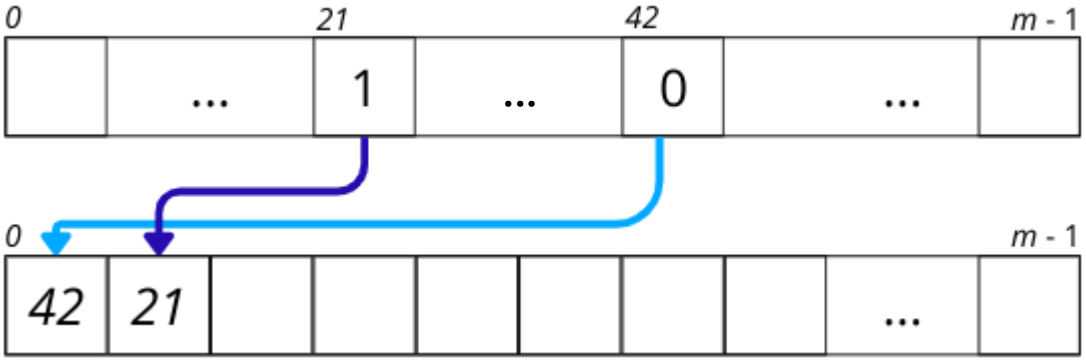
# SAE 3.6
# Algorithmique
## Dummy array




Clément BENONY
Gurvan DUMARCHAT

---

# Table des matières:



# Table des images:

# Présentation

Le but de cette étude est d'analyser les contraintes techniques, le comportement ainsi que les performances d'une structure de données, nommée: dummy array.

# La structure de donnée

Le principe de cette structure est de permettre la manipulation en temps constant sur trois opérations fondamentales : l'insertion de données, la recherche de données, la suppression de données.
Pour appliquer ce principe, deux tableaux doivent être initialisés à la taille $m - 1$, avec $m$ la valeur maximale à stocker dans la structure.

Cette structure possède deux composantes principales :

- Un tableau d'indices : Ce tableau contient les indices des valeurs correspondantes dans le second tableau.
- Un tableau de valeurs : Ce tableau contient les valeurs stockées et accessibles par la structure.

La structure contient également 2 valeurs cruciales pour le bon fonctionnement :

- La valeur maximale que peut stocker la structure ($m$).
- Le compteur d'éléments déjà insérés.

# Protocole d'implémentation

Pour l'implémentation de cette structure nous allons utiliser le langage de programmation C, connu pour sa vitesse d'exécution et un contrôle fin de la mémoire des machines. Ce choix est également pertinent dans le cas d'une étude algorithmique car le langage est dit de "bas-niveau" garantissant un contrôle presque total sur les performances de l'algorithme et sur les étapes exécutées par le programme.

Pour évaluer les performances, nous utiliserons le principe de banc d'essai (benchmark en anglais) permettant d'exécuter des tâches et de prendre des mesures pendant l'exécution. Pour ce benchmark, nous prendrons les mesures en temps. L'idée principale est d'exécuter des mesures sur toutes les opérations à la suite en faisant varier la taille de la structure ($m$) à la fin de la mesure complète de l'ensemble des opérations disponibles.

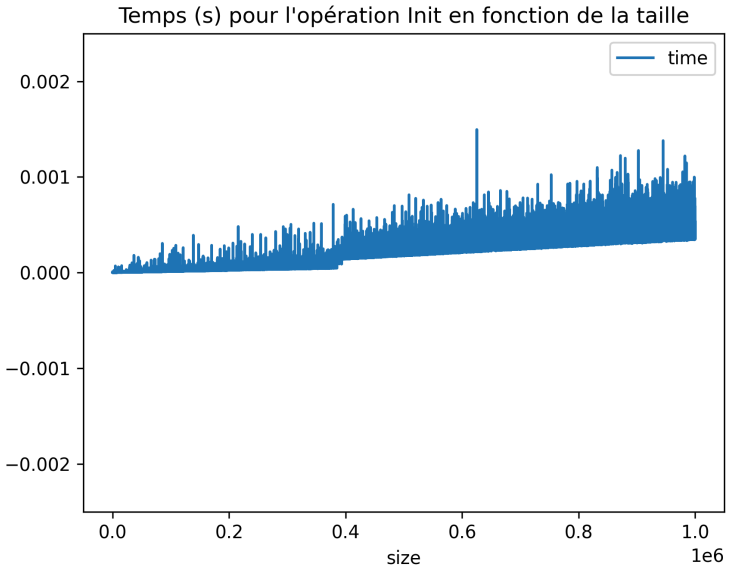
L'algorithme suivant illustre ce principe :

```
Action benchmark():
Début
    DummyArray dummy
    benchmarkInitialisation(ES dummy,m)
    benchmarkInsertion(ES dummy,m)
    benchmarkRecherche(ES dummy,m)
    benchmarkSuppression(ES dummy,m)
    libererMemoire(ES dummy)
Fin
```

