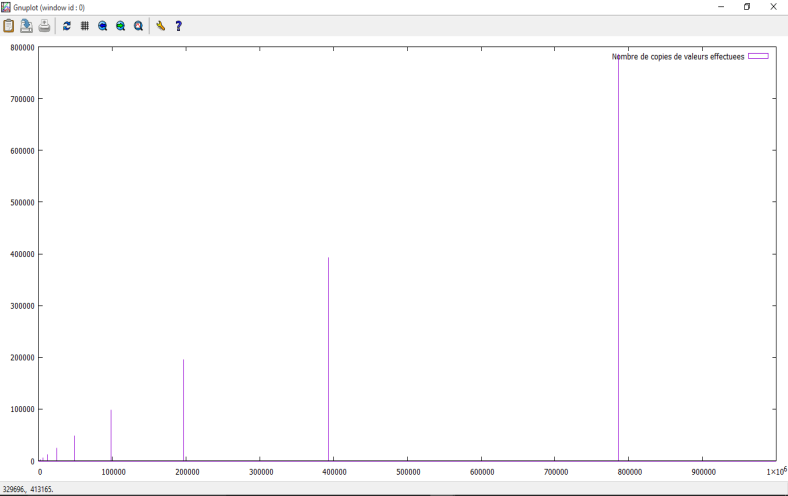
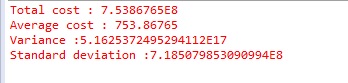
***TP1***

3.

Les coûts amortis et réels de l’opération Insérer-Table JAVA

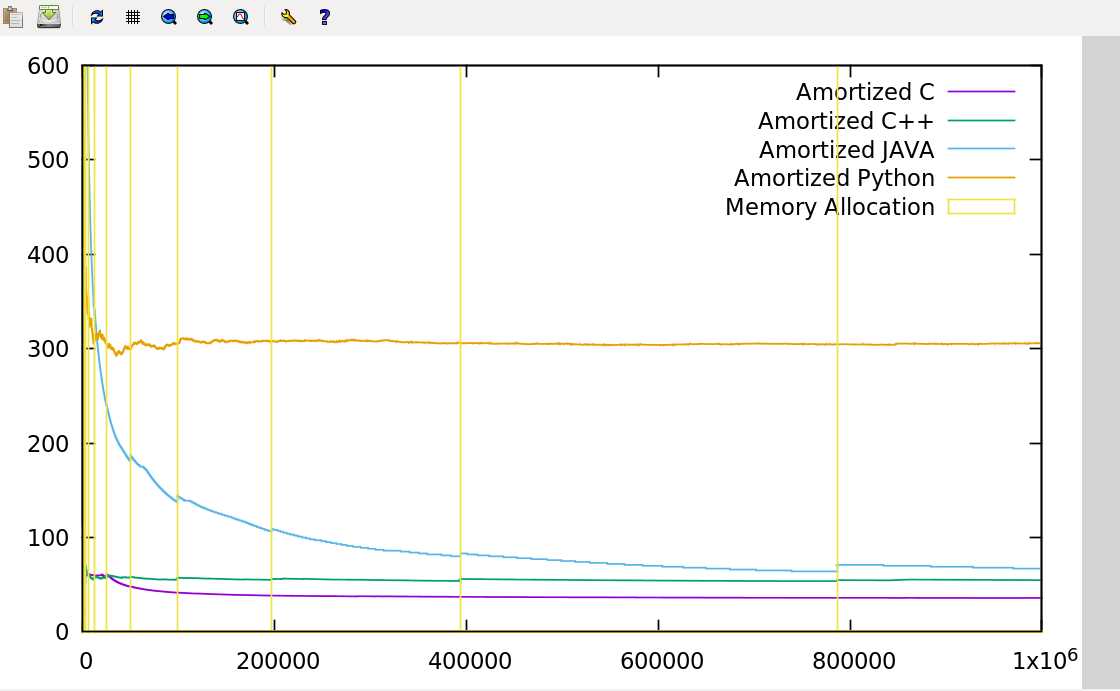


a).

