Corrigé du TD du 9 mars 2006

1 Tri par insertion

```
insertion:=proc(L)
   local M,i,j,n,m;
      n:=nops(L);
      M:=[];
      for i to n do
         M:=[op(M),L[i]];
         j:=i;
         while j>1 and M[j]<M[j-1] do
            m:=M[j]; M[j]:=M[j-1]; M[j-1]:=m;
            j:=j-1;
         end do;
      end do;
   M;
end proc;
```

Au départ `M` est une liste vide. A chaque passage dans la boucle `for`, on ajoute le i^{me} élément de `L` à droite dans `M` (on pourrait aussi le faire à gauche...) et on rentre dans la boucle `while`. Cette boucle permute le nouvel élément ajouté avec celui qui le précède tant que `M` n'est pas classée: c'est là qu'est faite l'insertion. `m` est une variable locale qui permet de faire ces permutations.
Il y plusieurs façons d'aborder la complexité du tri-insertion: on peut compter le nombre de comparaisons et le nombre d'interversions. On va regarder le nombre de comparaisons. Dans le meilleur cas on en fait n (liste triée) et dans le pire, environ $\sum_{k=1}^{n} k \simeq \frac{n^2}{2}$ (liste dans l'ordre décroissant). On fait en moyenne $\frac{n^2}{4}$ opérations. L'algo est donc de complexité quadratique.

2 Tri fusion

Il est commode de diviser l'algorithme de tri fusion en deux sous-programmes: un tri et une fusion. On commence par la procédure de fusion de deux listes classées. L'idée est de parcourir les deux listes de gauche à droite en comparant les éléments courants des deux listes et en mettant le plus petit dans la liste de fusion.

```
fusion:=proc(L1,L2)
   local L,n1,n2,m1,m2;
      L:=[]; n1:=nops(L1); n2:=nops(L2); m1:=1; m2:=1;
      while m1<=n1 and m2<=n2 do
```

```
         if L1[m1]<=L2[m2] then
            L:=[op(L),L1[m1]];
            m1:=m1+1;
         else
            L:=[op(L),L2[m2]];
            m2:=m2+1;
         end if;
      end do;
      if m1>n1 then
         L:=[op(L),op(L2[m2..n2])];
      else
         L:=[op(L),op(L1[m1..n1])];
      end if;
end proc;
```

`L` est la liste de fusion, `m1` et `m2` sont les positions des "curseurs" dans les listes `L1` et `L2`. Tant qu'une des deux listes n'a pas été entièrement parcourue, on compare les deux éléments courants et le plus petit rentre à droite dans `L`. Dès qu'une des listes est épuisée, on place les éléments restants de l'autre à droite dans `L`. On obtient une liste fusionnée et classée!

Maintenant, la procédure de tri. Elle est basée sur le principe "diviser pour régner". On divise récursivement la liste en deux sous-listes qu'on va trier et fusionner.

```
tri:=proc(L)
   local n;
      n:=nops(L);
      if n=1 then L;
      else fusion(tri(L[1..floor((n+1)/2)]),tri(L[floor((n+1)/2)+1..n]));
      end if;
end proc;
```

Quelle est la complexité du tri fusion? Regardons ce qui se passe pour une liste de longueur 2^k. On fait 2^{k-1} fusions de listes contenant 1 élément, qui demandent une comparaison chacune, 2^{k-2} fusions de listes à deux éléments, qui demandent 2 à 3 comparaisons chacune, ..., 2^{k-i} fusions de listes de 2^{i-1} éléments, qui demandent entre 2^{i-1} et $2^i - 1$ comparaisons chacune, etc... On fait donc à peu près entre $(k-1)2^{k-1}$ et $(k-1)2^k$ comparaisons. Ramené à une liste de longueur n, on a donc une complexité en $O(n\log(n))$. Un théorème stipule qu'une méthode de tri procédant par comparaisons ne peut pas être plus rapide que du $O(n\log(n))$, donc le tri fusion est optimal. Cependant, le tri rapide ou *quicksort* est plus utilisé dans les ordinateurs, car même si il est un petit peu moins rapide, il demande moins d'allocations de mémoire.