Titre: **Analyse de l'efficacité de différents algorithmes de tri**

Question de recherche:

Comment l'efficacité d'un algorithme de tri à bulles se compare-t-elle à celle d'un algorithme de tri par sélection et de tri par insertion pour les pires, moyens et meilleurs scénarios?

3921 mots

TABLE DE MATIÈRES:

INTRODUCTION:

Les algorithmes de tri jouent un rôle important dans notre vie de tous les jours à travers notre utilisation des logiciels. Divers détaillants en ligne tels que Amazon ou eBay utilisent des algorithmes de tri afin de trier les articles disponibles à l'achat en fonction de critères spécifiques (ex: du prix le plus bas au prix le plus élevé, du plus populaire au moins populaire, etc.) Ils sont également utilisés par les moteurs de recherche, tels que Google Chrome, pour fournir à l'utilisateur des sites web à visiter triés selon la pertinence en lien avec les mots-clés entrés. On les trouve également dans l'industrie du divertissement ; les services de streaming comme Netflix ou YouTube mettent en œuvre des algorithmes de tri pour organiser les films par pertinence, popularité ou nombre de visionnages. En effet, de nombreuses entreprises emploient les algorithmes de tri pour optimiser leur rendement et leurs services. Ces dernières doivent donc déterminer lesquels parmi les algorithmes de tri seraient les plus efficaces, afin d'obtenir un meilleur rendement et une expérience plus conviviale pour le consommateur se traduisant par davantage de ventes pour l'entreprise qui utilise l'algorithme. Par exemple, le plus rapidement un consommateur est capable d'identifier le produit qu'il aimerait acheter, le plus probable que celui-ci achèterait le produit. En outre, plus les résultats obtenus par un moteur de recherche sont affichés rapidement et sont pertinents aux mots-clés, plus l'utilisateur sera satisfait de l'expérience et aura tendance à réutiliser le service. , Il est donc important de tester l'efficacité des algorithmes de tri dans chaque cas spécifique. Dans ce mémoire, on va comparer trois algorithmes de tri : les algorithmes de tri à bulle, de tri sélectif et de tri par insertion, pour les pires, moyens et meilleurs scénarios afin de mieux comprendre comment des entreprises peuvent évaluer les divers algorithmes et déterminer lesquels sont les plus efficaces.

THÉORIE:

ALGORITHME DE TRI:

Avant de commencer cette expérience afin de déterminer l'efficacité algorithmique des trois algorithmes de tri, une bonne compréhension des algorithmes de tri, de leurs rôles et de leur fonction est nécessaire. Les algorithmes de tri sont un ensemble d'instructions qui prennent un tableau de données et classent les éléments dans un ordre particulier. Les données saisies peuvent être organisées de la manière que le développeur souhaite: par ordre croissant, décroissant, alphabétique, etc. Par exemple, si les valeurs saisies sont 5, 2, 4, 1, 3 dans cet ordre spécifique et que le développeur souhaite les classer par ordre croissant, l'algorithme de tri donnera 1, 2, 3, 4, 5. Comme le tri peut souvent réduire la complexité d'un problème, il s'agit d'un algorithme important en informatique. Ces algorithmes ont des applications directes dans les algorithmes de recherche, les algorithmes de base de données, les méthodes "diviser pour régner", les algorithmes de structure de données, et beaucoup d'autres (freeCodeCamp). Les algorithmes de tri ont un objectif simple, mais de nombreuses approches variées et différents algorithmes peuvent être utilisés pour atteindre cet objectif. Quelques-uns de ces algorithmes sont le tri à bulles, le tri sélectif et le tri par insertion. Ces trois algorithmes sont connus sous le nom d'algorithmes de tri par comparaison. Les tris par comparaison sont des algorithmes de tri qui ne lisent les éléments de la liste qu'à travers une seule opération de comparaison abstraite, par exemple, avec un opérateur "inférieur ou égal à" (≤) ou une comparaison à trois) qui détermine lequel de deux éléments doit apparaître en premier dans la liste triée finale (Wikipedia).

ALGORITHME DE TRI À BULLES:

Le premier algorithme de tri qui sera analysé dans ce mémoire est l'algorithme de tri à bulles. Cet algorithme fonctionne en intervertissant la position de 2 éléments l'un à côté de l'autre dans un tableau s'ils ne sont pas dans le bon ordre. Le diagramme ci-dessous montre la façon

dont un algorithme de tri à bulles fonctionnerait avec un ensemble de données de 2, 3, 1, 4 dans cet ordre spécifique.