la fonction append est le morceau qui prends le plus de temps car elle contient un if avec la fonction « do\_ we\_ need\_ to\_ enlarge\_capacity » puis dans ce if il ya la fonction qui agrandit la taille du tableau(on comprends ici que quand la taille du tableau a atteint la valeur qui est dans do we need to.. alors on applique un élargissement de la taille qui se trouve dans enlarge\_capacity)

la complexité = complexité de do\_we\_need\_to\_enlarge\_capacity(qui est le cout de la comparaison entre taille du tableau et la capacité max)+enlarge\_capacity(le cout de la réallocation mémoire de taille x)

b).



java= explosion

python= stable

c++=stable

c=semi-stable

Le cout amortis est une sorte de moyenne pour le temps réel, dans nos expériences le temps réel représente les insertions dans le tableau et les réallocations mémoires lorsque le tableau est plein, dans tous les langages il s’agit des mêmes opérations et pourtant le temps amortis diffère il s’agit donc de la façon dont chaque langage fonctionne

le programme ne tourne pas seul sur la machine donc parfois il arrive que les résultats ne sont pas toujours les mêmes

e)

Certains langages sont plus rapides que d’autres dans cette expérience

JAVA (qui accélère dans certains cas) compile les bouts de code souvent

JAVA et PYTHON sont interprètes

C et CPP sont compilés

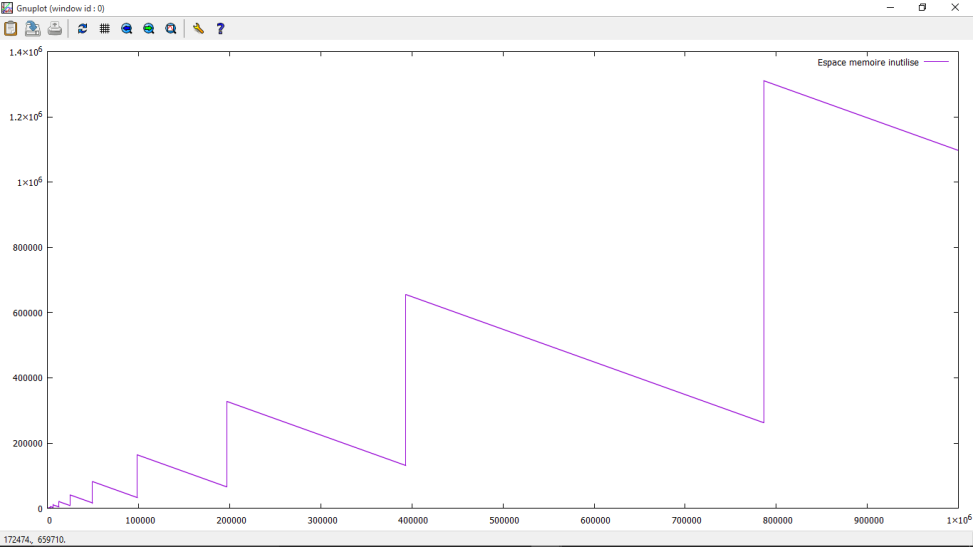
CPP doucle la taille mémoire

JAVA multiplie par nombre d’or

C++ , JAVA et PYTHON on code une classe qui appelle une autre (arraylist) qui fait le mm test donc c’est inutile contrairement au C.

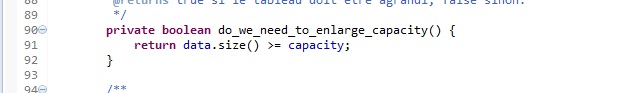
f).

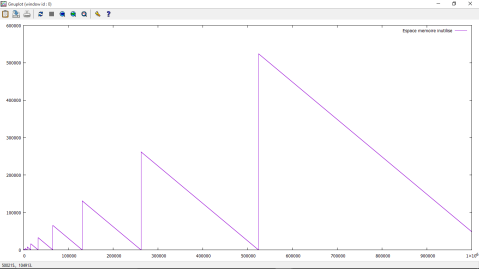
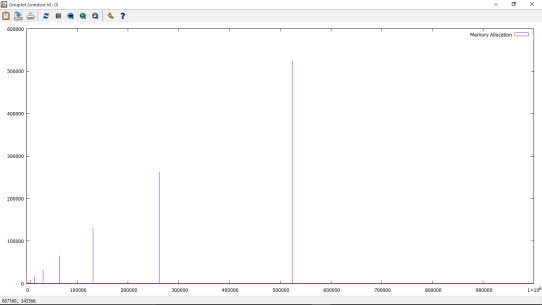
Espace mémoire inutilisé : la même dans tous les langages, le cas ou ça pose problème c’est lorsqu’on ajoute un espace mémoire mais qu’on ne l’utilise pas entièrement jusqu'à la fin.

