Université 8 Mai 1945 Guelma
Faculté de mathématiques et de l'informatique et de sciences de la matière
Département Informatique

RAPPORT MINI PROJET

ANALYSE DES METHODES D'ALLOCATION DE FICHIERS ET ALGORITHMES DE TRI

REALISE PAR :

NOM	PRENOM	GROUPE
NOM1 : ABDELLI	PRENOM1 : MOHAMED EL AMINE	G 03
NOM2 : BOUNEFLA	PRENOM2 : MOHAMED	

ANNEE UNIVERSITAIRE 2023/2024

1. Structure de données	
Structure de données	Rôle
Structure d'étudiant	Champs : Matricule : Le numéro matricule de l'étudiant. Nom : Le nom de l'étudiant. Prénom : Le prénom de l'étudiant. Groupe : le Groupe de l'étudiant. Suivant : pour traitement de la liste chainée des étudiants
Structure de nœud	Champs : Étudiant : C'est une structure représentant les informations d'un étudiant. Gauche : Pointeur vers le sous-arbre gauche. Droit : Pointeur vers le sous-arbre droit.
2. Liste des sous-programmes	
Description (signature)	Rôle
chargerInfoArbre()	Description : Charge les données d'un fichier texte dans un arbre binaire de recherche. Paramètres : Aucun. Retour : Aucun.
ajouterEtudiantCroissant()	Description : Ajoute un étudiant à l'arbre de manière décroissante selon le matricule. Paramètres : La racine de l'arbre et un pointeur vers un étudiant. Retour : La racine de l'arbre mise à jour.
afficherArbreCroissant()	Description : Affiche les données de l'arbre de manière croissante. Paramètres : La racine de l'arbre. Retour : Aucun.
creerNoeud	Description : Crée un nouveau nœud pour l'arbre avec un étudiant donné. Paramètres : Un pointeur vers un étudiant. Retour : Un pointeur vers le nouveau nœud.
charger_info_tableau()	Description : Charge les données d'un fichier texte dans un tableau