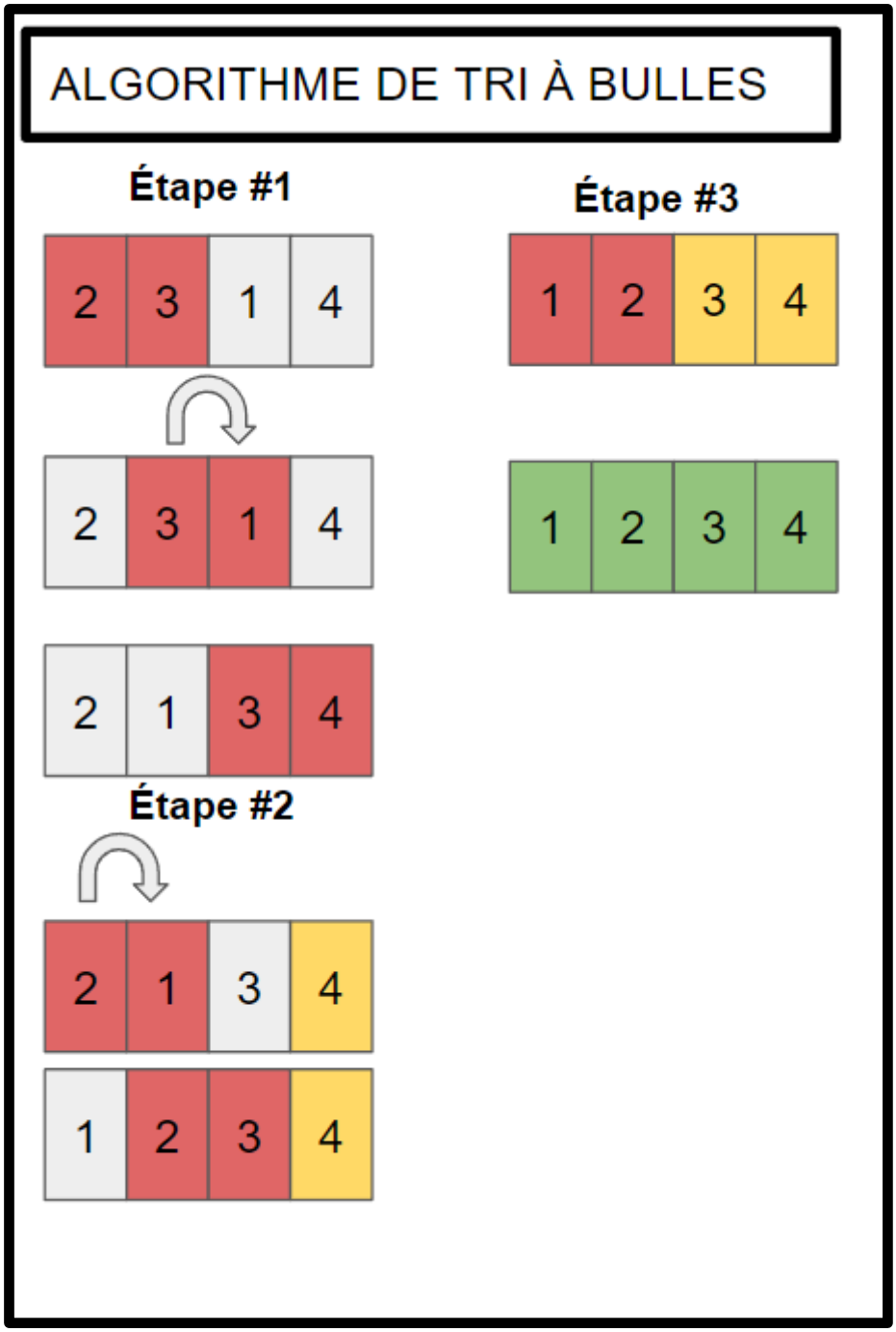


Figure 1 (J'ai créé ce diagramme moi-même en utilisant le logiciel Google Slides)

Dans le diagramme ci-dessus, les cases rouges représentent les valeurs qui sont vérifiées par l'ordre, si elles sont dans le bon ordre, l'élément suivant sera comparé à l'élément qui le suit (comme indiqué dans la séquence originale où 2 et 3 sont à l'intérieur d'une boîte rouge). S'il y a une flèche au-dessus des deux cases rouges, cela indique qu'ils doivent changer de place

car ils sont dans un ordre incorrect (comme le montre la deuxième version de l'ensemble de données où 3 et 1 sont dans un ordre incorrect). Ce processus se répète jusqu'à ce que toutes les valeurs soient triés dans l'ordre voulu (dans le cas du diagramme ci-dessus, il s'agit de l'ordre croissant). De plus, on peut voir qu'après l'étape 1, lorsqu'on atteint la fin de la liste, on revient de nouveau aux 2 premiers éléments, qui sont maintenant différents puisque les 2 premiers éléments originaux ont changé de place à l'étape 1, (par exemple dans le diagramme ci-dessus on peut voir que dans l'étape 1, les deux premières valeurs étaient 2 et 3 mais dans l'étape #2 elles sont 2 et 1) et on compare une fois de plus en répétant le même processus jusqu'à ce que tous les éléments soient dans l'ordre désiré.

Le code d'un algorithme de tri à bulles en C++ ressemblerait à ce qui suit (GeeksforGeeks):

```
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
}
```

ALGORITHME DE TRI SÉLECTIF:

Le deuxième algorithme qui sera analysé dans ce mémoire est l'algorithme de tri sélectif. La façon dont celui-ci fonctionne est qu'il identifie la valeur la plus basse du tableau et l'échange avec la valeur au début du tableau, puis trouve la deuxième valeur la plus basse et la place à la deuxième place et ainsi de suite jusqu'à ce que le tableau soit trié. Lorsqu'on utilise un tri sélectif, il y a deux sous-tableaux : le sous-tableau déjà trié et le sous-tableau restant qui n'est pas trié (initialement, le tableau complet n'est pas trié). Après chaque itération, la valeur la

plus basse du sous-tableau non trié sera placée à la fin du sous-tableau trié. La figure suivante démontre comment le tri sélectif fonctionne avec un tableau contenant les valeurs 4, 3, 2, 1.

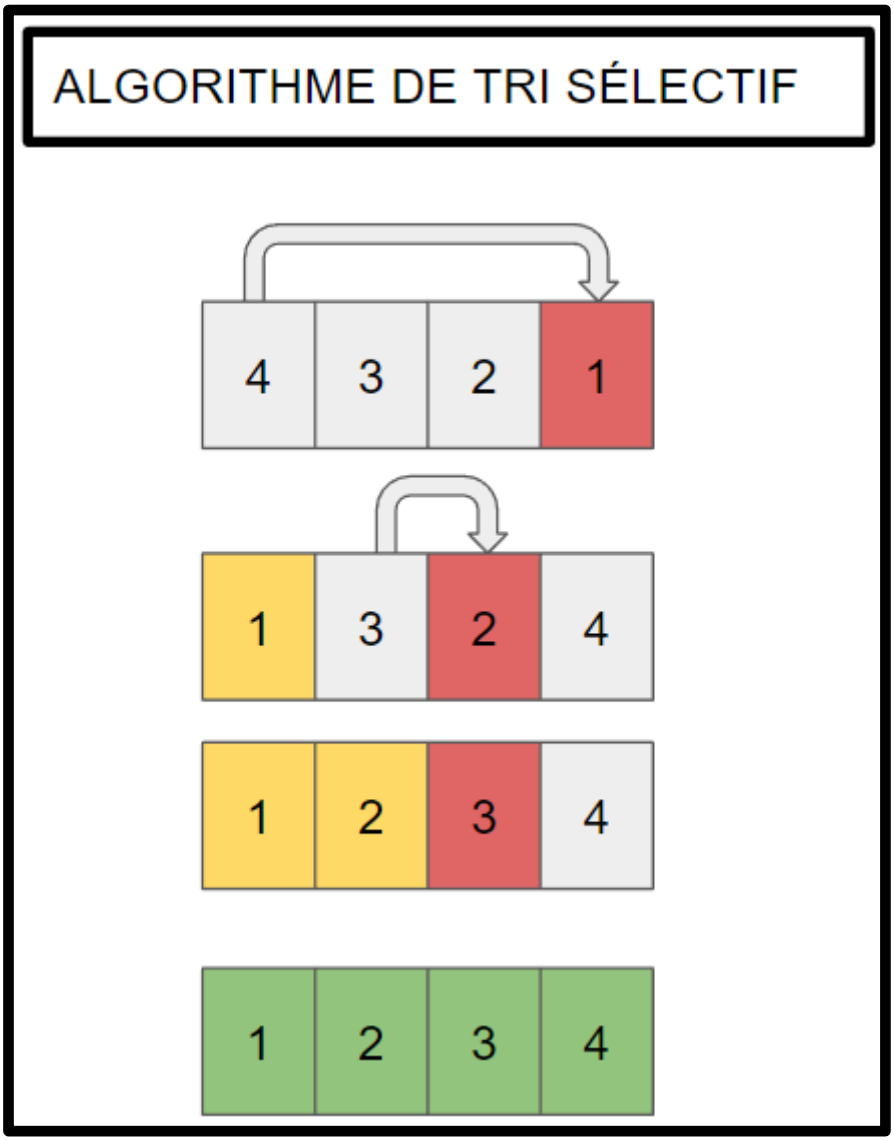


Figure 2 (J'ai créé ce diagramme moi-même en utilisant le logiciel Google Slides)

Ce diagramme simplifié montre, de manière très fondamentale, comment fonctionne un algorithme de tri sélectif. Tout d'abord, la première valeur du tableau est comparée aux autres valeurs du tableau pour déterminer laquelle de ces valeurs est la plus petite. Ensuite, lorsque la plus petite valeur est déterminée, la valeur initiale et la plus petite valeur sont échangées. Dans le schéma ci-dessus, les valeurs 4 et 1 sont échangées. Aussi, les valeurs en rouge sont les valeurs qui sont les plus petites parmi le tableau et sont celles qui sont échangées avec l'autre valeur. Ensuite, les valeurs jaunes représentent la partie triée du tableau et le reste est

la partie non triée. Les étapes se répètent encore une fois pour la partie non triée du tableau jusqu'à ce que tout le tableau soit trié.

