TD Maple n°8 : Algorithmes de tri
Corrigé

Lundis 30 mars et 6 avril 2009

1 Tri par sélection

1. Pour trouver la position du maximum d'une liste, on initialise d'abord le maximum à $-\infty$. Ensuite on parcourt la liste de gauche à droite, si on rencontre un élément strictement supérieur au max, on réactualise max et pos_max et on continue.

```
posmax :=proc(L)
local max,pos_max,i,n;
n :=nops(L);
max :=-infinity;
for i from 1 to n do
    if L[i] > max then
        max :=L[i];
        pos_max :=i;
    fi;
od;
pos_max;
end;
```

2. On fait varier i de 0 à $n-1$ où n est la taille de la liste. Pour i fixé, on cherche le maximum dans liste `L[1..n-i]` puis on l'échange avec le dernier terme de cette liste.

```
triselect :=proc(L)
local tmp,j,i,n,M;
M :=L;
n :=nops(M);
for i from 0 to n-1 do
    j :=posmax(M[1..n-i]);
    tmp :=M[n-i];
    M[n-i] :=M[j];
    M[j] :=tmp;
od;
M;
end;
```

Il est nécessaire de travailler sur une liste M et non pas sur la liste L donnée en argument, car lorsqu'on programme avec Maple, on ne peut pas modifier les éléments donnés en argument à une procédure. Noter aussi que lorsqu'on veut échanger deux termes d'une liste `M[n-i]` et `M[j]`, il faut stocker `M[n-i]` dans une variable temporaire `tmp` car elle est écrasée par l'opération M[n-i] :=M[j].

3. Faire les tests vous même.
4. Les comparaisons se font toutes à l'intérieur de la procédure `posmax`. Pour une liste de taille p, cette pocédure parcourt la liste et effectue p comparaisons du type `L[i] > max`. Ainsi, à chaque étape i de la boucle de `triselect`, on fait $n-i$ comparaisons. Donc, `triselect` effectue en tout :

$$C(n) = \sum_{i=0}^{n-1} n - i = \frac{n(n+1)}{2} = \mathcal{O}(n^2)$$

2 Tri par insertion

1. Au départ $k = i = 6$ puis on fait les échanges successifs :
   `[2,3,6,10,12,4,9,5,16]`
   `[2,3,6,10,4,12,9,5,16]`
   `[2,3,6,4,10,12,9,5,16]`
   `[2,3,4,6,10,12,9,5,16]`

   Puis on recommence avec le 7-ème élément qui est 9.
2. On commence à `k :=i` puis, "tant que $k > 1$ et `L[k-1]>L[k]`, on échange `L[k-1]` et `L[k]` puis on fait `k :=k-1`".
3. Voici la procédure `triinser(L)` :

```
triinser :=proc(L)
local tmp,i,k,n,M;
M :=L;
n :=nops(M);
for i from 2 to n do
    k :=i;
    while (k>1 and L[k-1]>L[k]) do
        tmp :=M[k];
        M[k] :=M[k-1];
        M[k-1] :=tmp;
     od;
od;
M;
end;
```

   Si on utilise des listes trop longues (plus de 100 éléments), il faut convertir L en tableau avec `M :=array(L)`, puis on fait `eval(M);` pour renvoyer le résultat et non plus simplement `M;`.
4. Faire vous même les tests.
5. Si la liste est ordonnée dans l'ordre décroissant, alors à chaque étape i, la boucle `while` continue jusqu'à atteindre la gauche du tableau i.e. jusqu'à ce que $k = 2$. On fait alors $i-1$ comparaisons du type `L[k-1]>L[k]`. D'où le nombre total de comparaisons :

$$C(n) = \sum_{i=2}^{n} i - 1 = \frac{1}{2}(n+1)^2 - \frac{3}{2}n - \frac{1}{2} = \mathcal{O}(n^2)$$

   Remarque : les comparaisons du type `k>1` ne sont pas prises en compte car en général, lorsqu'on implémente cet algorithme, on fixe `M[0] :=-infinity`, ce qui permet de ne pas échanger `M[1]`

et `M[0]` (qui n'existe pas réellement) et donc d'arrêter la boucle `while`. On dit qu'on a placé une *sentinelle* en 0. Mais cela n'est pas possible avec Maple donc on est obligé de rajouter le test `k>1`.

3 Tri par fusion

1. Voici la procédure `fusion(L1,L2)` :

```
fusion :=proc(L1,L2)
local i1,i2,n1,n2,L;
n1 :=nops(L1);n2 :=nops(L2);
i1 :=1;i2 :=1;
L :=[ ];
while(i1<=n1 and i2<=n2) do
    if L1[i1]<L2[i2] then L :=[op(L),L1[i1]]; i1 :=i1+1;
    else L :=[op(L),L2[i2]]; i2 :=i2+1;
    fi;
 od;
if i1>n1 then L :=[op(L),op(L2[i2..n2])];
elif i2>n2 then L :=[op(L),op(L1[i1..n1])];
fi;
L;
end;
```

2. Voici la procéure `trifus(L)` :

```
trifus :=proc(L)
local n,M,L1,L2;
M :=L;
n :=nops(M);
if n >1 then
    L1 :=M[1..floor(n/2)];
    L2 :=M[floor(n/2)+1..n];
    M :=fusion(trifus(L1),trifus(L2));
fi;
M;
end;
```

3. Faire les tests vous-même.
4. Pour trier une liste de taille $2n$, on regarde si $2n > 1$ (1 comparaison), puis on tri deux listes de tailles n (donc $2C(n)$ comparaisons) puis on fusionne ces deux listes ($2n-1$ comparaisons). Finalement, $C(2n) = 1 + 2C(n) + 2n - 1 = 2C(n) + 2n$.
5. On en déduit que $C(2^k) = 2^k C(1) + \alpha_k$ où la suite (α_k) vérifie :
$$\begin{cases} \alpha_1 & = 1 \\ \alpha_{k+1} & = 2\alpha_k + 2^{k+1} \end{cases}$$
En divisant par 2^{k+1}, on montre facilement que $\forall k, \alpha_k = k2^k$. D'où en voyant que $C(1) = 0$, $C(2^k) = k2^k$.
6. On a $2^k < n \leq 2^{k+1}$ donc $k\ln 2 < \ln n \leq (k+1)\ln 2$. Comme tous les termes sont positifs, on a alors $2^k k\ln 2 < n\ln n \leq 2^{k+1}(k+1)\ln 2$. Pour montrer que $C(n) = \mathcal{O}(n\ln n)$, il suffit de trouver $M > 0$ indépendant de k tel que $Mk2^k \ln 2 \geq (k+1)2^{k+1}$. $M \geq \dfrac{4}{\ln 2}$ convient.