X 2010 : Info pour non-informaticiens

```
> restart;
> m := 14; alloue := proc(m) Array(1..m) end;
> P8 := alloue(8);
> taille := ArrayNumElems;
> taille(P8);
```

Question 1

version listes

```
> PolTestL := [$1..8];

> evalueListe := proc (P, u) if nops(P) = 1 then u*P[1] else
  u*(P[1]+evalueListe(P[2 .. -1], u)) end if end proc;
> evalueListe(PolTestL, 10);
```

version arrays

```
> met := proc(k) global P8; P8[k] := k end: map(met, [$1..8]) : P8;
> evalueArray := proc(P, u) local tP, k, valH;
        tP := taille(P);  valH := P[tP];
        for k from (tP - 1) by -1 to 1 do valH := valH*u + P[k] end do;
        u·valH
    end:
> evalueArray(P8, 10);
```

Question 2

version listes

```
> listeNulle := proc(l) if nops(l) = 1 then if l[1] = 0 then true else false fi; else ((l[1]
      = 0) and listeNulle(l[2..-1])) fi end;
> valuationNonNulle := proc(l)  if l[1] ≠ 0
                    then 1
               else 1 + valuationNonNulle(l[2..-1]) fi end;
> valuationL := proc(l)
          if listeNulle(l) then 0 else valuationNonNulle(l) fi end;
> valuationL([0, 1, 2, 3]); valuationL([5, 0, 1, 2, 3]); valuationL([0, 0, 0, 0]);
```

version arrays

```
> P8 := alloue(8); P8[4] := 1; X4 := P8; P8 := alloue(8); P8[8] := -21; X8 := P8; P8
      := alloue(8); X0 := P8;
> valuationA := proc(P) local k, maxNul;
      maxNul := 0;
```

```
        for k from 1 by 1 to taille(P) while P[k] = 0 do maxNul := k end do;
        if maxNul = taille(P) then 0 else (1 + maxNul) fi ; end:
>
> valuationA(X4); valuationA(X8); valuationA(X0);
>
```

Question 3

```
>
```

avec des listes

```
> P1l := [seq(rand(4)() - 2, i = 1..7)] : P2l := [seq(rand(11)() - 3, i = 1..4)] : P3l
      := [seq(rand(11)() - 3, i = 1..5)] : P4l := [seq(rand(11)() - 3, i = 1..8)] :
     P5l := [seq(rand(11)() - 3, i = 1..3)] : P6l := [seq(rand(11)() - 3, i = 1
     ..9)] : P7l := [seq(rand(11)() - 3, i = 1..6)] :
>
> differenceL :=proc(P1, P2) local n1, n2, mini, maxi, new, k;
        n1 := nops(P1); n2 := nops(P2); mini := min(n1, n2); maxi := max(n1, n2);
        new := [$1..maxi];
            for k from 1 to mini do new[k] := P1[k] - P2[k] od;
            if n1 > n2 then for k from mini + 1 to maxi do new[k] := P1[k] od fi;
           if n2 > n1 then for k from mini + 1 to maxi do new[k] := -P2[k] od fi;
        new; end:
>
> differenceL(P1l, P2l); differenceL(P2l, P1l);
```

avec des arrays

```
> P1a := alloue(8); P2a := alloue(6);
  P1a[1 + rand(8)()] := rand(4)() - 2; P2a[1 + rand(6)()] := rand(6)() - 3;;
>
>
> differenceA :=proc(A1, A2) local n1, n2, mini, maxi, new, k;
        n1 := taille(A1); n2 := taille(A2); mini := min(n1, n2); maxi := max(n1,
    n2);
        new := alloue(maxi);
            for k from 1 to mini do new[k] := A1[k] - A2[k] od;
            if n1 > n2 then for k from mini + 1 to maxi do new[k] := A1[k] od fi;
           if n2 > n1 then for k from mini + 1 to maxi do new[k] := -A2[k] od fi;
        new; end:
>
> differenceA(P1a, P2a); differenceA(P2a, P1a);
```