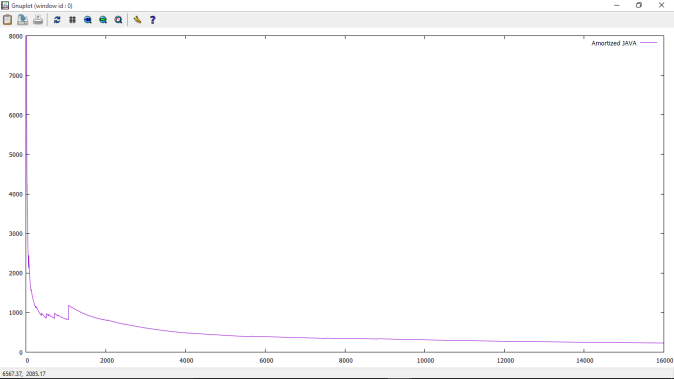
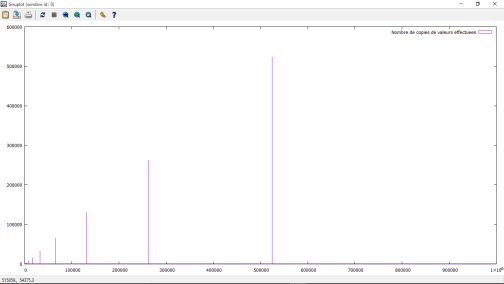


4.

