Séance 7: algorithmes de tri

Les objectifs de cette séance, et du travail individuel à fournir d'ici la séance suivante, sont :

21-Implémenter différents algorithmes de tri en C

22-Caractériser la complexité de différents algorithmes de tri

Je vous suggère, dans un premier temps, de réaliser les exercices 1 et 2, puis d'en choisir un parmi les 3,4 et 5 et un autre parmi les 6, 7 et 8. Veillez à vous répartir ces exercices pour qu'on puisse faire des comparaisons entre vous.

1.Tri à gobelets

Inventé par Benoit Vleminckx, le tri par gobelet consiste, tant que le tableau n'est pas trié, à en permuter deux éléments sélectionnés au hasard.

-Implémente cet algorithme de tri sous la forme d'une fonction qui prend en arguments un tableau et sa taille et qui renvoie le nombre de permutations qui ont été effectuées.

-Détermine si cet algorithme se termine dans tous les cas.

-Détermine l'évolution du nombre de permutations nécessaires pour trier ce tableau par rapport à la taille du tableau fourni en entrée.

-Que conclure sur l'efficacité de cet algorithme ?

2.Complexité d'un algorithme

L'étude de la complexité d'un algorithme consiste à mesurer le temps (sous la forme d'un nombre d'opérations) que met un algorithme pour se terminer, en fonction de la taille du problème qu'il doit traiter. Quand on étudie la complexité, on cherche généralement à définir le comportement asymptotique de l'algorithme sur le pire des cas possible. Par exemple, si je fais une recherche simple dans un tableau de taille "n", élément par élément jusqu'à trouver celui que je cherche, dans le pire des cas je parcourrai les "n" éléments pour trouver celui que je cherche. On dit que la complexité est de O(n). Ça signifie que si la taille du tableau est de 10, il me faudra au pire 10 opérations pour trouver ce que je cherche. Si la taille est de 50 il me faudra au pire 50 opérations pour le trouver.

Pour un algorithme de complexité $O(n^2)$, par exemple, si la taille du problème est de 10, il me faudra au pire 100 opérations pour le résoudre. Si la taille du problème est de 50, il me faudra au pire 2500 opérations. L'augmentation n'est plus linéaire, mais quadratique !

Trouve des algorithmes simples qui correspondent aux classes de complexités usuelles suivantes : $O(1)$, $O(\log(n))$, $O(n)$, $O(n \log(n))$, $O(n^2)$, $O(2^n)$

3.Tri par bulles

Le tri par bulle consiste à parcourir deux à deux les éléments d'un tableau et à inverser ces éléments s'ils ne sont pas dans le bon ordre, jusqu'à arriver au dernier élément. Puis à recommencer, jusqu'à l'avant-dernier élément et ainsi de suite jusqu'à ce que le tableau soit trié.

-Implémente cet algorithme de tri sous la forme d'une fonction qui prend en arguments un tableau et sa taille et qui renvoie le nombre de comparaisons qui ont été effectuées

-Détermine si cet algorithme se termine dans tous les cas

-Détermine la complexité algorithmique de cet algorithme

-Évalue expérimentalement sa complexité moyenne (en nombre de comparaisons effectuées)

4. Tri par sélection

Ce tri consiste à trouver le plus petit éléments du tableau et à le mettre au début, puis à trouver le deuxième plus petit et à le mettre en deuxième position etc, jusqu'à la fin.

-Implémente cet algorithme de tri sous la forme d'une fonction qui prend en arguments un tableau et sa taille et qui renvoie le nombre de comparaisons qui ont été effectuées

-Détermine si cet algorithme se termine dans tous les cas

-Détermine la complexité algorithmique de cet algorithme

-Évalue expérimentalement sa complexité moyenne (en nombre de comparaisons effectuées)

5. Tri par insertion

Ce tri consiste à trier les deux premiers éléments, puis à insérer le 3^{e} pour que les trois premier soient triés, puis à faire pareil pour les suivants jusqu'à la fin.

-Implémente cet algorithme de tri sous la forme d'une fonction qui prend en arguments un tableau et sa taille et qui renvoie le nombre de comparaisons qui ont été effectuées

-Détermine la complexité algorithmique de cet algorithme

-Évalue expérimentalement sa complexité moyenne (en nombre de comparaisons effectuées)

6.Quick sort

Ce tri récursif consiste, si le tableau a une taille de 1, à renvoyer le tableau. Sinon, choisir un élément "p" qui sera le pivot. Placer tous les éléments du tableau plus petit que "p" à gauche de "p" et tous les éléments plus grands à droite. Puis réappliquer l'algorithme sur le sous-tableau à gauche de "p" et puis sur le sous-tableau à droite de "p".

-Implémente cet algorithme de tri sous la forme d'une fonction qui prend en arguments un tableau, l'indice du premier élément du sous-tableau à trier, et l'indice du dernier élément du sous-tableau à trier, et qui renvoie le nombre de comparaisons qui ont été effectuées

-Détermine la complexité algorithmique de cet algorithme

-Évalue expérimentalement sa complexité moyenne (en nombre de comparaisons effectuées)

7.Tri fusion

Ce tri récursif se base sur le fait que fusionner deux tableaux déjà triés est très peu complexe. Si le tableau a une taille de 1, il le renvoie. S'il a une taille plus grande que 1, il le coupe en deux, les trie séparément, puis fusionne les deux sous-tableaux triés.

-Implémente cet algorithme de tri sous la forme d'une fonction qui prend en arguments un tableau, l'indice du premier élément du sous-tableau à trier, et l'indice du dernier élément du sous-tableau à trier, et qui renvoie le nombre de comparaisons qui ont été effectuées

-Détermine la complexité algorithmique de cet algorithme

-Évalue expérimentalement sa complexité moyenne (en nombre de comparaisons effectuées)

8.Tri par tas

Il s'agit d'un tri qui emploie une file de priorité implémentée au moyen d'un tas binaire (binary heap). Il consiste simplement à créer une nouvelle file de priorité de puis à y ajouter un à un les éléments du tableau à trier. Par définition, les éléments seront ainsi accessible de façon ordonnée depuis la file de priorité et peuvent être remis dans le tableau dans l'ordre.

-Implémente cet algorithme de tri sous la forme d'une fonction qui prend en arguments un tableau et sa taille et qui renvoie le nombre de comparaisons qui ont été effectuées

-Détermine la complexité algorithmique de cet algorithme

-Évalue expérimentalement sa complexité moyenne (en nombre de comparaisons effectuées)