Question 4

avec des listes

```
> CompareNegL := proc(P1, P2) local d, v ;
        d := differenceL(P1, P2); v := valuationL(d);
            if v = 0
            then 0
           else if type(v, even)
                then signum(d[v])
                else (-signum(d[v]))
                end if
```

```
                    end if end:
> CompareNegL(P1l, P2l); CompareNegL(P2l, P1l); CompareNegL(P2l, P2l);
>
```

avec des arrays

```
> CompareNegA := proc(P1, P2) local d, v;
                 d := differenceA(P1, P2); v := valuationA(d);
                  if v = 0
                  then 0
                 else if type(v, even)
                       then signum(d[v])
                       else (-signum(d[v]))
                       end if
                 end if end:
> CompareNegA(P1a, P2a); CompareNegA(P2a, P1a); CompareNegA(P2a, P2a);
>
```

Question 5

> C'est "purement" la question de cours ... hors programme. Quand on a fait de l'informatique on sait que 'tous' les tris simples sont de complexité O(n²/2), et que le plus simple des tris par comparaison en O(n ln(n)) est le tri fusion. Ce tri-fusion est clair à programmer en pur fonctionnel, il est très vilain en pur itératif. Le tri-par-tas est bien sûr encore mieux mais pour des non-informaticiens ???
>
> Ce problème voulait-il un tri-bulle? par sélection? par insertion?
>
> Il pourrait (?) y avoir un intérêt si on avait un type de données particulier

> Après avoir fini ce problème, j'ai en plus constaté que les fonctions ultérieures n'utilisaient pas ce tri (sauf bien sûr la Question 6 qui en conclut l'usage). Je décrête donc gaiement que je ne vais PAS programmer un tri fusion pous les arrays

tri fusion avec des listes

```
> fuse:=proc(ordre,a,b)
      if a=[]
      then b
      elif b=[]
      then a
      elif ordre(a[1],b[1])
      then [a[1],op(fuse(ordre,a[2..-1],b))]
      else [b[1],op(fuse(ordre,a,b[2..-1]))]
      fi
  end:
> ioe :=proc(x, y) x ≤ y end; soe :=proc(x, y) x ≥ y end;
```

```
> fuse(ioe,[1,4,7,9,10],[2,5,8]);fuse(soe,[1000,410,72,9,1],
  [92,51,8]);
> prepare:=proc(a)
     if a=[]
     then []
     else [[a[1]],op(prepare(a[2..-1]))]
     fi end;
> prepare([1,4,7,8,5,2,17]);
> FusePaires:=proc(ordre,a)
     if nops(a)<=1
     then [a[1]]
     elif nops(a)=2
     then [fuse(ordre,a[1],a[2])]
     else [fuse(ordre,a[1],a[2]),op(FusePaires(ordre,a[3..-1]
  ))]
     fi
  end;
> FusePaires(ioe,prepare([1,4,7,8,5,2,14]));
> FusePaires(ioe,%);
> TriFusion:=proc(ordre,a)
      local aa;
      aa:=prepare(a);
      while nops(aa)>1
      do aa:=FusePaires(ordre,aa) od;
      aa[1]
  end;
> TriFusion(ioe,[24,1,4,7,8,5,2,30]);TriFusion(soe,[24,1,4,7,
  8,5,2,30]);
```

un exemple d'usage (du tri fusion de listes)

```
> listePol := [P1l, P2l, P3l, P4l, P5l, P6l, P7l];
> BoolCompareNegL :=proc(P, Q) CompareNegL(P, Q) ≥ 0 end;
> TriFusion(BoolCompareNegL, listePol);
```