Langage JAVA



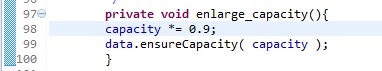


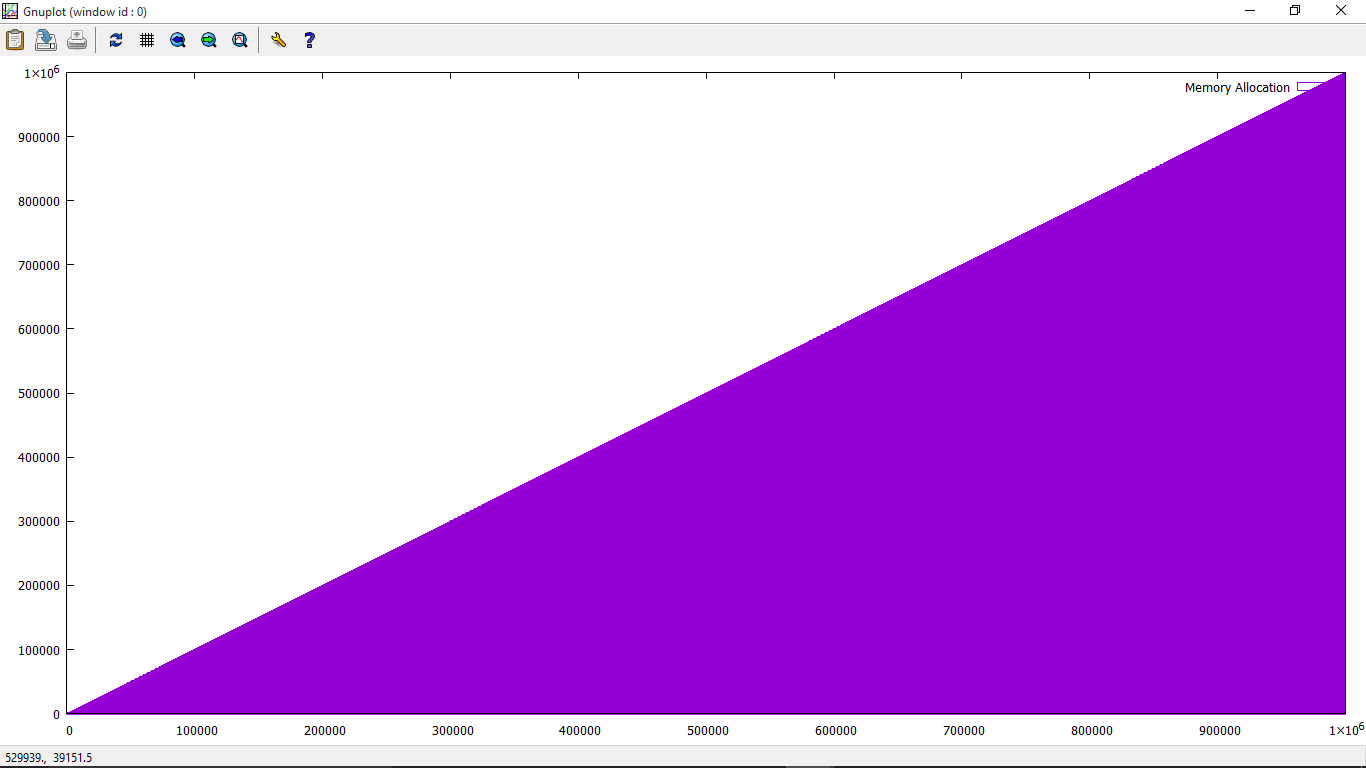


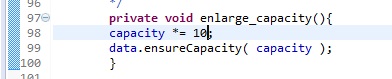
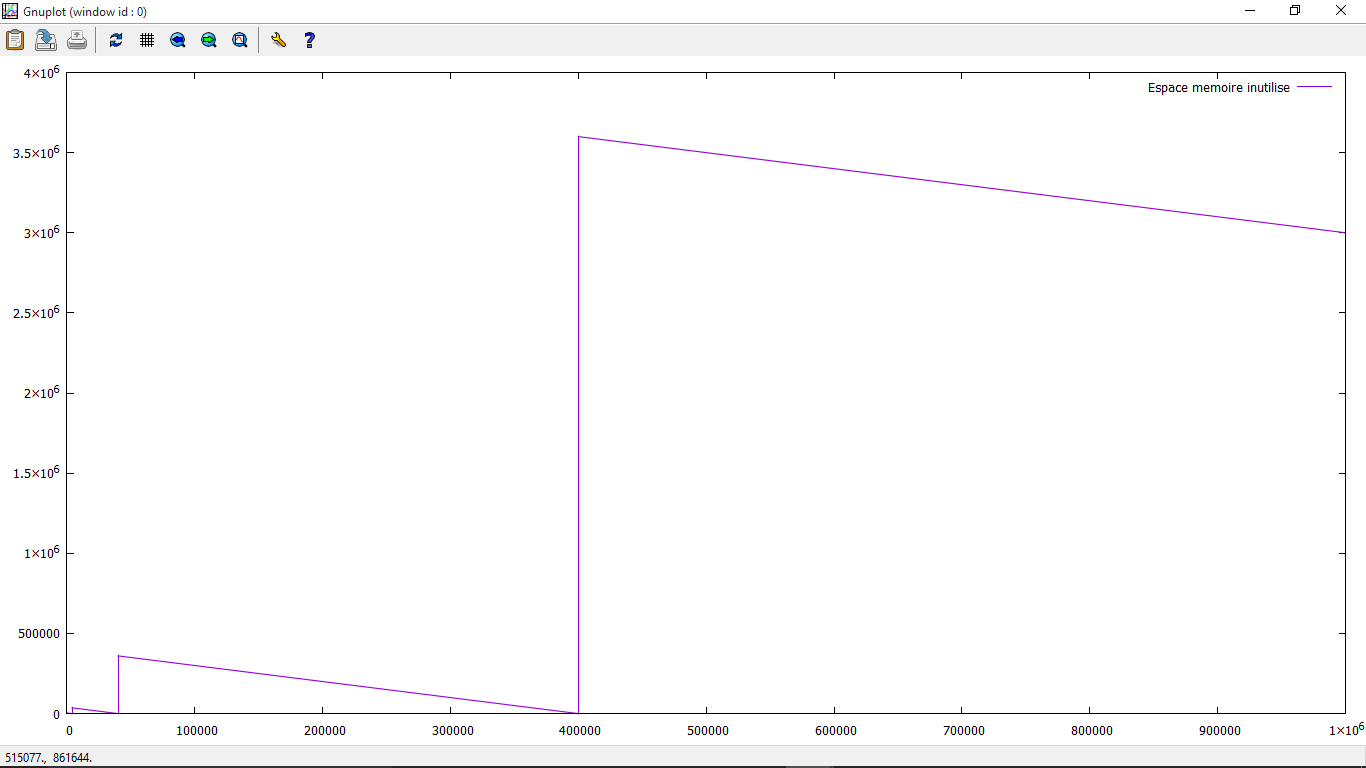
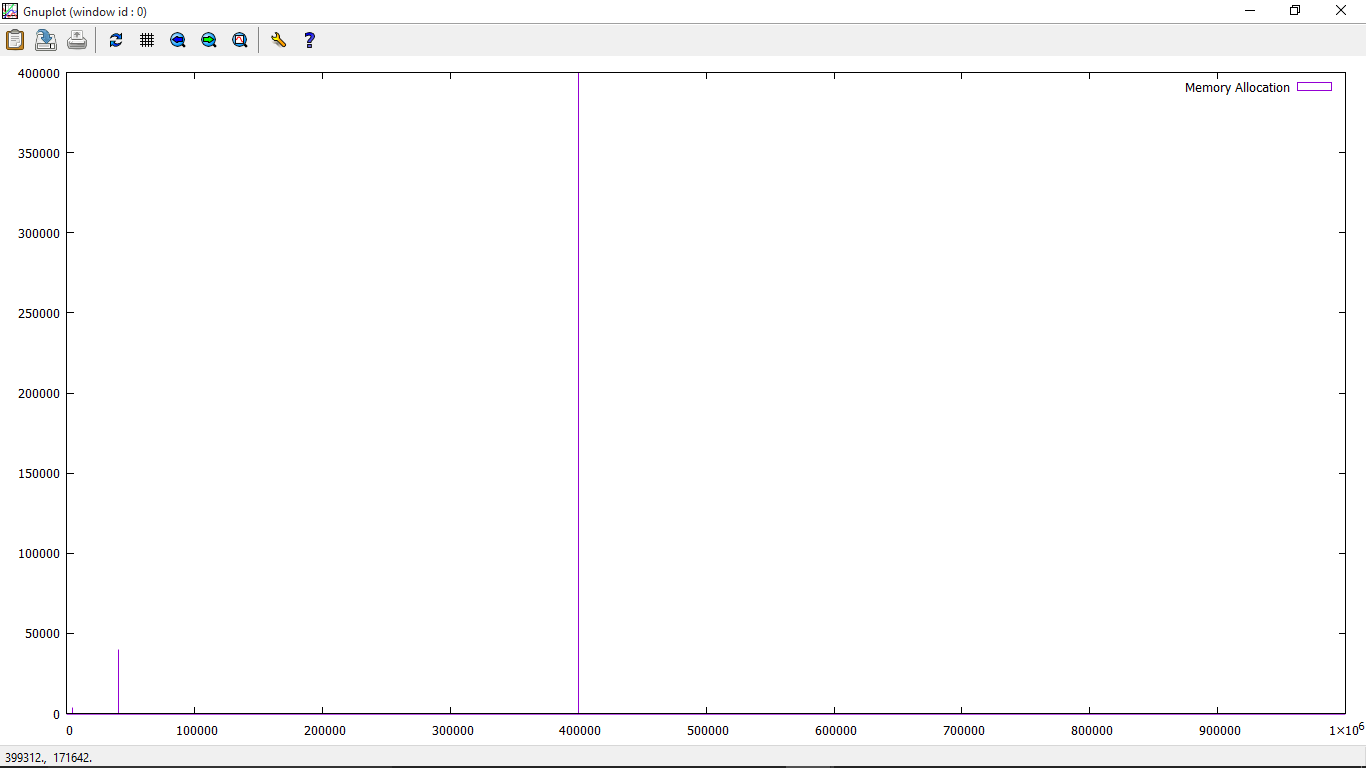
On remarque que maintenant l’espace mémoire non utilisé atteint a chaque fois 0

On remarque aussi dans le plot de l’allocation mémoire que on alloue moins de fois la mémoire étant donné que avant on allouait quand on atteignait ¾ de la capacité tandis que maintenant on attend que le tableau soit plein.

