TD Maple n°8 : Algorithmes de tri*

Lycée Louis le Grand

PCSI 1

Lundis 30 mars et 6 avril 2009

Le but de ce TD est d'étudier quelques méthodes classiques pour trier des listes d'entiers dans l'ordre croissant.

1 Quelques rappels sur les listes

Définition d'une liste : `[> L :=[1,4,9,16 ] ;`
Liste vide : `[> L :=[ ] ;`
Longueur d'une liste : `[> nops(L) ;`
i-ème élément d'une liste : `[> L[i] ;`
Extraction d'une sous-liste : `[> L[i..j] ;`
Ajout d'un élément à la fin d'une liste : `[> L :=[op(L),25] ;`
Ajout d'un élément à la i-ème place d'une liste : `[> L :=[op(L[1..i-1]),25,op(L[i..nops(L)])] ;`

ATTENTION : dans tout ce TD, vous n'aurez le droit d'utiliser que les instructions ci-dessus, les boucles `for` et `while`, les conditions `if` et les relations binaires de comparaisons `< > <= >= = <>`.

2 Tri par sélection

Le tri par sélection est un des algorithmes de tri les plus triviaux. Il consiste en la recherche du plus grand élément (resp. le plus petit) que l'on va replacer à sa position finale c'est-à-dire en dernière position (resp. en première), puis on recherche le second plus grand élément (resp. le second plus petit) que l'on va replacer également à sa position finale c'est-à-dire en avant-dernière position (resp. en seconde), etc..., jusqu'à ce que la liste soit entièrement triée. Exemple :

$$1, 10, 3, 5, 2, 1, 6$$

$$1, 6, 3, 5, 2, 1, 10$$

$$1, 1, 3, 5, 2, 6, 10$$

$$1, 1, 3, 2, 5, 6, 10$$

$$1, 1, 2, 3, 5, 6, 10$$

*Ce TD est inspiré d'un TD donné par S. Gonnord au lycée du Parc à Lyon.

1. Ecrire une procédure `posmax(L)` qui prend en argument une liste et qui renvoie la position du plus grand élément de la liste. Si cet élément est présent plusieurs fois dans la liste, on renvoie le plus petit indice de position. Exemple :
   `[> posmax([1,25,3,-5]) ;` renvoie 2 et `[> posmax([1,6,24,-5,24]) ;` renvoie 3.
2. Ecrire une procédure `triselect(L)` qui prend en argument une liste et qui renvoie la même liste triée dans l'ordre croissant en utilisant l'algorithme de tri par sélection.
3. Tester votre procédure sur des listes de tailles différentes.
4. Montrer que pour une liste de taille n, le nombre de comparaisons $C(n)$ nécessaires pour ordonner la liste est en $\mathcal{O}(n^2)$, c'est-à-dire que $\exists M > 0, \forall n,\ C(n) \leq Mn^2$.

3 Tri par insertion

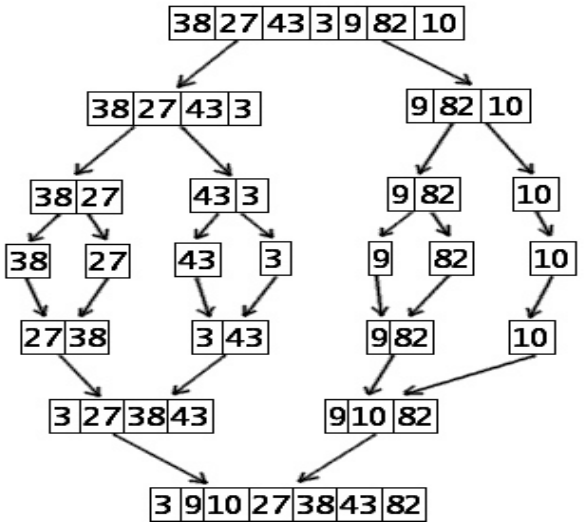
Le principe de ce tri est très simple : c'est le tri que toute personne utilise naturellement quand elle a des dossiers à classer. On prend un dossier et on le met à sa place parmi les dossiers déjà triés. Puis on recommence avec le dossier suivant.

