UE Programmation Avancée - TP2

Frédéric Flouvat
Université de la Nouvelle-Calédonie

Tri à bulles VS Tri Sélection VS Tri fusion VS Tri rapide

— Nombre de séances : 3
— TP à déposer sur Moodle : code sources + exécutable + rapport décrivant (succinctement) chaque méthode développée (entrées/sorties et captures d'écran des tests réalisés) et les résultats des expérimentations effectuées.

Objectif

L'objectif de ce TP est de comparer expérimentalement le temps d'exécution de plusieurs algorithmes de tri en fonction de la taille des données et du langage de programmation, et de faire le parallèle avec la complexité théorique. Les données à trier seront des ensembles d'entiers générés aléatoirement.

1 Implémentation des algorithmes de tri en C

Dans cette partie du TP, vous devrez implémenter les 4 algorithmes de tri suivants en C.

1.1 Algorithme Tri Sélection

Ecrivez l'algorithme de tri sélection vu en cours (procédure *triSelection(int * tab, int n)*).

1.2 Algorithme Tri Fusion

Ecrivez l'algorithme de tri fusion (version améliorée) vu en cours (procédure *triFusion(int * tab, int n)*).

1.3 Algorithme Tri à Bulles

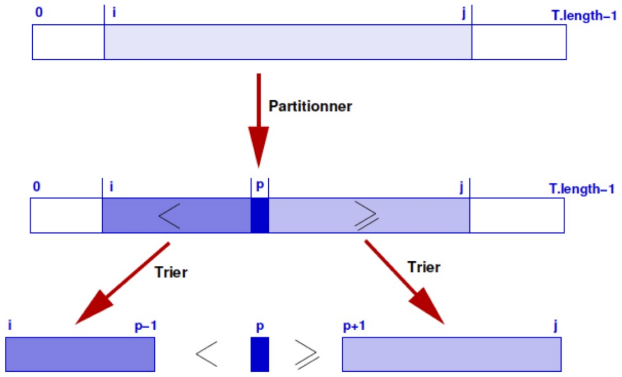
Ecrivez l'algorithme de tri à bulles décrit p. 35 du livre "Introduction à l'algorithmique" (procédure *triBulles(int[] tab, int n)*).

1.4 Algorithme Tri Rapide

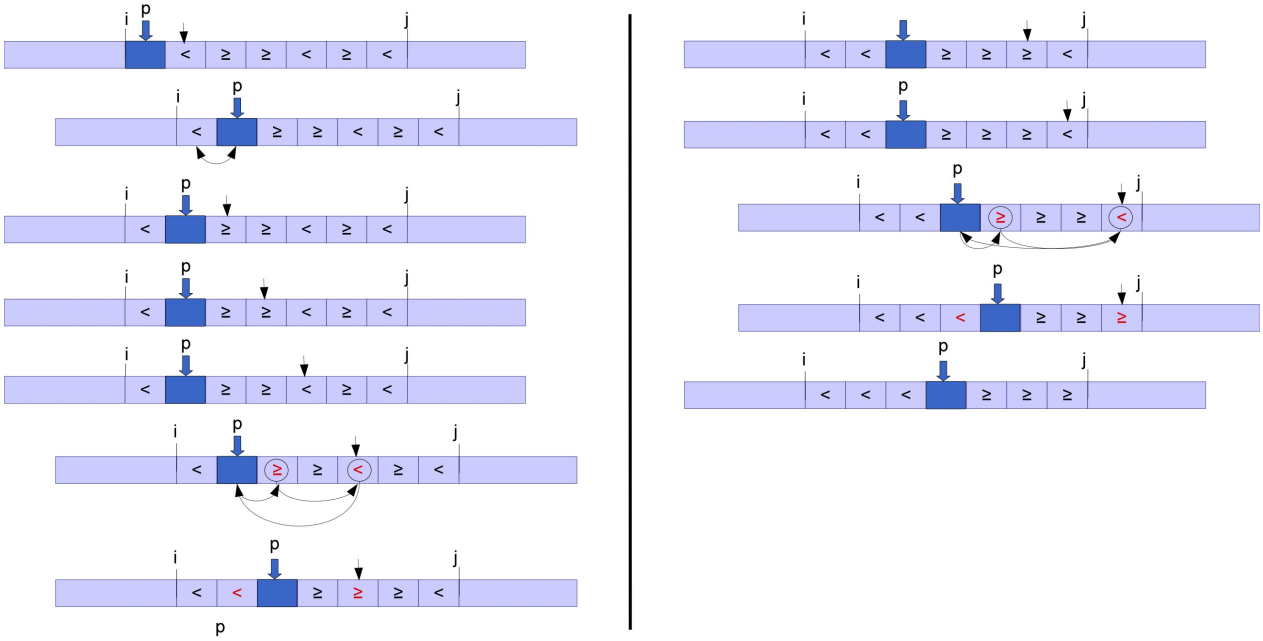
Principe (cf figure) :
— prendre le premier élément (le pivot), le mettre à sa place en plaçant à sa gauche les éléments plus petits que lui et à sa droite les éléments plus grands que lui.
— répéter récursivement l'opération d'une part sur le sous-tableau de gauche, et d'autre part sur le sous-tableau de droite (en ignorant l'élément pivot car celui-ci est déjà à sa place).