5.





Je remarque que je réalloue trop souvent quand le facteur alpha est petit.

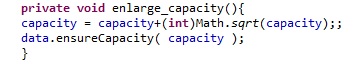
Quand le facteur alpha est trop grand l’espace mémoire non utilisé est grand.

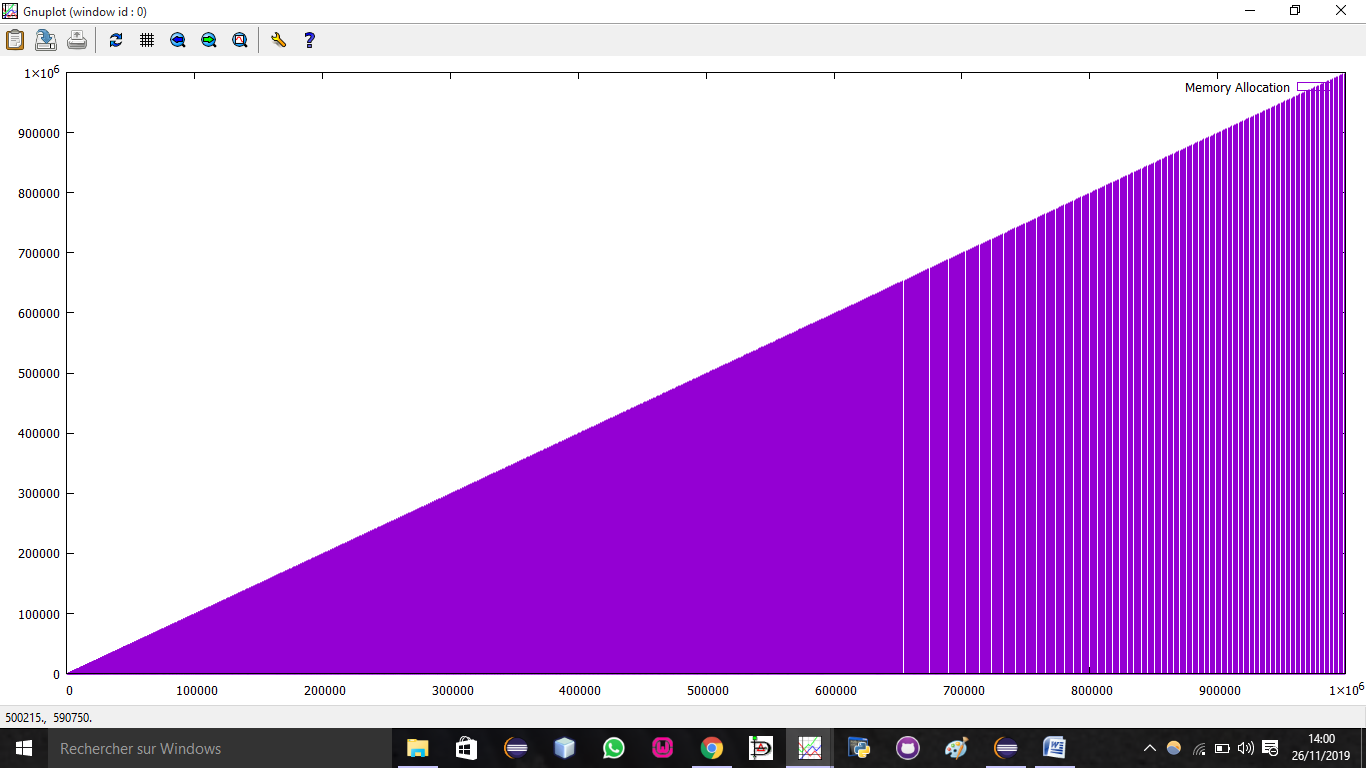
Quand on ne gaspille pas en mémoire (cout en espace) on perd du temps en calcul donc en temps (cout en temps).