Le code d'un algorithme de tri sélectif en C++ ressemblerait à ce qui suit (GeeksforGeeks):

```
void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;
    for (i = 0; i < n-1; i++)
    {

        min_idx = i;
        for (j = i+1; j < n; j++)
        if (arr[j] < arr[min_idx])
            min_idx = j;
        if(min_idx!=i)
            swap(&arr[min_idx], &arr[i]);
    }
}
```

ALGORITHME DE TRI PAR INSERTION:

Le troisième algorithme qui sera analysé dans ce mémoire est l’algorithme de tri par insertion. Cet algorithme trie les données en divisant le tableau en parties triées et non triées, puis les valeurs des parties non triées sont sélectionnées et placées à la bonne position dans la partie triée. Le fonctionnement d'un algorithme de tri par insertion est le suivant : la première valeur du tableau est supposée être triée et la deuxième valeur est stockée dans une variable. La valeur qui est stockée est comparée au premier élément. Si le premier élément est supérieur à la variable, cette dernière est placée devant le premier élément. Si ce n'est pas le cas, alors la variable gardera sa place après la première valeur puisqu'elle est plus grande.Une fois les deux premiers éléments sont triés, la troisième valeur est comparée aux deux valeurs qui la précèdent. Elle est ensuite placée derrière l'élément qui lui est inférieur. S'il n'y a pas d'élément plus petit que celle-ci, elle est placée au début du tableau. Le même processus est répété jusqu'à ce que toutes les valeurs soient triées.

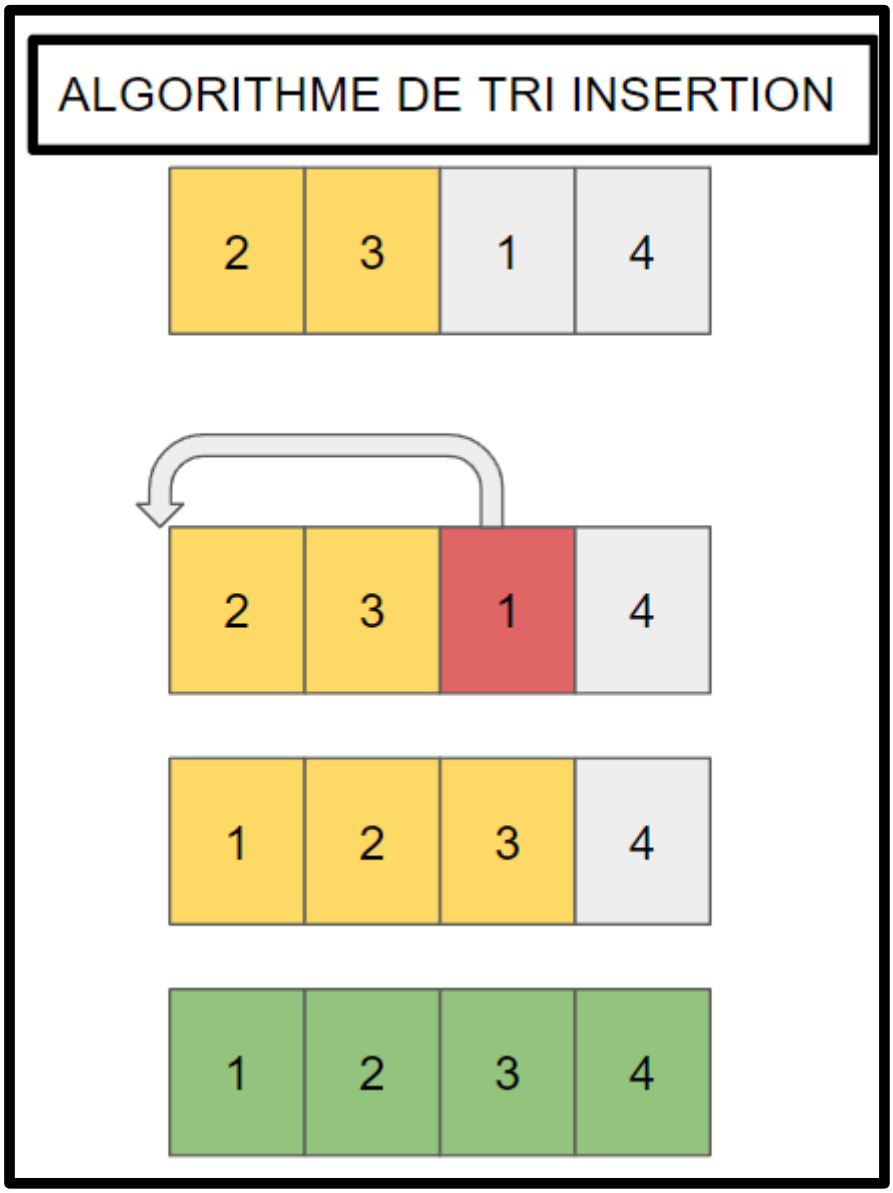


Figure 3 (J'ai créé ce diagramme moi-même en utilisant le logiciel Google Slides)

Le diagramme simplifié ci-dessus montre le fonctionnement d'un algorithme de tri par insertion. Chaque élément est comparé à l'élément situé à sa gauche et s'ils sont dans le bon ordre, les deux éléments suivants sont vérifiés. Les 2 valeurs 3 et 4 sont en jaune car elles sont déjà triées. Mais s'il y a une valeurdans la liste qui n'est pas triée, elle est insérée dans la bonne position par rapport aux autres éléments qui sont déjà triés. Par exemple, la valeur1, qui est affichée en rouge, n'est pas triée, donc l'algorithme la placera à sa place comme le montre la rangée dans le diagramme. Ce processus est répété jusqu'à ce que la liste soit triée.

