Rapport SAE 1.02 – Robineau Seiler

Algorithme logique

```
fonction adjlisT(l InOut: Liste(Chaine), c:chaine)
debut
    p <= tete(l)
    si finliste(l,p)
        alors
            adjqlis(l,c)
        sinon
            placeprec <== p
            trouve = faux
            tant que non finliste(l,p) et non trouve faire
                courant <== val(l,p)
                si courant ≥ c
                    alors
                        trouve <== vrai
                    sinon
                        placeprec <== p
                        p <== suc(l,p)
                fsi
            ftq

            si p = placeprec
                alors
                    adjtlis(l,c)
                sinon
                    adjlis(l, placeprec, c)
            fsi
    fsi
fin
```

```
fonction suplisT(l : Liste(Chaine), ch: chaine)
debut
    p <== tete(l)
    depasse <== faux
    tant que non finliste(l,p) et non depasse faire
        courant <== val(l,p)
        si courant ≥ c
            alors
                depasse <== vrai
                si courant = c
                    alors
                        suplis(l,p)
                fsi
            sinon
                p <== suc(l,p)
        fsi
    ftq
fin
```

```
fonction memlisT(l:Liste(chaine), c: chaine):booleen
debut
    p <== tete(l)
    depasse <== faux
    trouve <== faux
    tant que non finliste(l,p) et non depasse faire
        courant <== val(l,p)
        si courant ≥ c
            alors
                depasse <== vrai
                si courant = c
                    alors
                        trouve <== vrai
                fsi
            sinon
                p <== suc(l,p)
        fsi
    ftq
    retourner trouve
fin
```

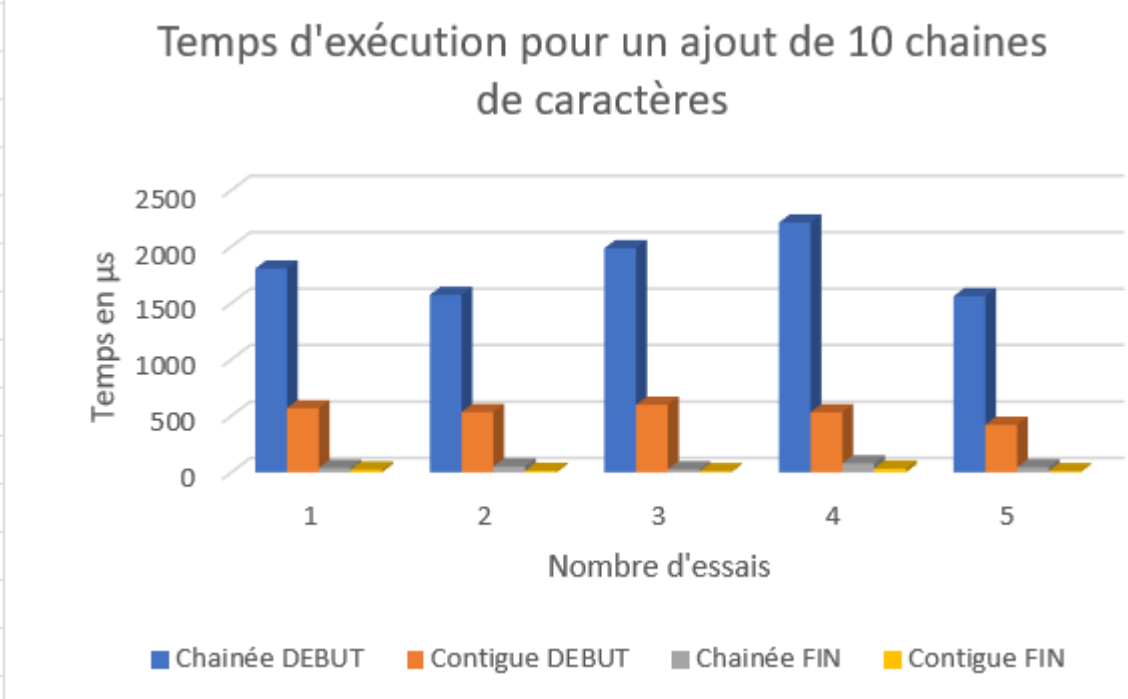
Environnement expérimental :