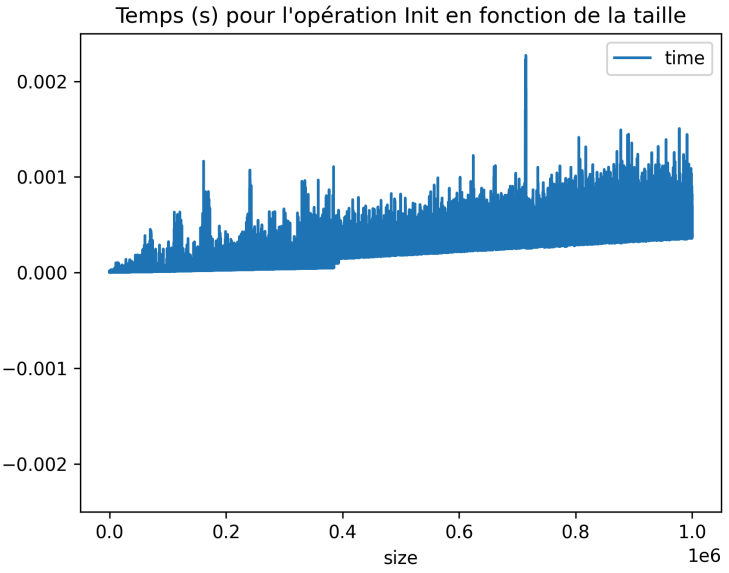
# Des essais pour chaque fonction

Chaque opération est mesurée de la même façon, elle est répétée *k* fois, variant en fonction de $m$. Si $m$ est inférieur ou égal à 100 alors $k = 10$, sinon $k = 100$.
Un grand nombre de mesures est nécessaire pour chaque opération car sa performance peut être influencée par de nombreux facteurs extérieurs liés à des processus en exécution sur la machine (ex : une requête réseau, une interruption périphérique…), alors la mesure est exécutée fois puis une moyenne est calculée pour approcher la valeur réelle du coût de l'opération. Cette quantité d'opérations est également intéressante dans la mesure où l'on teste également les performances de la structure en présence de données déjà existantes. La mesure en temps est effectuée avant le début de l'opération puis dès son accomplissement.