Question 6

avec des listes

```
> ComparePosL := proc(P1, P2) local d, v;
                 d := differenceL(P1, P2); v := valuationL(d);
                      if v = 0
                      then 0
                      else signum(d[v])
                      end if end;
> BoolComparePosL :=proc(P, Q) ComparePosL(P, Q) ≥ 0 end;
> ComparePosL(P1l, P2l); ComparePosL(P2l, P1l); ComparePosL(P2l, P2l);
```

```
> p1 := x→evalueListe(P1l, x); p1(X); p2 := x→evalueListe(P2l, x); p2(X);
> tabP := [P1l, P2l]; TriFusion(BoolCompareNegL, tabP);
> ech12 := [2, 1];
> map(k→ech12[k], [$1 ..2]);
> VerifierPermute :=proc(tab) map(i→tab[i], map(k→ech12[k], [$1 ..nops(tab)]))
      end;
> VerifierPermute(tabP);
>
```

Question 7

version listes

```
> Un nom comme EEAAA est à lire EstEchangeurAuxAuxAux
> permTestL := [4, 1, 8, 2, 7, 3, 5, 6]; n := 8;
> EEAAAL :=proc(perm, d, c, b) local n, mini, maxi, bon, a;
                  n := nops(perm);
                  mini := min(perm[d], perm[c]); maxi := max(perm[d], perm[c]);
                  bon := true; a := b + 1;
                  while (bon and a ≤ n) do bon := bon and ((perm[a] < mini)
      or (perm[a] > maxi)); a := a + 1 od;
                bon end:
> EEAAAL(permTest, 2, 4, 6);
> EEAAL :=proc(perm, d, c) local n, pc, pd, bon, b;
                  n := nops(perm);
                  pc := perm[c]; pd := perm[d];
                  bon := true; b := c + 1;
                  while (bon and b ≤ (n − 1)) do
                  bon := bon and
   ( (pc < pd) and (perm[b] < pd) or ((pc > pd) and (perm[b] > pd))
      or (EEAAAL(perm, d, c, b)));
                    b := b + 1 od;
                bon end:
> EEAAL(permTest, 3, 6);
> estEchangeurAuxL :=proc(perm, d) local n, bon, c;
                  n := nops(perm); bon := true; c := d + 1;
                  while (bon and c ≤ (n − 2)) do
                  bon := bon and EEAAL(perm, d, c);
                  c := c + 1 od;
                bon end;
>
```

version arrays

```
> arrayTest := alloue(8) :for k from 1 to 8 do arrayTest[k] := permTestL[k] od:
      arrayTest;
> EEAAAA :=proc(perm, d, c, b) local n, mini, maxi, bon, a;
                  n := taille(perm);
                  mini := min(perm[d], perm[c]); maxi := max(perm[d], perm[c]);
                  bon := true; a := b + 1;
```

```
                while (bon and a ≤ n) do bon := bon and ((perm[a] < mini)
        or (perm[a] > maxi)); a := a + 1 od;
            bon end:
> EEAAAA(arrayTest, 2, 4, 6);
> EEAAA :=proc(perm, d, c) local n, pc, pd, bon, b;
                n := taille(perm);
               pc := perm[c]; pd := perm[d];
               bon := true; b := c + 1;
               while (bon and b ≤ (n − 1)) do
               bon := bon and
    ( (pc < pd) and (perm[b] < pd) or ((pc > pd) and (perm[b] > pd))
        or (EEAAAL(perm, d, c, b) ));
                 b := b + 1 od;
            bon end:
> EEAAA(arrayTest, 3, 6);
> estEchangeurAuxA :=proc(perm, d) local n, bon, c;
                  n := taille(perm); bon := true; c := d + 1;
               while (bon and c ≤ (n − 2)) do
               bon := bon and EEAA(perm, d, c);
               c := c + 1 od;
            bon end;
>
>
```