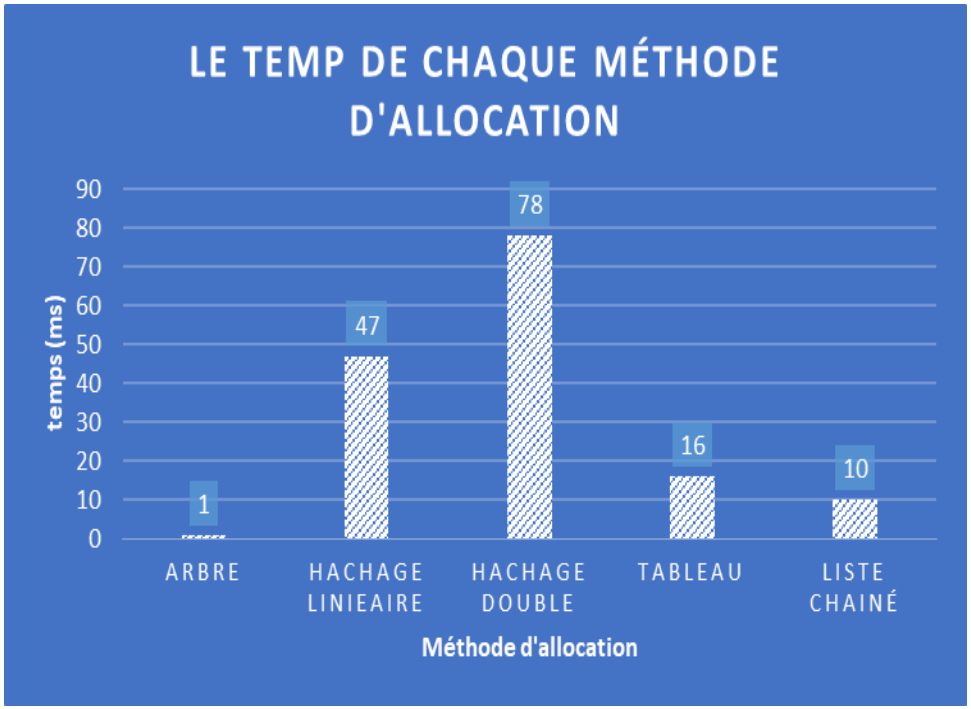
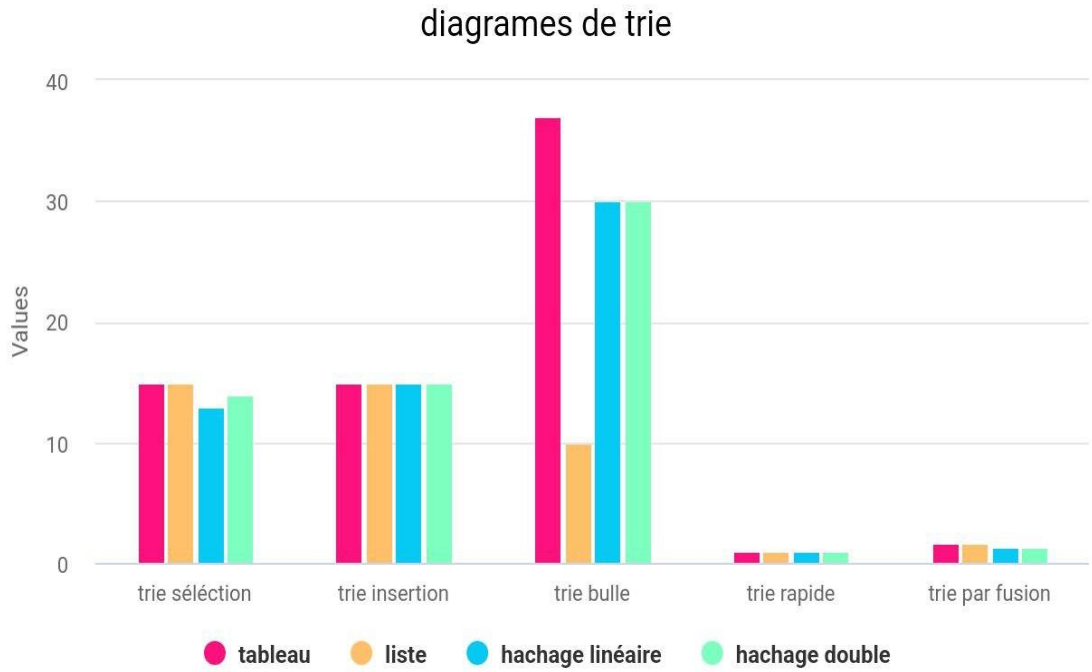
	dynamique. Paramètres : Aucun. Retour : Aucun.
trier_bulle	Description : Trie un tableau d'étudiants par la méthode de tri à bulles. Paramètres : Un tableau d'étudiants et sa taille. Retour : Le temps écoulé pour le tri.
trier_selection	Description : Trie un tableau d'étudiants par la méthode de tri par sélection. Paramètres : Un tableau d'étudiants et sa taille. Retour : Le temps écoulé pour le tri.
trier_insertion	Description : Trie un tableau d'étudiants par la méthode de tri par insertion. Paramètres : Un tableau d'étudiants et sa taille. Retour : Le temps écoulé pour le tri.
Fusionner	Description : Fusionne deux parties d'un tableau pendant le tri par fusion. Paramètres : Un tableau d'étudiants, les indices de début, de milieu et de fin. Retour : Aucun.
tri_fusion_etudiants	Description : Trie un tableau d'étudiants par la méthode de tri par fusion. Paramètres : Un tableau d'étudiants, les indices de début et de fin. Retour : Le temps écoulé pour le tri.
partitionner	Description : Partitionne un tableau pendant le tri rapide. Paramètres : Un tableau d'étudiants, les indices de début et de fin. Retour : L'index de pivot.
tri_rapide_etudiants	Description : Trie un tableau d'étudiants par la méthode de tri rapide. Paramètres : Un tableau d'étudiants, les indices de début et de fin. Retour : Le temps écoulé pour le tri.
afficher_resultats	Description : Affiche les résultats d'un tableau d'étudiants. Paramètres : Un tableau d'étudiants et sa taille. Retour : Aucun.

menu_de_tri_etudiants	Description : Affiche un menu pour choisir le type de tri à appliquer sur un tableau d'étudiants. Paramètres : Un tableau d'étudiants et sa taille. Retour : Le temps écoulé pour le tri.
fonction_hash_1	Description : Fonction de hachage pour le chargement dans une table de hachage. Paramètres : Un nombre (matricule). Retour : L'indice calculé.
fonction_hash_2	Description : Fonction de hachage pour le chargement dans une table de hachage. Paramètres : Un nombre (matricule). Retour : L'indice calculé.
charger_info_tableau_hachage	Description : chargez les informations d'un tableau de hachage en utilisant la première fonction de hachage et la linéarisation. Paramètres : Aucun. Retour : Aucun.
charger_info_tableau_hachage	Description : chargez les informations d'un tableau de hachage en utilisant la première et la deuxième fonction de hachage . Paramètres : Aucun. Retour : Aucun.
Charger_info_liste()	Description : Charge les données d'un fichier texte dans une liste chainée. Paramètres : Aucun. Retour : Aucun.
Trie_selection_liste	Description : Trie une liste d'étudiants par la méthode de tri par sélection. Paramètres : Aucun. Retour : Aucun.
Trie_bulle_liste	Description : Trie une liste d'étudiants par la méthode de tri à bulle. Paramètres : Aucun. Retour : Aucun
afficher_liste	Description : afficher la liste des étudiants après le trie