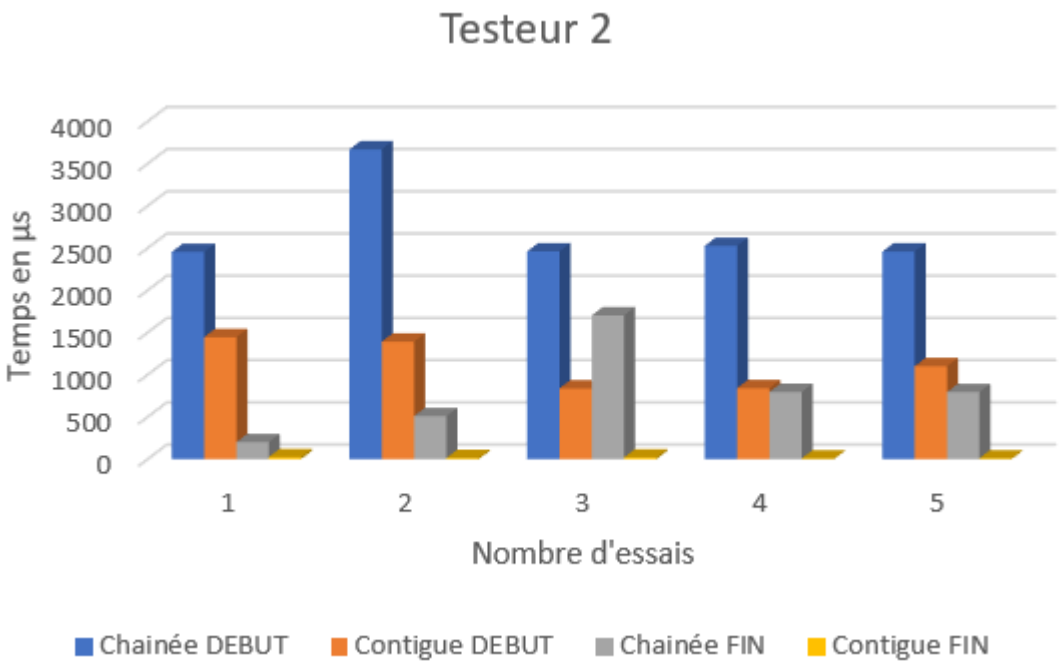
- CPU : AMD Ryzen 7 4700U
- RAM : 16Go 2666MHz

Testeur 2 :

- CPU : Intel Celeron N4000
- RAM : 4Go 2400MHz

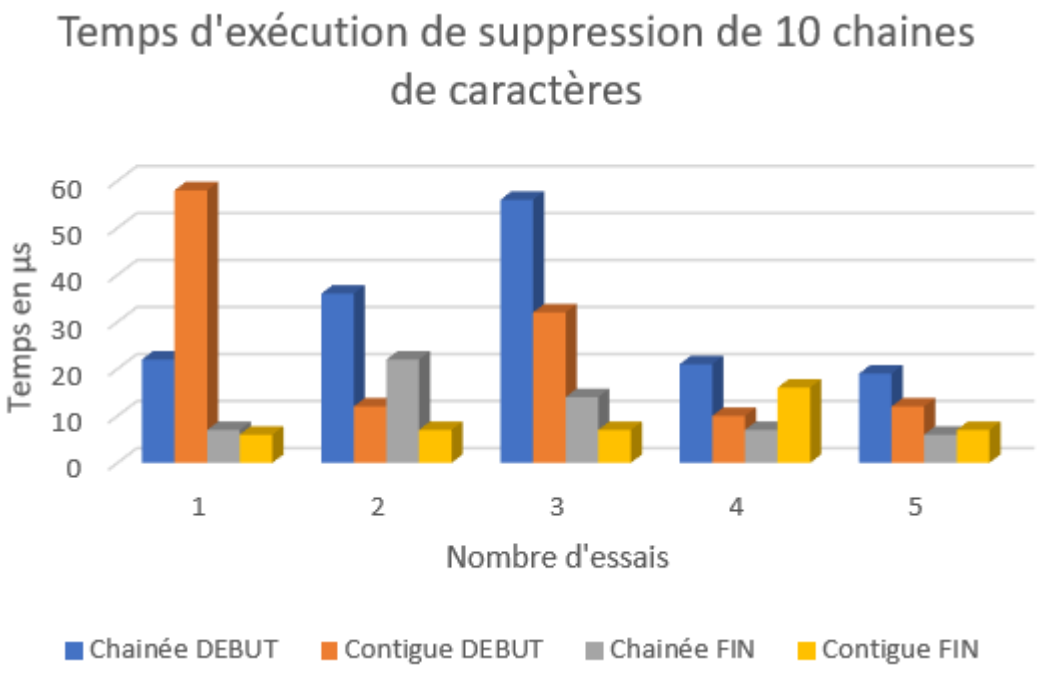
Résultats des tests de performances :