# Résultats

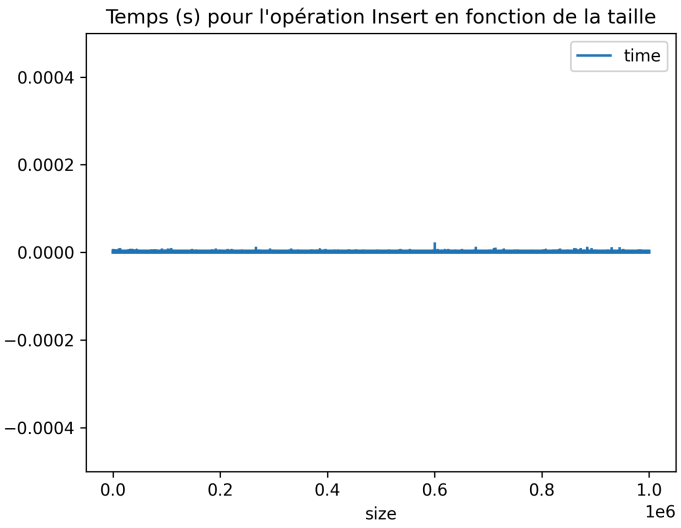


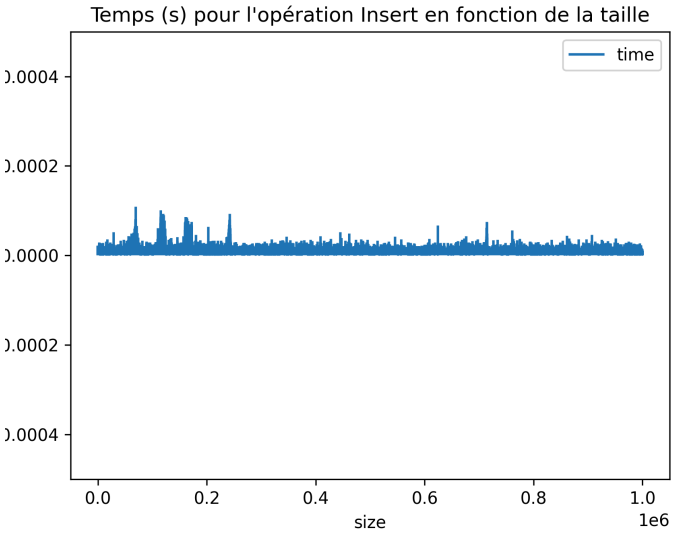

Avec une activité réservée au benchmark




Avec une activité forte de la machine

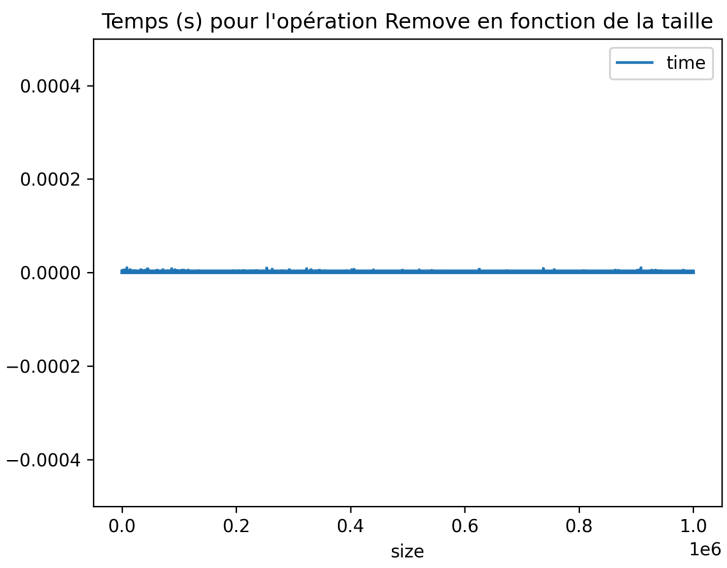
Sur ces graphiques nous pouvons constater que l'**allocation en mémoire** est croissante et à une complexité en mémoire de **O(**$log(n)$**)** ce qui est très favorable pour une grande structure de données.
Bien que cela soit considéré comme favorable, il est essentiel de préciser le contexte d'utilisation. La croissance logarithmique est généralement positive par rapport à une croissance linéaire ou quadratique. Toutefois, l'allocation initiale est de taille $m - 1$et si $m$ est très grand, des problèmes de fragmentation ou de limites de mémoire pourraient survenir. Une évaluation plus approfondie des coûts en mémoire, surtout pour des cas d'utilisation réels avec des données dynamiques, serait bénéfique.

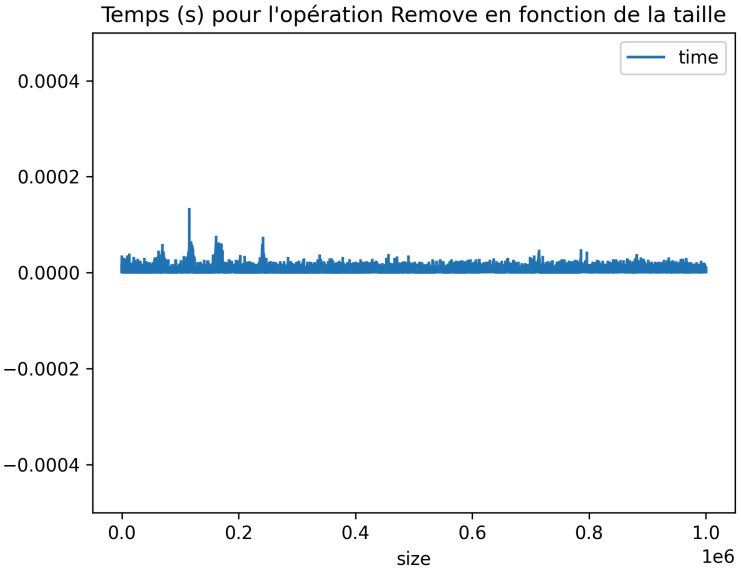



Avec une activité réservé au benchmark | Avec une activité forte de la machine

D’après ces graphiques nous pouvons constater que le coût d’une opération d’**insertion** est constant, soit une complexité de **O(1)** dans le meilleur cas comme dans le pire cas car on ajoute toujours les valeurs à la fin du tableau.