Pour metter en oeuvre ce tri, on prend un indice i qui va parcourir toute la liste. Le principe étant qu'après avoir inséré `L[i]`, la sous-liste `L[1..i]` sera triée. Pour insérer `L[i]`, on regarde si `L[i-1]` $>$ `L[i]`. Si c'est le cas, on échange `L[i]` et `L[i-1]`. Puis on continue avec le couple $(i-2, i-1)$... On s'arrête si à un moment `L[i-1]` $\leq$ `L[i]` ou si l'on a atteint l'extémité gauche de la liste.

1. Simuler à la main l'insertion de `L[i]` dans `L[1..i-1]` lorsque `L :=[2,3,6,10,12,4,9,5,16]` et $i = 6$.
2. Pour i fixé, l'insertion de `L[i]` est de la forme "tant que $k > 1$ et `L[k-1]>L[k]`, faire ..." Alors on fait quoi au juste ? Et k prend quelle valeur avant la boucle "tant que" ?
3. Ecrire une procédure `triinser(L)` qui prend en argument une liste et qui renvoie la même liste triée dans l'ordre croissant en utilisant l'algorithme de tri par insertion.
4. Tester votre procédure.
5. Quel est le nombre nécessaire de comparaisons du type `L[k-1]>L[k]` pour ordonner une liste de taille n dans le pire des cas (liste triée dans l'ordre décroissant) ?

4 Tri par fusion

L'algorithme de tri par fusion est conçu selon la stratégie "diviser pour mieux régner". On divise la liste à trier en deux listes de taille à peu près égales, on trie ces deux listes en applicant la même méthode puis un fusionne les deux listes déjà triées. Exemple :



1. Dans un premier temps, nous allons écrire une procédure `fusion(L1,L2)` qui prend en arguments deux listes de tailles respectives $n1$ et $n2$ déjà triées dans l'ordre croissant et qui renvoie une liste contenant les éléments des deux listes et triée dans l'ordre croissant. On va insérer les éléments au fur et à mesure dans une liste L initialement vide. On prend deux indices $i1$ et $i2$ partant de 1 et parcourant les deux listes L1 et L2. Tant que ($i1 \leq n1$ et $i2 \leq n2$) on regarde qui est le plus petit entre `L1[i1]` et `L2[i2]` et on le rajoute à la fin de L, on incrémente alors $i1$ ou $i2$ et on recommence. A la fin de la boucle "tant que", une des deux listes a été entièrement insérée, il reste à insérer le reste de l'autre liste à la fin de L.
2. Dans une procédure `trifus(L)`, implémenter l'algorithme récursif (cf l'annexe plus bas) de tri par fusion donné ci dessous.

```
trifus(L) :
Variables locales L1,L2
Si longueur de L > 1 alors
    L1 := première moitié de L
    L2 := deuxième moitié de L
    L := fusion(trifus(L1),trifus(L2))
fin si
Renvoyer L
fin
```

3. Tester votre procédure.

   On admet que, dans le pire des cas, il faut $n-1$ comparaisons du type `L1[i1]<L2[i2]` ou `L1[i1]>L2[i2]` pour fusionner deux listes ordonnées dont la somme des longueurs est n .
4. Soit $C(n)$ le nombre de comparaisons nécessaires pour trier une liste de taille n avec l'algorithme de tri fusion. Donner une relation simple entre $C(2n)$ et $C(n)$.
5. En déduire la valeur de $C(2^k)$.
6. Maintenant, à n fixé, on peut choisir k tel que $2^k < n \leq 2^{k+1}$. On a clairement $C(n) \leq C(2^{k+1})$. Prouver alors que $C(n) = \mathcal{O}(n\ln n)$. On dit que l'alorithme est asymptotiquement optimal car on ne peut pas faire mieux que $\mathcal{O}(n\ln n)$ en terme de complexité.