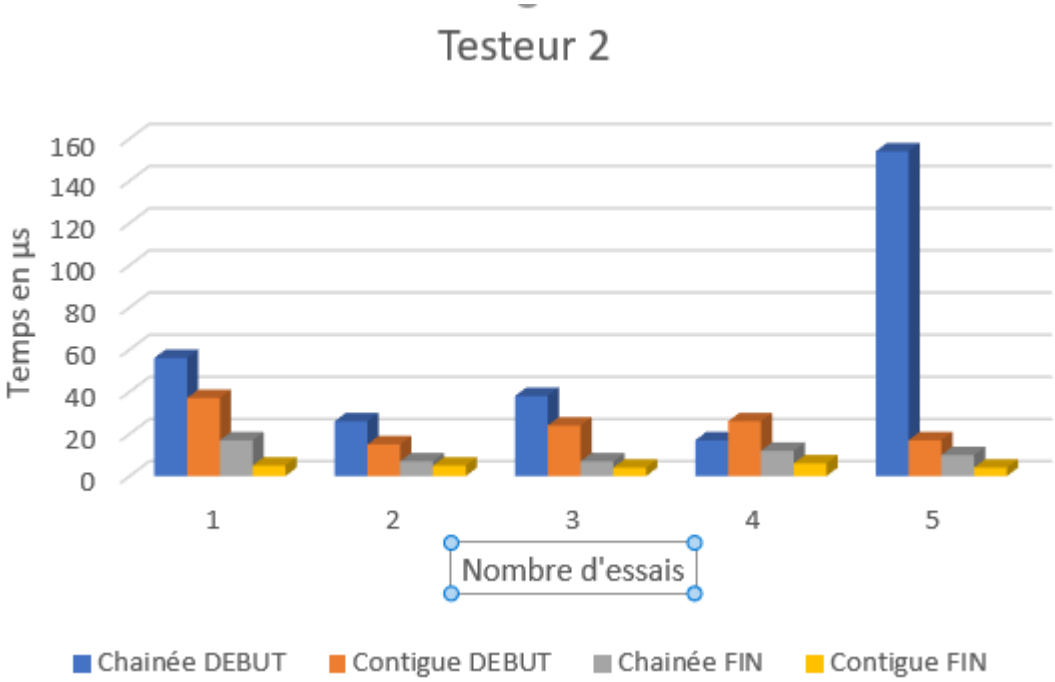




On remarque une différence frappante entre l'ajout de chaines dans une liste chainée et dans une chaine contigue. Ce qui est étonnant, c'est que l'ajout dans une liste chainée requiert plus de temps que

l’ajout dans une liste contigue, alors qu’on devrait remarquer. l’inverse. On voit également le





Les résultats sont extrêmement différents des ajouts, même si la liste chainée reste la plus couteuse en ressources, et on remarque que les chaines de fin d’alphabet consomment elles aussi plus de ressources

La mesure des temps d'exécutions s'est effectuée avec une remise à 0 entre chaque ajout / suppression.

-_-_-_-_-_-

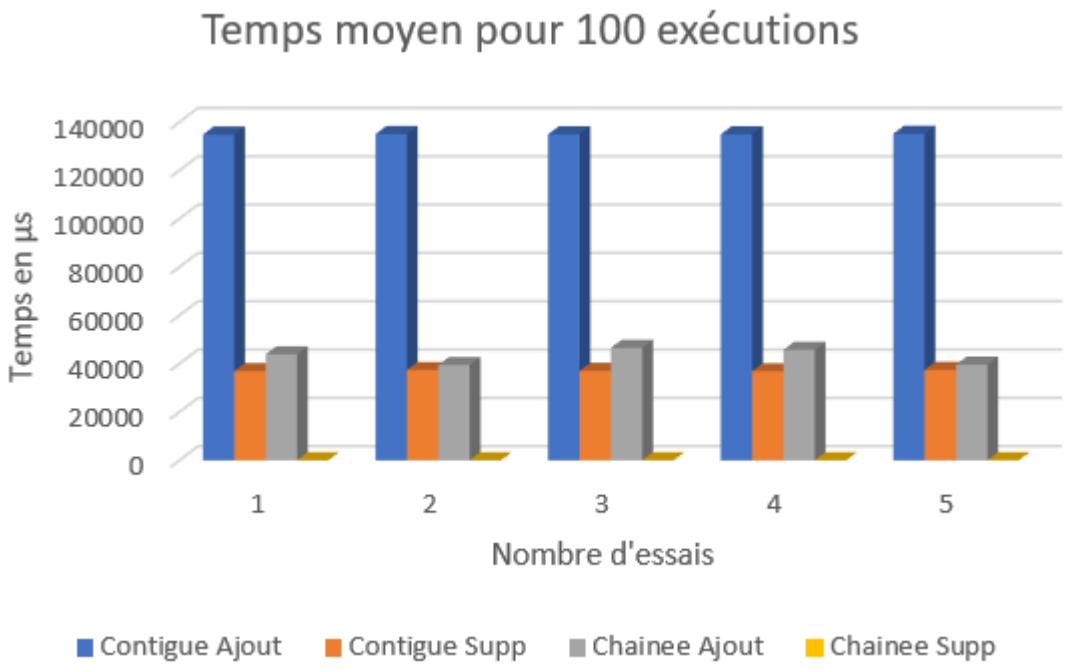
Dans TestListeTriee.java, nous avons jugé utile d'ajouter quelques tests sur l'ajout et la suppression de doublons, voir comment le méthode de la classe ListeTriee se comportaient. On a vérifié qu'en cas de doublons, la méthode suplisT supprime bien la première occurrence de la chaine passée en paramètre.