Le code d'un algorithme de tri par insertion en C++ ressemblerait à ce qui suit (GeeksforGeeks):

```
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

NOTATION ASYMPTOTIQUE:

Les notations asymptotiques sont les notations mathématiques utilisées pour décrire le temps d'exécution d'un algorithme lorsque l'entrée tend vers une valeur particulière ou une valeur limite. Il y a 3 notations principales pour la complexité temporelle, la notation *Big O* pour les pires scénarios, la notation *Omega* pour les meilleurs scénarios et la notation *theta* pour les scénarios moyens. (Khan Academy) Le tableau suivant démontre la notation asymptotique pour les trois algorithmes et les trois cas.

	Notation Big O (pire scénario)	Notation Theta (scénario moyen)	Notation Omega (meilleur scénario)
Tri à bulles	$O(n^2)$	$O(n^2)$	O(n)
Tri sélectif	$O(n^2)$	$O(n^2)$	$O(n^2)$
Tri par insertion	$O(n^2)$	$O(n^2)$	O(n)

BUT:

L'objectif de ce mémoire est d'analyser les algorithmes de tri à bulles, de tri par sélection et de tri par insertion en termes de complexité temporelle pour les scénarios du meilleur cas, du pire cas et du cas moyen.

HYPOTHÈSE:

L'objectif de ce mémoire est d'analyser les algorithmes de tri à bulles, de tri par sélection et de tri par insertion en termes de complexité temporelle pour les scénarios du meilleur cas, du pire cas et du cas moyen. L’algorithme de tri à bulles sera probablement le plus lent parmi les trois algorithmes, quel que soit le scénario, simplement en raison de son mode de fonctionnement. Cet algorithme compare deux valeurs, puis les intervertit et continue de le faire jusqu'à ce que la plus grande valeur se trouve à la fin de la liste. Il recommence alors ce processus jusqu'à ce que toute la liste soit triée. Dans l'ensemble, il s'agit d'un moyen moins efficace de trier une liste. Il serait aussi raisonnable d’estimer que les différents scénarios auront un effet sur le temps de tri, car les algorithmes de tri à bulles ont un temps d'exécution linéaire pour les listes qui sont déjà triées (meilleur scénario) mais ont un temps d'exécution quadratique pour les listes qui sont triées dans un ordre décroissant (pire scénario dans le cas où on veut trier par ordre décroissant). L’algorithme de tri par insertion serait plus rapide

mais plus lent que le tri de sélection parce que plus d'échanges sont effectués dans le tri d'insertion pour trier l'algorithme. Cependant, ici, le scénario ne devrait pas avoir un effet sur le temps de tri car pour tous les scénarios, cet algorithme a un temps d'exécution quadratique. Enfin, pour le tri sélectif, j'émets l'hypothèse qu'il sera le plus rapide parmi les trois algorithmes pour les trois scénarios et qu'il sera également plus rapide pour le meilleur scénario par rapport aux autres scénarios puisque pour tous les scénarios, il a une complexité temporelle quadratique sauf pour le meilleur scénario où elle est linéaire.

SÉLECTION DES VARIABLES:

VARIABLE INDÉPENDANTE:

La variable indépendante pour cette analyse est les scénarios dans lesquels les données sont censées être triées, ou en d'autres termes, l'ordre original des données qui doivent être triées. Il y aura trois scénarios différents: le meilleur scénario, où les données sont déjà triées par ordre croissant, le scénario moyen, où les données sont présentées dans un ordre aléatoire, et le pire scénario, où les données sont triées par ordre décroissant et doivent être triées par ordre croissant.

VARIABLE DÉPENDANTE:

La variable dépendante est le temps nécessaire pour trier les données car le temps varie en fonction de la façon dont les données doivent être triées.

VARIABLES CONTRÔLÉES:

Variable	Description	Spécifications
Ordinateur et système	le programme sera exécuté	Processeur: AMD Ryzen 7

d’exploitation	sur un Acer Aspire A515 - 45 avec Windows 11 Home.	5700U avec 1.80 GHz Mémoire: 16 GB
Environnement de développement intégré	Visual Studio 2022 sera utilisé	
L’algorithme utilisé	L'algorithme qui se trouve dans l'annexe A sera utilisé pour cette expérience.	
Le nombre de valeurs à trier	Pour tous les tableaux du meilleur cas, du cas moyen et du pire cas, il y aura 100, 500, 1000, 5000, 10000 et 50000 valeurs respectivement	
Le type de données	Pour cette expérience, seulement les données int seront utilisées	

MANIPULATION:

1. On a créé un nouveau projet en utilisant le modèle d'application de la console C++
2. Dans ce projet, on a écrit l'algorithme trouvé dans l'annexe #1.
3. On a exécuté le programme pour trier 100, 500, 1000, 5000, 10000 et 50000 valeurs en changeant la valeur de la variable `maxNum` (voir l'annexe #1)

4. On a exécuté le programme 10 fois pour les six différents montants de valeurs afin d'obtenir 10 résultats différents et réduire l'effet des erreurs, ce qui rend les résultats plus fiables.