Question 8

avec des listes

```
> estEchangeurL :=proc(perm) local bon, d;
               bon := true; d := 1;
               while (bon and d ≤ (n − 3)) do
               bon := bon and estEchangeurAuxL(perm, d);
               d := d + 1 od;
            bon end;
> estEchangeurL(permTestL);
> permMauvaisL := [8, 5, 4, 7, 3, 2, 1, 6]; permBonL := [1, 2, 3, 4, 5, 6, 7, 8];
> estEchangeurL(permMauvaisL); estEchangeurL(permBonL);
> EEAAAL(permMauvaisL, 3, 4, 6);
> EEAAL(permMauvaisL, 3, 4);
```

avec des arrays

```
> estEchangeurA :=proc(perm) local bon, d;
               bon := true; d := 1;
               while (bon and d ≤ (n − 3)) do
               bon := bon and estEchangeurAuxA(perm, d);
               d := d + 1 od;
            bon end;
> estEchangeurA(arrayTest);
> arrayMauvais := alloue(8) :for k from 1 to 8 do arrayMauvais[k]
```

```
        := permMauvaisL[k] od: arrayMauvais;
    arrayBon := alloue(8) :for k from 1 to 8 do arrayBon[k] := permBonL[k] od:
        arrayBon;
> estEchangeurA(arrayMauvais); estEchangeurA(arrayBon);
> EEAAAA(arrayMauvais, 3, 4, 6);
> EEAAA(arrayMauvais, 3, 4);
```

Question 9

```
> NombreEchangeurs :=proc(n) local ne, k, i, somme; ne := alloue(n);
                    ne[1] := 1; ne[2] := 2; ne[3] := 6;
                    for k from 4 to n do
                        somme := ne[k − 1];# print('début', k, somme);
                        for i from 1 to (k − 1) do somme := somme + ne[i]·ne[k − i] od;
      ne[k] := somme;# print(k, somme);
                    od;
                     somme end;
> NombreEchangeurs(4);
> NombreEchangeurs(5); NombreEchangeurs(6); NombreEchangeurs(7);
```

Question 10

version Arrays

```
> decalerA :=proc(t, v) local i, n, new;
        n := taille(t); new := alloue(n + 1);
        new[1] := v ;
        for i from 2 to (n + 1)
        do   if (t[i − 1] < v)
             then new[i] := t[i − 1]
             else new[i] := 1 + t[i − 1]
             end if
        end do;
    new end:
> arrayTest :=proc( ) local z; z := alloue(8);for k from 1 to 8 do z[k]
        := permMauvais[k] end do; z end;
> decalerA(decalerA(arrayTest( ), 3), 6);
```

version listes

```
> decalerL :=proc(t, v) local i, n, new;
        n := nops(t); new := [$1 ..(n + 1)];
        new[1] := v ;
        for i from 2 to (n + 1)
        do   if (t[i − 1] < v)
             then new[i] := t[i − 1]
             else new[i] := 1 + t[i − 1]
             end if
        end do;
    new end:
```

```
> decalerL(decalerL(permMauvais, 3), 6);
>
```

Question 11

> Pour qu'une permutation soit un échangeur, il faut que toutes ses sous permutations aient la propriété (1). Donc, au lieu de tester toutes les permutations, on prend les échangeurs de 1..n, par utilisation de la fonction decaler on les préfixe par 1,2...(n+1) et ce sont ces permutations là que l'on teste par estEchangeurAux

> Maple vient de me refuser toutes mes tentatives de manipulations d'Array du type Array. Cette question 11 va donc être limitée à la version listes de listes

```
> LE := proc(n) if n = 3 then [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]] fi
      end;
> map(z→decalerL(z, 1), LE(3));
> S4 := map(u→op(map(z→decalerL(z, u), LE(3))), [$1..4]);
> LE4 := select(estEchangeurL, S4);
> nops(LE4);
> Poss5 := map(u→op(map(z→decalerL(z, u), LE4)), [$1..5]);
> LE5 := select(estEchangeurL, Poss5);
> nops(LE5);
```