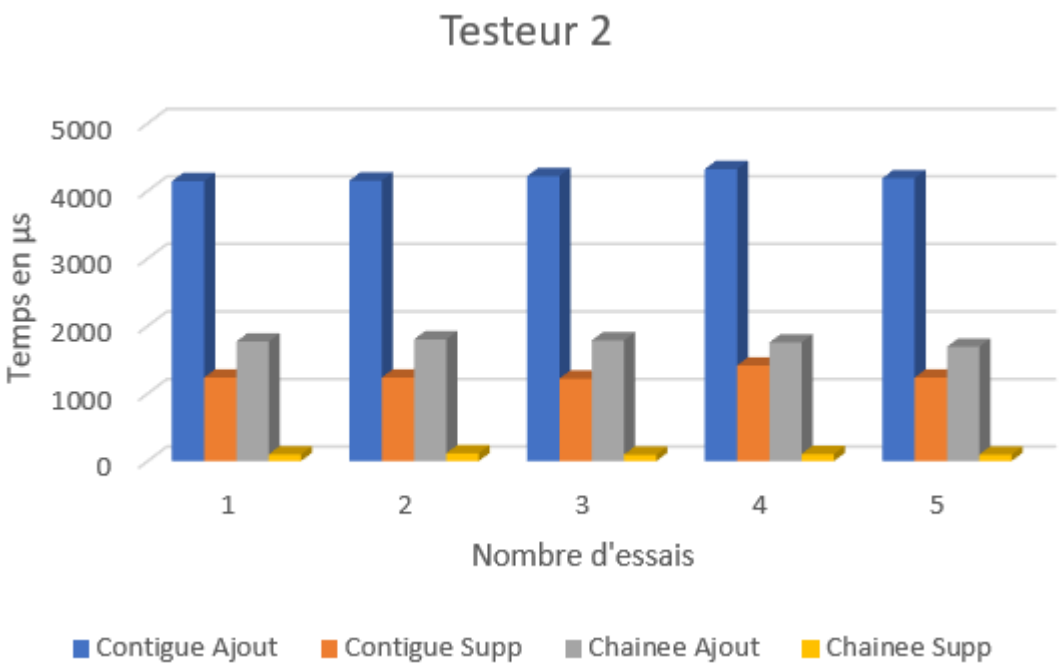
Ces tests supplémentaires sont représentés par les tests n°5,6,11

-_-_-_-_-_-

Question optionnelle 22 (10000 noms)

Après comparaison des résultats avec ceux de la classe Principale, on remarque l'effet inverse, l'ajout de chaine dans une liste chainée est beaucoup plus rapide que dans une liste contigue





En moyenne, sur 100 exécutions, la suppression dans une liste chainée est tellement rapide, que la barre jaune ne se distingue presque pas.

Question optionnelle 23

La question a été traitée comme demandée. On peut choisir le nombre d'exécutions et la taille des listes chainées et contigues (voir LISEZMOI.md

-_-_-_-_-_-

Conclusion : Lors de l'utilisation de très longues listes (+ de 10 000 places), La liste chainée reste avantageuse en ajout comme en suppression, cela est dû au fait que l'ajout dans une liste chainee requiert de décaler toutes les valeurs vers la droite. Mais lorsque l'on travaille sur des courtes listes (10 places) la suppression sur les listes contigues sont plus rapide que les listes chainées.

Finalement, si on travaille sur des longues listes, il est recommandé d’utiliser des listes chainées, sinon il faut utiliser une liste contigue.