RÉSULTATS:

TABLEAUX DE DONNÉES

Tableau #1: Moyenne des dix essais du temps nécessaire au tri (en ms) pour les trois algorithmes en fonction des différentes tailles de tableaux pour le scénario du meilleur cas

	Tri à bulles	Tri sélectif	Tri par insertion
100	0,0369	0,0114	0,0004
500	0,2901	0,2635	0,0015
1000	2,00517	1,84468	0,00622
5000	30,29	26,4571	0,0138
10000	113,9448889	102,8267	0,02704
50000	2900,57	2571,38	0,1349

Tableau #2: Moyenne des dix essais du temps nécessaire au tri (en ms) pour les trois algorithmes en fonction des différentes tailles de tableaux pour le scénario du cas moyen

	Tri à bulles	Tri sélectif	Tri par insertion
100	0,1443	0,0252	0,0008
500	1,6765	0,2606	0,0017
1000	11,56847	1,8941	0,00494
5000	172,036	26,6252	0,0204

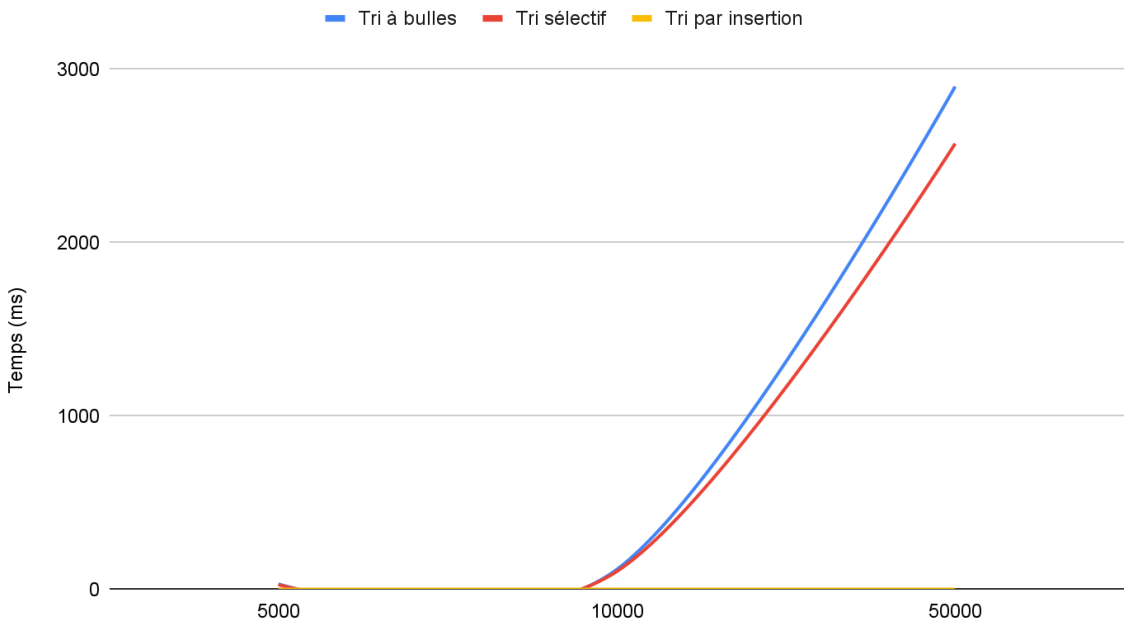
10000	677,9807	102,6989	0,02712
50000	17083,8	2567,72	0,1343

Tableau #3: Moyenne des dix essais du temps nécessaire au tri (en ms) pour les trois algorithmes en fonction des différentes tailles de tableaux pour le scénario du pire cas

	Tri à bulles	Tri sélectif	Tri par insertion
100	0,2254	0,0249	0,0009
500	2,5313	0,2677	0,0019
1000	17,60236	1,88261	0,00536
5000	271,203	27,6051	0,0186
10000	1010,391111	102,7395	0,02699
50000	25372,9	2571,15	0,1919

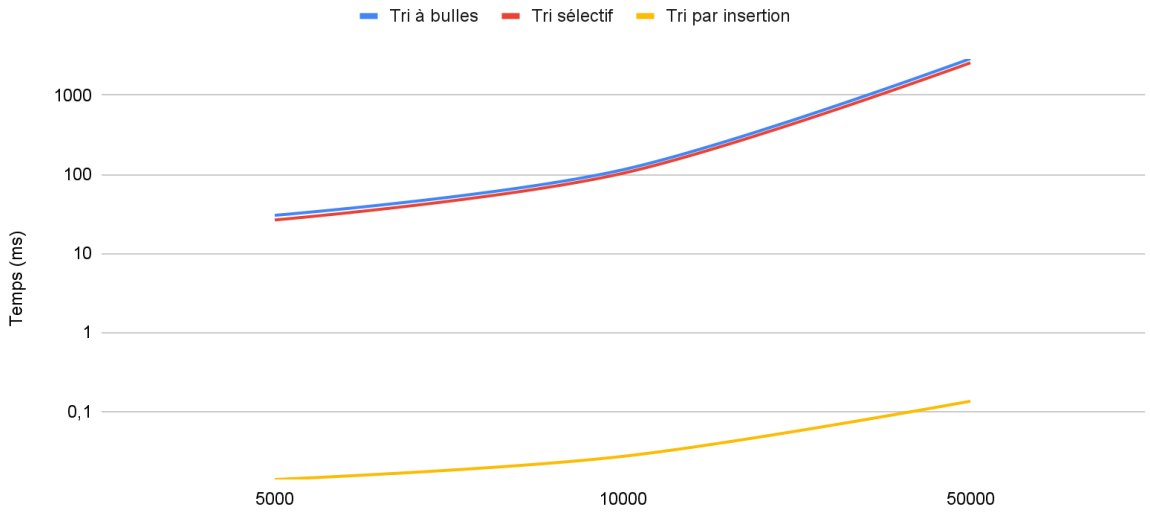
GRAPHIQUES:

Graphique #1: Moyenne des dix essais du temps nécessaire au tri pour les trois algorithmes en fonction des différentes tailles de tableaux pour le scénario du meilleur cas



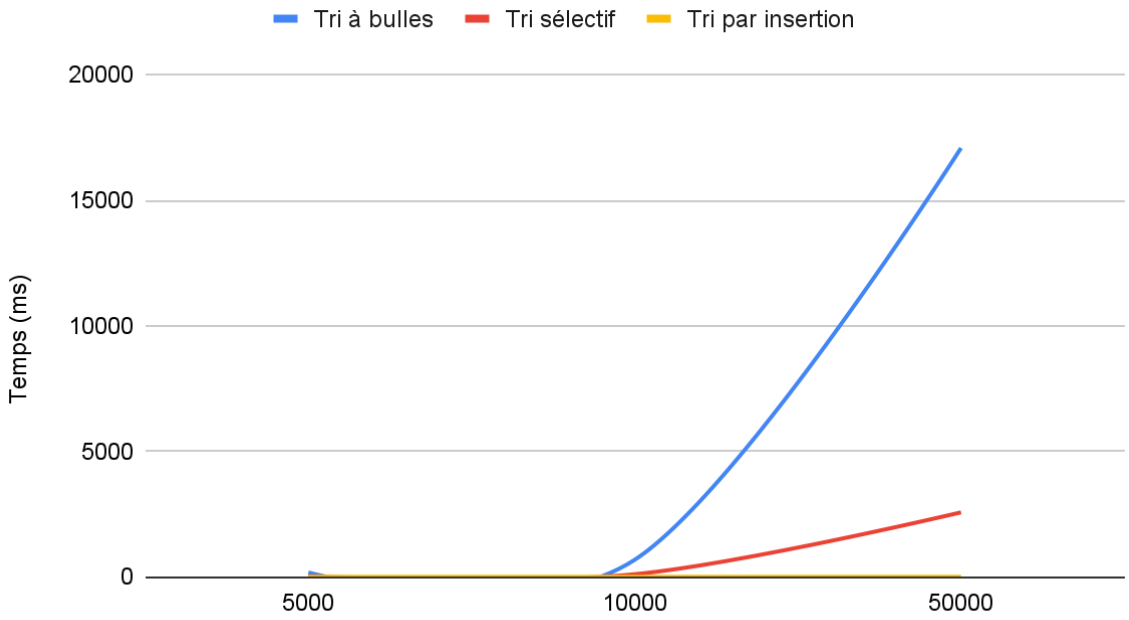
Dans ce graphique, on peut voir que pour le scénario du meilleur cas, le tri à bulles est le moins efficace, le tri par sélection est le deuxième moins efficace et le tri par insertion est le plus efficace parmi les trois algorithmes. Ceci est le cas pour les trois différentes tailles de tableaux. Le tri par insertion, qui est le plus efficace des trois algorithmes, se trouve très près de l'axe des x. Le prochain diagramme (graphique #2) le montre plus clairement. De plus, bien que le tableau soit déjà trié dans le meilleur des cas, les algorithmes prennent encore plus de temps pour trier les tableaux en fonction de leur taille. Plus la taille est grande, plus le temps de tri est long pour les trois algorithmes même si aucun échange n'est effectué entre les différentes valeurs des tableaux.

Graphique #2: Moyenne des dix essais du temps nécessaire au tri pour les trois algorithmes en fonction des différentes tailles de tableaux pour le scénario du meilleur cas



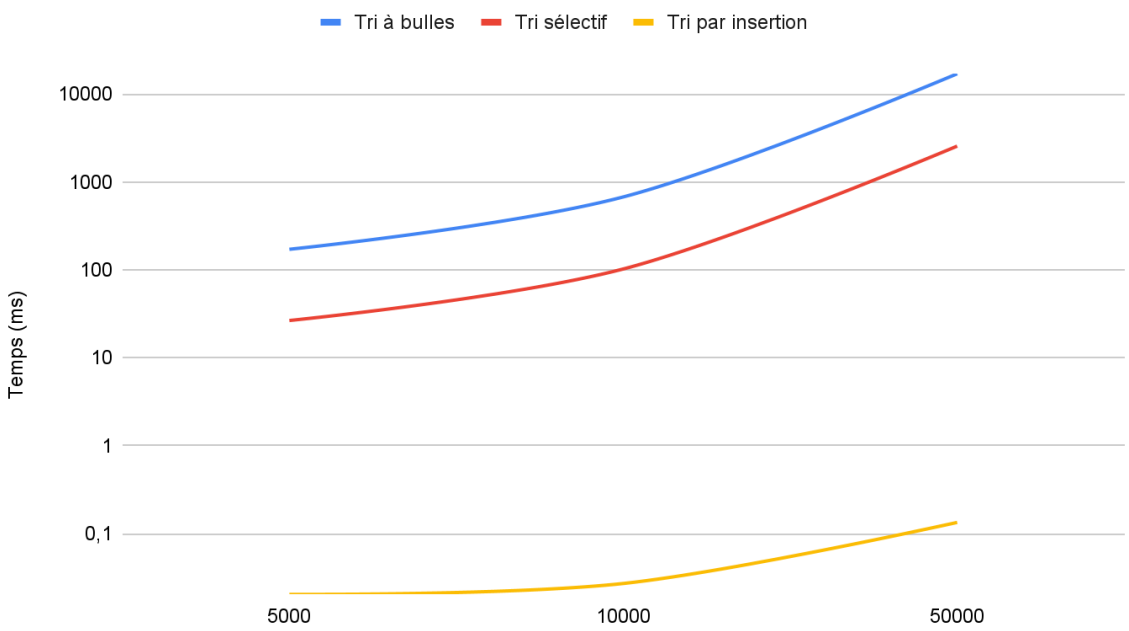
Ce graphique présente les mêmes données que le graphique #1 mais avec une échelle logarithmique pour faire ressortir les différences entre les trois types d'algorithmes. On constate que les pentes représentant les trois algorithmes ne se croisent pas, ce qui signifie que, quelle que soit la taille du tableau, le tri à bulles est toujours plus lent que le tri par sélection et le tri par sélection est toujours plus lent que le tri par insertion. De plus, la pente du tri par insertion est plus éloignée des pentes des algorithmes de tri à bulles et de tri par sélection, ce qui montre à quel point cet algorithme est plus rapide que les deux autres. Pour les algorithmes de tri à bulles et de tri sélectif, la différence en pourcentage est d'environ 12% pour les trois tailles de tableau. Mais si l'on compare la différence entre le tri par sélection et le tri par insertion, elle est beaucoup plus importante, avec une différence de 200 % pour les trois tailles.

Graphique #3: Moyenne des dix essais du temps nécessaire au tri pour les trois algorithmes en fonction des différentes tailles de tableaux pour le scénario du cas moyen



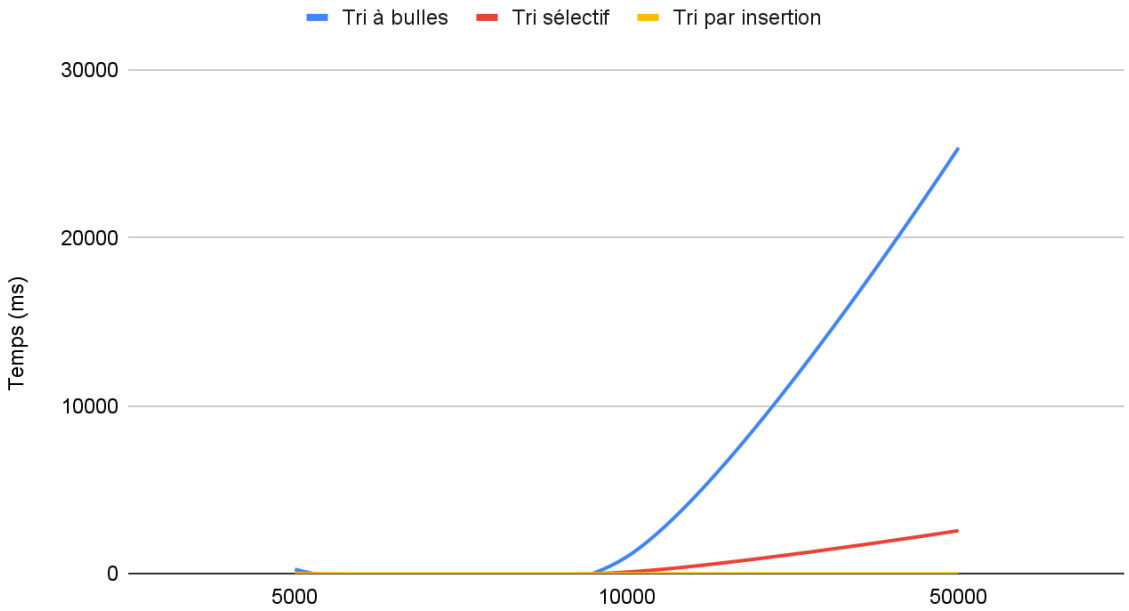
La tendance observée dans le premier graphique se répète dans ce graphique en affichant le même ordre d'efficacité pour les trois algorithmes. De plus, la taille du tableau a un effet sur le temps de tri des algorithmes. Par conséquent, le fait que le scénario soit différent pour ce graphique ne semble pas avoir influencé les tendances générales affichées par les trois algorithmes pour le meilleur scénario.

Graphique #4: Moyenne des dix essais du temps nécessaire au tri pour les trois algorithmes en fonction des différentes tailles de tableaux pour le scénario du cas moyen



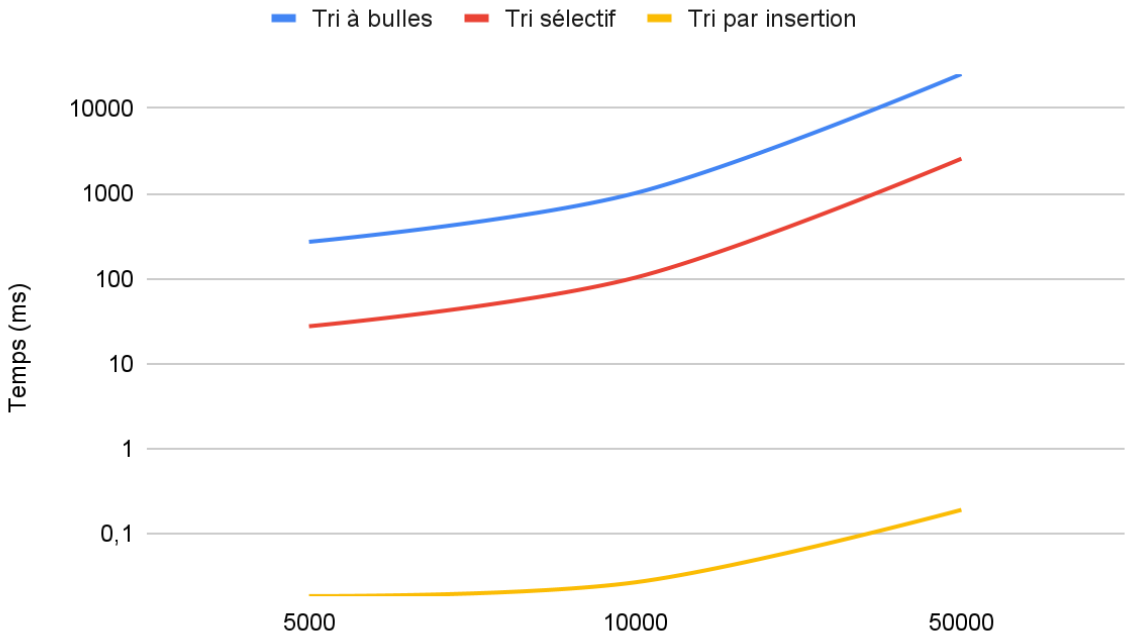
