UE Programmation Avancée - TP2

Implémentation des algorithmes de tri en C :

Fonction Tri Sélection : Permet de trier un tableau du plus petit au plus grand de n entier. Le tri par selection consisiste à rechercher le plus grand élément (ou le plus petit), le placer en fin de tableau (ou en début), recommencer avec le second plus grand (ou le second plus petit), le placer en avant-dernière position (ou en seconde position) et ainsi de suite jusqu'à avoir parcouru la totalité du tableau.

* tab est un tableau d'entiers
* n est un entier
* returne rien (a modifier en amont le tableau “tab” en triant ses éléments)

Fonction Tri Fusion : le tableau est découpé récursivement en une série de listes, de taille 1. Ces listes d'un élément sont ainsi triées. On fusionne ensuite progressivement les listes contiguës pour finalement avoir un tableau complètement trié. Le découpage récursif se fait grâce à la fonction mergeSort et la fusion des listes triées grâce à la fonction fusion qui les tris également.

* tab est un tableau d'entiers
* n est un entier
* returne rien (a modifier en amont le tableau “tab” en triant ses éléments)

Fonction Tri à Bulles : cherche des suites de deux éléments consécutifs dont le premier est supérieur au second. Tant que l'on trouve de telles séquences, on échange les deux éléments.

* tab est un tableau d'entiers
* n est un entier
* returne rien (a modifier en amont le tableau “tab” en triant ses éléments)

Fonction Tri Rapide: triRapide permet de trier un tableau qui sépare le tableaux en deux partitions tels que tous les éléments de la partition gauche(debut) sont plus petits qu’une valeur pivot et ceux de la partition droite(fin) sont plus grands. REMARQUE: Dans mon cas, j’ai également laissé un trie utilisant un pivot random en commentaire. En effet, j’avais un probleme de profondeur de récursivité limité à un tableau de taille 1000. Cette limite pouvait être franchie grâce à une bibliothèque : #include <sys/types.h>. Cependant il y avait toujours une limite de 20 000 pour mon tableau. J’ai tenté de faire un pivot random et même si le code fonctionné, cette limite ne pouvait être dépassé. C’est pour cela que mon programme en Python ne dépasse pas 20 000 éléments pour la taille des tableaux de test. Après un test sur l’ordinateur d’un collègue avec ce même code, j’ai constaté que celui-ci fonctionner pour une taille supérieur à 20 000. j’en conclue donc que le probleme était dû à la version de python (je possède la version 3.10.4 et lui la 3.9).

*tab est un tableau d'entiers
* deb est un entier sous forme d'indice de départ
* fin est un entier sous forme d'indice de fin
* returne rien (a modifier en amont le tableau “tab” en triant ses éléments)

Fonction de Test: Permet de vérifier si des tableaux avec peu d'éléments sont triés correctement après l'appel d'une fonction correspondant au Test voulu. Ainsi, en amont, nous avons deux tableaux de 7 valeurs non trié, et deux autres avec les mêmes valeurs mais trié.

*tab1 est un tableau d'entiers
*tab2 est un tableau d'entiers
*tab3 est un tableau d'entiers
*tab4 est un tableau d'entiers
*n est un entier
* returne "le tableau est trié" ou "le tableau n'est pas trié"

Fonction Echanger: Permet d'echange la position de deux éléments d'un tableau
*tab est un tableau d'entiers
*n est un entier
retourne rien

Fonction affiche: Affiche les éléments d'un tableau passer en paramètre de taille n.

*tab est un tableau d'entiers.
*n est un entier
*retourne le tab d'entier en paramètre

Fonction genTab: genTab génère n entiers aléatoirement et les insères dans le tableau

* tab est un tableau d'entiers.
* n est un entier inférieur limité à 10 avec le modulo choisit ici.
* retourne rien

Fonction copy: Permet de copier les éléments du premier tableau passer en paramètre dans le deuxieme pour une taille n.

*tab et tabBackup sont des tableaux d'entiers.
*n est un entier
*retourne rien

Comparaison des algorithmes (benchmarking) :

Mesure des temps d'exécution (Fonction Perf): Permet d'obtenir le temps d'excécution de la fonction entrée. Dans notre cas, on réalise une boucle afin d'obtenir 3 temps d'excécution et d'en faire une moyenne afin d'avoir des résultats plus robustes.

* tab est un tableau d'entiers.
*n est un entier

*retourne le temps moyen d'excécution du tri voulu

Construction du plan expérimental global et Enregistrement dans un fichier au format CSV :

Fonction BenchmarkGlobal: Dans mon cas, j'ai trouvé plus simple de réunir en une seul fonction le benchmark et le planGlobal. En effet, d'une part, cette fonction retourne les perfommances en temps des 4 fonctions de tris pour un tableau de même taille. Pour cela, on utilise un tableau de backup pour copier un tableau et le restituer à un autre tri dans un état "non trié". Grace à un pas donné, on parcours un tableau avec un pas afin d'avoir le temps d'excécution de chaque taille jusqu'à la taille n. Enfin, d'une autre part, on met nos données directement dans un fichier .csv créer en amont.

* pas est un entier
*n est un entier

Implémentation des algorithmes de tri en Python et comparaison avec leurs implémentations en C :

En ce qui concerne la comparaison des tris Python et des tris en C:

triBulles (en C) : 0,3
triBulles (en python): 29s

triSelection (en C):0,22s
triSelection (en python) :8s

triRapide (en C):0,12s
triRapide(en python): 2,5s

triFusion(en C): ~0,001s
triFusion(en python):~0,001s

On constate que le tri à bulles est environ 97 fois plus rapide qu'en python, que le tri selection est environ 36 fois plus rapide qu'en python et que le tri rapide est environ 21 fois plus rapide qu'en python.

Les fonctions sont toujours dans le même ordre d'efficacité selon le C ou le python. on peut s'apercevoir que le tri bulle est le moins efficace en terme de temps d'exécution.Viennent ensuite le tri par sélection puis le tri rapide. L'efficacité du tri fusion est sans commune mesure avec les trois tris « lents » précédents. De plus, que ce soit en python ou en C, ses mesures restes casiment les mêmes.

Conclusion:

Ainsi, on observe des résultats logique correspondant au calcul de complexité théorique. Le triFusion et le triRapide ont une complexitée en $\Theta(n \lg n)$.

Le triSelection a une complexité moyenne en n-ln n, Ce tri est donc intéressant lorsque les éléments sont aisément comparables, mais coûteux à déplacer dans la structure, d'où sa 3ème position, et sa différence de temps avec les deux tris cités précédement. Le tri à bulles à en moyenne un complexité de $\Theta(n2)$. Cette complexité démontre bien dans les faits qu'il est le plus long des algorithmes de tri étudier et de loin.

Enfin, Le C est compilé en binaire qui contient des instructions processeur à exécuter par le processeur, jusqu'à la fin du programme. Ce code binaire est chargé en mémoire dans l'ordinateur puis est exécuté aussi rapidement que le processeur et la RAM le permettent (lorsque le programme trouve le droit d'accéder au processeur).
Le code python est interpréter par un process qui va devoir prendre du temps convertir en action et exécuter les actions pour chaque ligne. Ça prend donc du temps en plus et cela explique les énormes écarts de temps entre le python et le C.