Trie_insertion_liste	Description : Trie une liste d'étudiants par la méthode de tri par insertion. Paramètres : Aucun. Retour : Aucun
quick_sort(liste)	Description : Trie une liste d'étudiants par la méthode de tri par rapide. Paramètres : début et la fin de la liste. Retour : Aucun
Trie_fussion_liste	Description : Trie une liste d'étudiants par la méthode de tri par fusion. Paramètres : Aucun. Retour : Aucun
swap_list	Description : Pour le swap des valeurs des nœuds lors de trie de la liste Paramètres : Aucun. Retour : Aucun
fusionner2	Description : pour le fusionnement des deux listes chainées en gardant l'ordre du trie pendant le trie par fusion Paramètres : deux listes. Retour : le résultat de fusionnement dans une seule liste en gardant l'ordre de trie
free_liste()	Description : Pour libérer l'espace mémoire après le trie de la liste chainé Paramètres : Aucun. Retour : Aucun
list milieu	Description : calculez le milieu de tableau lors de la fusion pendant le trie par fusion Paramètres : la tête de liste. Retour : le pointeur vers l'élément en milieu
list partition(list debut, list fin)	Description : trouvez le pivot lors de trie rapide Paramètres : le début et la fin de la liste dans chaque itération. Retour : le pivot
3. Détails du programmes	

Nombre de lignes	1044
Nombre de variables globales	l : etudiant*, un pointeur vers le début d'une liste chaînée d'étudiants. Nœud arbre : la racine de l'arbre Nbr_etudiants : le nombre des étudiants total
Description de la méthode de hachage	**Le Hachage linéaire** : Initialisation de la somme : La variable somme est initialisée à zéro. Elle servira à accumuler les résultats intermédiaires. Boucle while : La fonction utilise une boucle while pour traiter chaque chiffre du nombre en entrée (nombre). La boucle continue tant que nombre est supérieur à zéro. Extraction du chiffre : À chaque itération de la boucle, le chiffre le moins significatif de nombre est extrait en utilisant l'opération modulo (nombre % 10). Cela donne le reste de la division de nombre par 10, donc le chiffre le moins significatif. Multiplication par 2 : Le chiffre extrait est ensuite multiplié par 2. Addition à la somme : Le résultat de la multiplication par 2 est ajouté à la variable somme. Mise à jour de nombre : Enfin, nombre est mis à jour en divisant par 10, ce qui équivaut à supprimer le chiffre le moins significatif. Retour de la somme modulo 100 : Une fois que tous les chiffres du nombre ont été traités, la fonction retourne la valeur de somme modulo 100. Cela signifie que la fonction renvoie le reste de la division de somme par 100, limitant ainsi le résultat final à une valeur entre 0 et 99 Si cette fonction retour une case qui est déjà occupé on continue par linéarisation la recherche d'une case vide dans le tableau si on fait le toure de toutes les cases sans l'insertion de cet élément en question cela signifie que le tableau est déjà plein Pour **le Hachage double** : on appelle la fonction précédente si on trouve une case pour insérez l'étudiants donc c'est le meilleur cas sinon la deuxième fonction de hash est plus sophistiquée que la premier, voici