Le graphique ci-dessus présente les mêmes données que le graphique #3 avec une échelle logarithmique. Encore une fois, la tendance se répète puisque les trois pentes représentant les trois algorithmes ne se croisent pas. Une autre tendance qui se répète est la distance de la pente représentant l'algorithme de tri par insertion par rapport aux deux autres algorithmes. Cependant, la distance entre la pente de l'algorithme de tri à bulles et celle de l'algorithme de tri par sélection semble augmenter un peu par rapport au graphique #2. En fait, la différence entre les trois valeurs pour les pentes du tri à bulles et du tri sélectif pour ce graphique est d'environ 146%. Il s'agit d'une différence beaucoup plus importante que le 13 % du graphique #2. Mais, la différence de pourcentage entre les données pour le tri d'insertion et le tri sélectif est encore d'environ 200% ce qui est similaire pour les différences calculées pour le scénario du meilleur cas.

Graphique #5: Moyenne des dix essais du temps nécessaire au tri pour les trois algorithmes en fonction des différentes tailles de tableaux pour le scénario du pire cas



Les mêmes tendances sont observées avec le pire scénario. Les algorithmes ne semblent pas être significativement affectés par les différents scénarios, l'algorithme de tri par insertion demeure le plus efficace suivi de l'algorithme de tri par sélection et de tri à bulles.

Graphique #6: Moyenne des dix essais du temps nécessaire au tri pour les trois algorithmes en fonction des différentes tailles de



Pour le graphique représentant le scénario le plus défavorable avec une échelle logarithmique, la tendance des graphiques précédents se répète ici. En outre, la pente représentant l'algorithme de tri à bulles est, encore une fois, plus séparée de la pente du tri sélectif par rapport aux autres graphiques. Pour ce graphique, il y a une différence approximative de 163% entre les temps observés pour le tri à bulles et le tri par sélection. Cependant, la différence entre les temps pour les algorithmes de tri par sélection et de tri par insertion est toujours d'environ 200% comme observé dans les deux derniers cas. Par conséquent, nous pouvons déduire que le seul algorithme à être impacté par les différents cas est l'algorithme de tri à bulles, car il a tendance à s'écarter davantage des autres algorithmes lorsque le tableau est moins trié.

DISCUSSION:

D’après les données collectées lors de cette expérience, on a vu que pour chaque scénario, quelle que soit la quantité de valeurs dans le tableau, le tri à bulles était toujours le moins efficace des trois algorithmes, suivi par l'algorithme de tri par sélection et que l'algorithme de tri par insertion était le plus efficace. Il est également clair que l'algorithme de tri à bulles est l'algorithme le plus affecté par les différents scénarios, Ceci peut être à cause du fait qu’il y a plus d’échanges qui prennent place entre les valeurs dans le tableau alors cet algorithme va prendre plus de temps pour s’exécuter. Les deux autres algorithmes ne semblent pas être autant affectés par les trois cas différents comparés à l’algorithme de tri à bulles. Dans l'ensemble, ces résultats nous indiquent également comment les différents algorithmes devraient être utilisés à des fins différentes. Mais, l'algorithme d'insertion s'est avéré être le plus efficace et le plus rapide à trier les valeurs dans les trois différents scénarios peu importe laquantité de valeurs dans le tableau.

Cependant, cette recherche présente de nombreuses limites. Par exemple, le matériel énuméré dans les sections des variables contrôlées a été utilisé pour exécuter l'algorithme. Mais, une personne exécutant l'algorithme sur une autre machine avec des spécifications différentes

pourrait obtenir des résultats différents. Malgré ceci, les mêmes tendances devraient encore se montrer peu importe le matériel utilisé. Une autre limite de cette recherche s'applique spécifiquement à l'ensemble des données des scénarios moyens. Lorsque les valeurs ont été triées au hasard, elles ne contenaient pas toutes les nombres de 1 à la variable `maxNum`, mais plutôt différents nombres entre 1 et `maxNum`. Par conséquent, il est très probable qu'un nombre spécifique ait été trouvé deux fois dans l'ensemble de données et que certains nombres qui se trouvaient dans les ensembles de données du meilleur ou du pire cas ne se trouvaient pas dans l'ensemble de données du cas moyen parce qu'il y avait des doublons d'un autre nombre et qu'il n'y avait pas de place pour ce nombre spécifique. Cela peut avoir un effet sur le temps nécessaire pour trier l'ensemble de données du cas moyen, car parfois, le cas moyen peut avoir plus de doublons et moins de variété de nombres, ce qui rend le temps de tri plus rapide et vice versa. Par conséquent, si cet algorithme était exécuté une autre fois, les valeurs pour le cas moyen seraient différentes, ce qui pourrait affecter l'analyse globale des données lors de la comparaison des différents scénarios. Toutefois, cela ne s'applique qu'à l'algorithme de tri à bulles, car celui-ci semble être le seul à être affecté par les différents cas. De plus, le fait que l'algorithme ait été exécuté 10 fois et que le résultat pris en compte soit la moyenne des dix fois réduit l'effet de cette limitation puisque pour les différents essais où l'algorithme a été exécuté, il pourrait y avoir plus ou moins de doublons. Une troisième limite à cette expérience est le fait que de nombreux processus étaient en cours d'exécution dans la mémoire.Ceci signifie que l'allocation de mémoire aux tableaux et au tri pourrait avoir un impact sur l’exactitude du temps de tri, car il prend en compte le temps pour la mémoire d'exécuter le processus de tri et donc le résultat n'est pas seulement le temps d'exécution de l'algorithme, mais il est combiné avec l'allocation de processus dans la mémoire aussi.