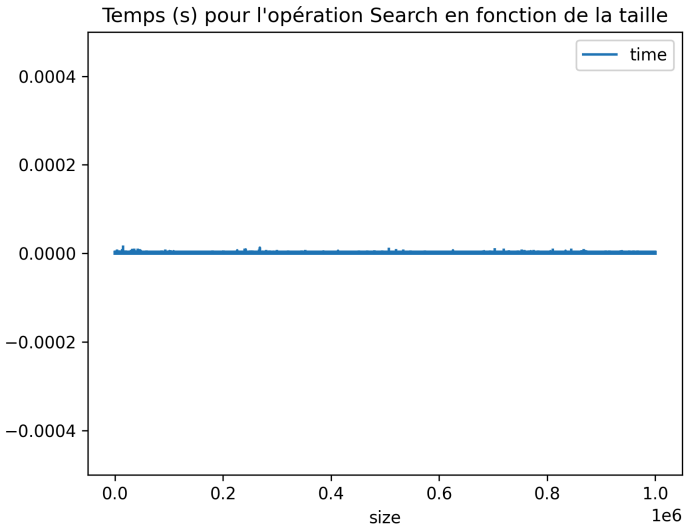


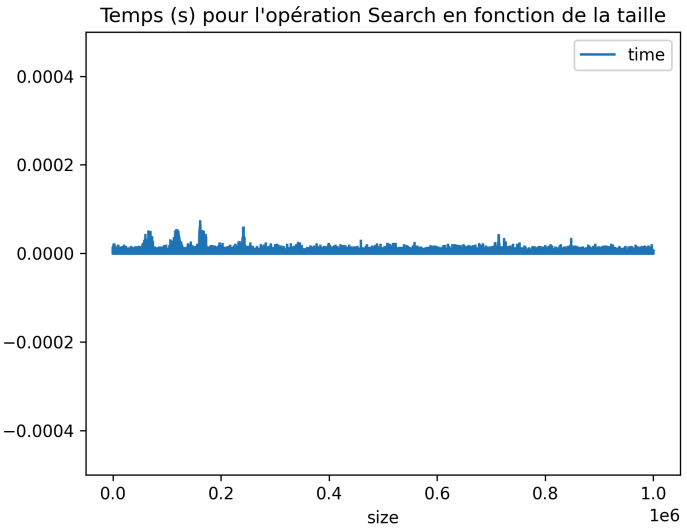




Avec une activité réservé au benchmark

Avec une activité forte de la machine

En observant ces graphiques, nous constatons que le coût d'une opération de **suppression** d'un élément est constant, soit une complexité de **O(1)**




Avec une activité réservé au benchmark

Avec une activité forte de la machine

Sur ces graphiques, nous pouvons constater que le coût d'une opération de **recherche** d'un élément est constant, soit une complexité de **O(1)**

# Discussion

## Performances des opérations

Les performances des opérations d'insertion, de suppression et de recherche sont toutes déclarées constantes, avec une complexité de O(1). Cela semble idéal en théorie, mais il est crucial de considérer les scénarios pratiques. Par exemple, bien que l'insertion soit constante dans le cas d'ajouts à la fin du tableau, cela suppose que la taille maximale de mmm n’est pas dépassée. Une fois la capacité atteinte, des stratégies de redimensionnement pourraient nécessiter des opérations plus coûteuses. De plus, les performances pourraient se dégrader si des mécanismes de gestion des collisions ou des réajustements sont nécessaires.

## Répétition et moyennage des mesures

Le protocole de mesure a été soigneusement conçu pour tenir compte de divers facteurs externes pouvant influencer les résultats. Cependant, il reste à savoir dans quelle mesure ces facteurs peuvent réellement impacter les performances dans un environnement de production. Par exemple, la répétition des opérations et le calcul des moyennes aident à atténuer l'influence des interruptions, mais une évaluation dans des scénarios concurrents ou avec des processus plus variés serait pertinente.

# Conclusion

En conclusion, les résultats obtenus mettent en avant les atouts de la structure de données "dummy array" en termes de performances théoriques. Cependant, une attention particulière doit être accordée à la gestion de la mémoire et aux conditions réelles d’utilisation pour confirmer la viabilité de cette structure dans des applications pratiques. Des études supplémentaires, incluant des tests en conditions réelles et des analyses des scénarios limites, seraient bénéfiques pour valider ces résultats et garantir une robustesse dans des environnements variés.