	ces instructions : Initialisation de la somme : La variable somme est initialisée à zéro. Elle sera utilisée pour accumuler les résultats intermédiaires des opérations. Boucle while : Une boucle while est utilisée pour traiter chaque chiffre du numéro de matricule (matricule). La boucle continue tant que matricule est supérieur à zéro. Extraction du chiffre : À chaque itération de la boucle, le chiffre le moins significatif de matricule est extrait en utilisant l'opération modulo (matricule % 10). Cela donne le reste de la division de matricule par 10, donc le chiffre le moins significatif. Multiplication par 3 : Le chiffre extrait est ensuite multiplié par 3. Addition à la somme : Le résultat de la multiplication par 3 est ajouté à la variable somme. Mise à jour de matricule : Enfin, matricule est mis à jour en divisant par 10, ce qui équivaut à supprimer le chiffre le moins significatif. Ajout de 80 à la somme : Après avoir traité tous les chiffres, la valeur 80 est ajoutée à la variable somme. Division par 13(division par un nombre premier sert à élargir la différence entre les résultats retourné par la fonction) : Ensuite, la variable somme est divisée par 13. Conversion en entier : La valeur de somme est ensuite convertie en entier en utilisant le casting (int). Retour de la somme modulo 100 : Enfin, la fonction retourne le reste de la division de la valeur obtenue par 100. Cela limite le résultat final à une valeur entre 0 et 99. Noté que cette fonction est appelée x fois pour trouver une case dans le tableau de hachage

4. Résultats	
Diagramme de comparaison des méthodes d'allocations avec et sans hachage	Diagramme de comparaison des méthodes de tri par type d'allocation
	
Conclusion	**Conclusion**
L'allocation par hachage requiert davantage de temps que d'autres méthodes d'allocation, du fait que les deux approches exigent un temps accru lors de la gestion des collisions. De plus, le hachage double se révèle moins rapide en raison de la nature moins souple de sa gestion des collisions, comparativement à des méthodes telles que la linéarisation. En effet, la fonction de hachage double est appelée plusieurs fois pour trouver un emplacement vide, selon les circonstances. **Cependant, les méthodes d'allocation par d'autres structures telles que les tableaux, les listes et les arbres sont plus rapides que**	**Les méthodes de tri se montrent plus efficaces lorsque l'utilisateur opte pour le tri du tableau après l'application du hachage, étant donné que la table de hachage contient un nombre de cases non triées inférieur à celui du tableau et de la liste chaînée. Cela se traduit par un temps de tri post-hachage légèrement plus rapide. Cependant, il est crucial de souligner que, lorsqu'il s'agit de données relatives à un milliard d'étudiants, le temps nécessaire devient considérablement plus significatif. À titre d'exemple, en choisissant le tri à bulle dans le tableau avec hachage, cette méthode prend 37 ms, alors qu'avec le hachage, le tri par la fonction de tri à bulle nécessite seulement 30 ms. Ainsi, la**

l'allocation par hachage. Même les opérations d'allocation dynamique surpassent en rapidité la gestion des collisions. Finalement, la méthode de chargement la plus performante s'avère être l'arbre binaire de recherche, car la localisation de l'emplacement se fait rapidement, avec une complexité de log(n) qui est plus rapide que l'ordre linéaire (O(n)) observé dans le cas des tableaux et des listes chaînées.	**différence s'élève à 7 ms. Pour un milliard d'étudiants, cela se traduit par un total de 7 milliards de millisecondes, une valeur de grande ampleur.** **En d'autres termes, l'utilisation de l'arbre binaire de recherche pourrait être envisagée si l'on constate une augmentation des collisions en fonction des données, car cette méthode demeure tout aussi efficace. La distinction entre l'utilisation de l'arbre binaire de recherche et du hachage dépend de la nature des données traitées.** **En ce qui concerne les algorithmes de tri, il émerge que le tri rapide se distingue, dans notre contexte, par ses performances supérieures par rapport à d'autres, notamment en raison de son efficacité démontrée dans le traitement de grands ensembles de données. Toutefois, il convient de noter que le tri rapide présente une certaine instabilité en comparaison avec le tri par fusion, qui, lui, maintient la stabilité des résultats.** **Par ailleurs, les algorithmes reposant sur le paradigme des comparaisons manifestent une certaine lenteur. Ces approches, bien que relativement simples à mettre en œuvre, se révèlent peu efficaces, notamment dans le cas de vastes ensembles de données. Leur facilité d'implémentation est contrebalancée par une inefficacité marquée lorsqu'il s'agit de traiter des données d'ampleur considérable.** **En fin de compte, le choix de l'algorithme de tri dépend des exigences spécifiques du problème à résoudre. Il est essentiel de considérer la taille des données, la stabilité, la complexité temporelle et spatiale, ainsi que d'autres facteurs pertinents pour prendre la décision la plus adaptée à la situation particulière.**

NOTE : Les informations ont été altérées en intégrant 1900 étudiants afin de permettre au processus de tri de calculer le temps. Puisque le calcul du temps pour un grand nombre d'étudiants fourni dans le document donne toujours 0 en raison de l'architecture avancée des microprocesseurs de nos jours