Pour avoir des informations complémentaires, veuillez vous reporter au chapitre 7 du livre "Introduction à l'algorithmique" (p.139).

Ecrivez la fonction *partitionner(int * tab, int deb,int fin)* qui sépare le tableaux *tab* (entre les cases i et j) en deux partitions tels que tous les éléments de la partition gauche sont plus petits



qu'une valeur pivot et ceux de la partition droite sont plus grands. Cette méthode commence par prendre pour valeur de référence (cad pivot) $tab[deb]$. Puis, pour chaque élément, $tab[i]$, entre les indices deb et fin, elle compare la valeur pivot à $tab[i]$. Si la valeur pivot est supérieur à l'élément courant $tab[i]$, alors il faut permuter $tab[i]$ et la valeur pivot (car les valeurs plus petites doivent être à gauche du pivot). Sinon, la méthode ne fait rien et continue son parcours du tableau (car l'élément testé est plus grand que le pivot et à droite de celui-ci). A la fin du parcours, la méthode retourne la nouvelle position du pivot. La figure suivante présente un exemple d'exécution de la fonction partitionner.



Ecrivez la procédure récursive *triRapide(int * tab, int deb, int fin)* implémentant l'algorithme du tri rapide entre les cases i et j du tableau.

2 Comparaison des algorithmes (*benchmarking*)

2.1 Génération aléatoire des entrées

Afin de pouvoir comparer les performances des algorithmes sur des entrées différentes, vous devez écrire une procédure permettant de générer aléatoirement les valeurs stockées dans un tableau

d'entier dont la taille est passé en paramètre. Par exemple, le prototype de cette procédure pourra être de la forme *genTab(int * tab, int n)* avec en paramètre un tableau de taille n (déjà alloué).

2.2 Mesure des temps d'exécution

Pour chaque algorithme de tri implémenté, vous devez créer un fonction de "benchmark". Pour le tri sélection, cette fonction pourra par exemple avoir le prototype suivant : *double triSelectionPerf(int * tab, int n)*. En entrée, cette fonction prendra en paramètre un tableau *tab* de taille n. En sortie, elle donnera le temps en secondes qu'aura mis l'algorithme pour trier ce tableau. Afin d'avoir des mesures plus robustes, la mesure retournée correspondra à la moyenne de trois exécutions de l'algorithme pour les mêmes entrées (taille et contenu).

Pour mesurer le temps d'exécution (le temps CPU) d'une méthode C, il faut utiliser la fonction *clock*() de la librairie `time.h` (importer aussi la librairie `sys/types.h`) de la manière suivante :

```
   ...
clock_t start, end;
start = clock();
maMethodeATester();
end = clock();
printf(" temps d'exécution de la méthode: %f",(double)(end-start)/CLOCKS_PER_SEC )
;
...
```

2.3 Construction du plan expérimental global

Ecrivez ensuite un programme principal pour comparer les différents algorithmes avec des entrées de taille de plus en plus grandes. Pour chaque taille, vous devrez générer un tableau aléatoirement. Puis, vous devrez en passer une **copie** (différente, mais avec les mêmes valeurs) en paramètre des différents algorithmes de tris implémentés.

2.4 Enregistrement dans un fichier au format CSV

Vous enregistrerez vos résultats dans un fichier texte au format CSV. L'intérêt de ce fichier est de pouvoir facilement être importé dans des outils tels que Excel de Microsoft ou Calc de LibreOffice. Vous organiserez ces fichiers de façon à pouvoir facilement générer des graphiques montrant l'évolution du temps d'exécution des différents algorithmes en fonction de la taille de l'entrée.

2.5 Visualisation des résultats et analyse

Vous commenterez les graphiques obtenus et discuterez les résultats. Il s'agira ici notamment de faire le parallèle avec les complexités théoriques de ces algorithmes.

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

Vous implémenterez ensuite ces algorithmes en Python et referaient les mêmes expérimentations. L'objectif sera de comparer les performances des algorithmes codés en Python avec ceux codés en C.