En comparant les résultats à mon hypothèse, la seule incohérence claire est le fait que les scénarios n'ont pas eu d'effet sur les performances de l'algorithme de tri sélectif et de tri par insertion. En fait, les valeurs semblent être similaires pour les trois cas avec des différences

négligeables. Cependant, j'ai prédit l'ordre correct d'efficacité des trois algorithmes dans mon hypothèse.

L'une des principales façons dont cette recherche aurait pu être améliorée est que le tableau des cas moyens aurait pu être modifié de manière significative. Au lieu de contenir des nombres aléatoires, il devrait contenir tous les nombres compris entre 1 et le nombre maximum triés de manière aléatoire, de sorte qu'il n'y ait pas de doublons, que chaque nombre compris entre 1 et le nombre maximum soit présent dans le tableau et que le temps de tri soit plus fiable. Un autre changement qui peut être apporté à cette recherche est d'analyser l'utilisation de la mémoire pour chaque algorithme de tri. Dans cette expérience, seul le temps de tri a été pris en compte et non la mémoire utilisée. Une analyse de la mémoire permettrait d'avoir une vision plus globale des algorithmes et de savoir lesquels sont les meilleurs dans les divers cas en tenant compte de la mémoire et pas seulement du scénario et du temps. De plus, cette recherche aurait pu examiner d'autres algorithmes de tri différents qui ne sont pas nécessairement des tris par comparaison, comme le tri par seau ou le tri par radis, ce qui aurait également permis de mieux comprendre le concept des algorithmes de tri dans son ensemble. On pourrait aussi essayer de comparer les temps de tri en utilisant différents langages de programmation pour écrire les algorithmes. Le C++ (le langage utilisé pour écrire l'algorithme de cette expérience) est considéré comme rapide car c'est un langage compilé. Cependant, un langage interprété tel que Python est beaucoup plus lent et il serait intéressant de voir comment le langage de programmation affecte l'efficacité du tri. Une autre modification qui pourrait être apportée à cette expérience serait d'optimiser les trois algorithmes et de voir comment cela affecte le temps de tri. Les trois algorithmes de tri ont été pris dans leur forme standard, mais il existe de nombreuses façons de les optimiser et cela permettrait d'avoir une meilleure idée de la façon dont ces optimisations améliorent les algorithmes.

CONCLUSION:

En conclusion, le but de comparer les trois différents algorithmes en termes de scénarios de meilleur cas, de pire cas et de cas moyen a été atteint. Ce mémoire permet de mieux comprendre la manière dont les algorithmes de tri à bulles, de sélection et d'insertion fonctionnent pour des ensembles de données comportant différentes quantités de valeurs triées dans le meilleur, le pire et le moyen des scénarios. L'étape suivante consisterait à comparer de la même manière d'autres algorithmes plus efficaces, tels que les algorithmes de tri rapide ou de tri shell, afin de comprendre comment les algorithmes de tri fonctionnent dans les trois cas.

BIBLIOGRAPHIE:

1. “Sorting Algorithms Explained.” FreeCodeCamp.org, 18 Jan. 2020, https://www.freecodecamp.org/news/sorting-algorithms-explained/.
2. “Comparison Sort.” Wikipedia, Wikimedia Foundation, 28 Apr. 2022, https://en.wikipedia.org/wiki/Comparison_sort.
3. “Bubble Sort Algorithm.” GeeksforGeeks, 14 July 2022, https://www.geeksforgeeks.org/bubble-sort/.
4. “Bubble Sort.” Programiz, https://www.programiz.com/dsa/bubble-sort.
5. “Insertion Sort.” GeeksforGeeks, 13 July 2022, https://www.geeksforgeeks.org/insertion-sort/.
6. “Insertion Sort Algorithm.” Programiz, https://www.programiz.com/dsa/insertion-sort.
7. “Selection Sort Algorithm.” GeeksforGeeks, 1 Sept. 2022, https://www.geeksforgeeks.org/selection-sort/.
8. “Selection Sort Algorithm.” *Programiz,* https://www.programiz.com/dsa/selection-sort.
9. “Asymptotic Notation (Article) | Algorithms.” Khan Academy, Khan Academy, https://www.khanacademy.org/computing/computer-science/algorithms/asymptotic-notation/a/asymptotic-notation.
10. “<chrono>” *Cplusplus.com*, https://cplusplus.com/reference/chrono/.

ANNEXE #1:

```
#include <iostream>
#include <chrono>
#include <string>
#include <fstream>
using namespace std;
const int maxNum = 50000;
void bubbleSort(int *arr, int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
}
void swap(int* xp, int* yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int *arr, int n)
{
    int i, j, min_idx;


    for (i = 0; i < n - 1; i++)
    {
        min_idx = i;
        for (j = i + 1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;

        if (min_idx != i)
            swap(&arr[min_idx], &arr[i]);
    }
}

void insertionSort(int *arr, int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key)
        {
```

```
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

int main()
{
    ofstream File_;
    File_.open("numbers.txt", ios::app);

    int BestCase[maxNum];
    for (int i = 0; i < maxNum; i++) {
        BestCase[i] = i + 1;
    }
    int n = sizeof(BestCase) / sizeof(BestCase[0]);
    auto startBubbleBest = chrono::steady_clock::now();
    bubbleSort(BestCase, n);
    auto endBubbleBest = chrono::steady_clock::now();
    auto diffBubbleBest = endBubbleBest - startBubbleBest;
    File_ << "Tri a bulles, meilleur cas: " <<
chrono::duration<double, milli>(diffBubbleBest).count() << "
ms" << endl;

    auto startSelectionBest = chrono::steady_clock::now();
    selectionSort(BestCase, n);
    auto endSelectionBest = chrono::steady_clock::now();
    auto diffSelectionBest = endSelectionBest -
startSelectionBest;
    File_ << "Tri selectif, meilleur cas: " <<
chrono::duration<double, milli>(diffSelectionBest).count() <<
" ms" << endl;

    auto startInsertionBest = chrono::steady_clock::now();
    insertionSort(BestCase, n);
    auto endInsertionBest = chrono::steady_clock::now();
    auto diffInsertionBest = endInsertionBest -
startInsertionBest;
    File_ << "Tri par insertion, meilleur cas: " <<
chrono::duration<double, milli>(diffInsertionBest).count() <<
" ms" << endl;

    unsigned seed = time(0);
    srand(seed);
    ofstream File;
    File.open("data.txt");
    const int x = maxNum;
```

```
    for (int i = 0; i < x; i++) {
        int a = rand() % maxNum + 1;
        File << a << endl;
    }
    ifstream file;
    file.open("data.txt");
    int numbers;
    int arr[x] = {};
    int i = -1;

    if (file.is_open()) {
        while (file >> numbers) {
            i = i + 1;
            arr[i] = numbers;
        }
        int n = sizeof(arr) / sizeof(arr[0]);
        auto startBubbleRand = chrono::steady_clock::now();
        bubbleSort(arr, n);
        auto endBubbleRand = chrono::steady_clock::now();
        auto diffBubbleRand = endBubbleRand - startBubbleRand;
        File_ << "Tri a bulles, cas moyen: " <<
chrono::duration<double, milli>(diffBubbleRand).count() << "
ms" << endl;

        auto startSelectionRand = chrono::steady_clock::now();
        selectionSort(arr, n);
        auto endSelectionRand = chrono::steady_clock::now();
        auto diffSelectionRand = endSelectionRand -
startSelectionRand;
        File_ << "Tri selectif, cas moyen: " <<
chrono::duration<double, milli>(diffSelectionRand).count() <<
" ms" << endl;

        auto startInsertionRand = chrono::steady_clock::now();
        insertionSort(arr, n);
        auto endInsertionRand = chrono::steady_clock::now();
        auto diffInsertionRand = endInsertionRand -
startInsertionRand;
        File_ << "Tri par insertion, cas moyen: " <<
chrono::duration<double, milli>(diffInsertionRand).count() <<
" ms" << endl;
    }
    int WorstCase[maxNum];
    for (int i = maxNum; i > 0; i--) {
        WorstCase[maxNum - i] = i;
    }
    auto startBubbleWorst = chrono::steady_clock::now();
    bubbleSort(WorstCase, n);
```

```
    auto endBubbleWorst = chrono::steady_clock::now();
    auto diffBubbleWorst = endBubbleWorst - startBubbleWorst;
    File_ << "Tri a bulles, pire cas: " <<
chrono::duration<double, milli>(diffBubbleWorst).count() << "
ms" << endl;

    auto startSelectionWorst = chrono::steady_clock::now();
    selectionSort(WorstCase, n);
    auto endSelectionWorst = chrono::steady_clock::now();
    auto diffSelectionWorst = endSelectionWorst -
startSelectionWorst;
    File_ << "Tri selectif, pire cas: " <<
chrono::duration<double, milli>(diffSelectionWorst).count() <<
" ms" << endl;

    auto startInsertionWorst = chrono::steady_clock::now();
    insertionSort(WorstCase, n);
    auto endInsertionWorst = chrono::steady_clock::now();
    auto diffInsertionWorst = endInsertionWorst -
startInsertionWorst;
    File_ << "Tri par insertion, pire cas: " <<
chrono::duration<double, milli>(diffInsertionWorst).count() <<
" ms" << endl << endl;
    return 0;
}
```