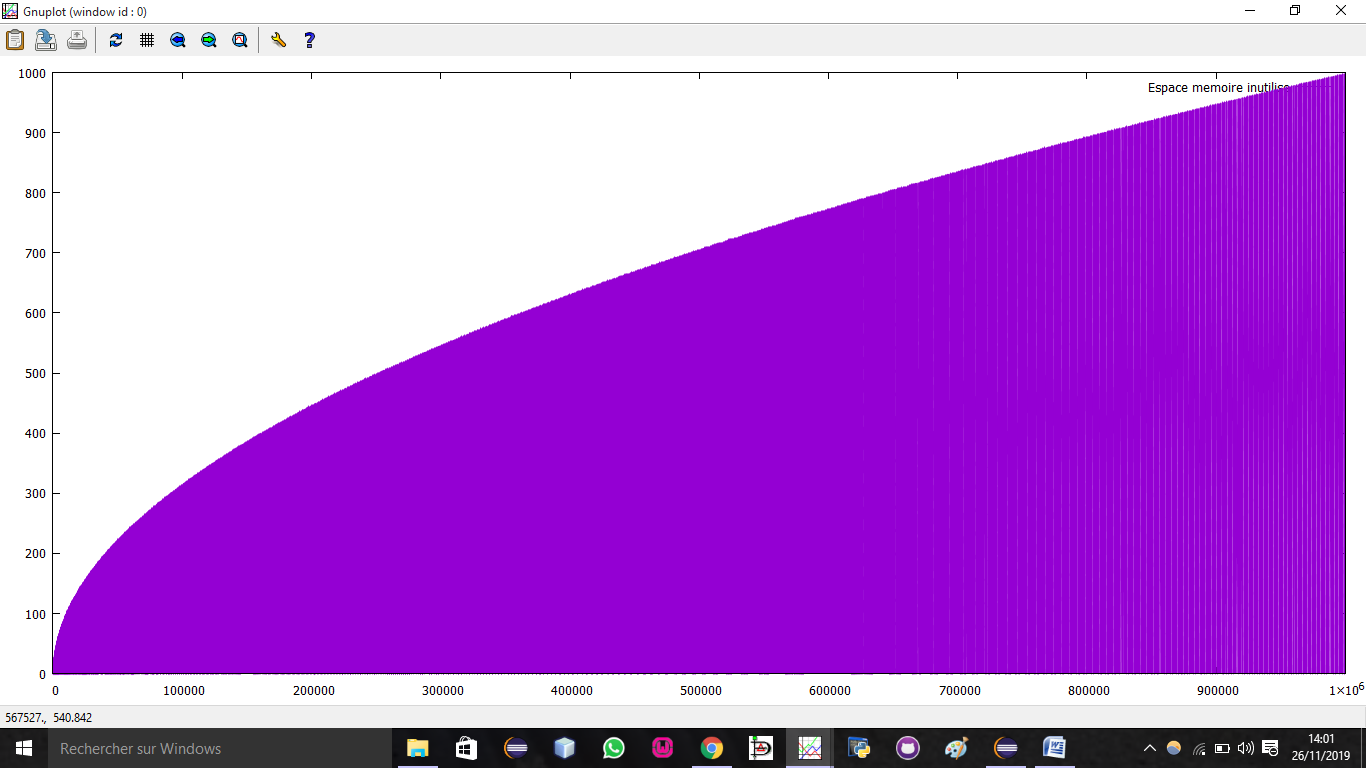
On remarque que quand on réduit la consommation de mémoire le temps augmente.

Si on veut gagner du temps on agrandit le facteur multiplicatif mais on perd en mémoire et inversement pour gagner en mémoire il faut que le facteur multiplicatif soit petit mais à chaque fois on perds du temps a réallouer de l’espace mémoire

6.







Au début on fait beaucoup d’allocations mais on observe une réduction a la fin, a force que la capacité n grandit on ajoute beaucoup d’espace quand le tableau est plein ce qui fait qu’on a plus besoin d’ajouter autant d’espace qu’avant (quand la capacité n était petite au début).