Assignment 4 Question 2


```
> Greater := proc(a::list(nonnegint),b::list(nonnegint))
  # compare two monomials a and b input as exponent vectors
  # in the graded monomial ordering.  Output true if a>b else false.
  local dega,degb,n,i,z;
     n := nops(a);
     if nops(b)<>n then error "exponent vectors of different sizes"
  fi;
     dega := add(z,z in a); # = deg(a)
     degb := add(z,z in b); # = deg(b)
     if dega>degb then return true elif dega<degb then return false
  fi;
     for i to n do # compare with lexicographical order
         if a[i]>b[i] then return true;
         elif a[i]<b[i] then return false;
         fi;
     od;
     false;
  end:

  IsSorted := proc(f)
  local i;
     for i to nops(f)-1 do
         if Greater(f[i][2],f[i+1][2]) then else return false fi;
     od;
     true;
  end:
```

```
> Maple2SDMP := proc(a,X::list(name))
  local C,M,n,t,A,B,E,i,m;
     if not type(a,polynom(rational,X)) then error "bad input" fi;
     C := [coeffs(a,X,'M')];
     M := [M];
     n := nops(X);
     t := nops(M);
     E := [seq( [seq(degree(m,X[i]),i=1..n)], m in M)]; # exponent
  vectors
     A := [seq([C[i],E[i]],i=1..t)];
     B := sort(A,proc(x,y) Greater(x[2],y[2]) end);
     B;
  end:

  SDMP2Maple := proc(A::list([rational,list(nonnegint)]),X::list
  (name))
  local t,i,n;
     n := nops(X);
     add(t[1]*mul(X[i]^t[2][i],i=1..n),t in A);
  end:
```

```
> a := 3*x^2*y^2*z+2*y^3*z-5*x*y*z;
  A := Maple2SDMP(a,[x,y,z]);
  SDMP2Maple(A,[x,y,z]);
  b := 3/2*x^2+5/2*x*y^2-7/2*y^4;
  B := Maple2SDMP(b,[x,y]);
  SDMP2Maple(B,[x,y]);
```

$$a := 3x^2y^2z + 2y^3z - 5xyz$$

$$A := [[3, [2, 2, 1]], [2, [0, 3, 1]], [-5, [1, 1, 1]]]$$

$$3x^2y^2z + 2y^3z - 5xyz$$

$$b := \frac{3}{2}x^2 + \frac{5}{2}xy^2 - \frac{7}{2}y^4$$

$$B := \left[\left[-\frac{7}{2}, [0, 4]\right], \left[\frac{5}{2}, [1, 2]\right], \left[\frac{3}{2}, [2, 0]\right]\right]$$

$$\frac{3}{2}x^2 + \frac{5}{2}xy^2 - \frac{7}{2}y^4 \quad \textbf{(1)}$$

```
> `type/multipoly` := proc(f) type(f,list([rational,list(nonnegint)])
  ) end:
```

```
> type(A,multipoly);
```

$$true \quad \textbf{(2)}$$

```
> AddPol := proc( f::multipoly, g::multipoly)
  local m,n,h,i,j,k,c;
  global N;
      # Compute h = f + g using merging
      m,n := nops(f),nops(g);
      h := Array(1..m+n);
      i,j,k := 1,1,0;
      while i<=m and j<=n do
          N++;
          if f[i][2] = g[j][2] then
             c := f[i][1]+g[j][1];
             if c<>0 then k++; h[k] := [c,f[i][2]]; fi;
             i++; j++;
          elif Greater(f[i][2],g[j][2]) then k++; h[k] := f[i]; i++;
          else k++; h[k] := g[j]; j++;
          fi;
      od;
      while i<=m do k++; h[k] := f[i]; i++; od;
      while j<=n do k++; h[k] := g[j]; j++; od;
      h := [seq( h[i], i=1..k )];
  end:
```

```
> MulPolM := proc(f::multipoly,g::multipoly)
  # Multiply f by g using repeated merging
  local h,i,s,t,T;
```

```
  global M;
     h := []; # h = 0
     for i to nops(f) do
        s := f[i];
        # Multiply the coefficients and the exponent vectors (Maple
  lists)
        T := [seq( [s[1]*t[1], s[2]+t[2]], t in g )];
        M += nops(g);
        h := AddPol(h,T);
     od;
     h;
  end:
```

```
> MulPolDC := proc(f::multipoly,g::multipoly)
  # Multiply f by g using divide and conquer
  local m,n,c,e,term,k,a,b,h;
  global M;
     m := nops(f);
     n := nops(g);
     if m=1 then
         c := f[1][1];
         e := f[1][2];
         M += nops(g);
         [seq( [c*term[1],e+term[2]], term in g )];
     else
         k := iquo(m,2);
         a := MulPolDC(f[1..k],g);
         b := MulPolDC(f[k+1..m],g);
         h := AddPol(a,b);
     fi;
  end:
```

```
> HeapMult := proc(f,g)
  # Multiply f by g using a binary Heap
  # Copyright, Michael Monagan, November 2023
  local m,n,H,i,j,k,Z,c,h,gtr,term;
  global M;
     m,n := nops(f),nops(g);
     if m=0 or n=0 then return [] fi;
     gtr := proc(s,t) global N; N++; Greater(t[3],s[3]) end;
     # Here s = [i,j,u] and t = [k,l,v] so we want to compare u with
  v.
     # A Maple heap is a min heap and we want a max heap so switch
  the order
     H := heap[new](gtr);
     heap[insert]([1,1,f[1][2]+g[1][2]],H);
     k := 0;
     h := table();
     while not heap[empty](H) do
```

```
      term := heap[extract](H);
      i,j,Z := op(term);
      if j=1 and i<m then heap[insert]([i+1,1,f[i+1][2]+g[1][2]],H)
; fi;
      if j<n then heap[insert]([i,j+1,f[i][2]+g[j+1][2]],H) fi;
      c := f[i][1]*g[j][1]; M++;
      while not heap[empty](H) and heap[max](H)[3]=Z do
         term := heap[extract](H);
         i,j,Z := op(term);
         if j=1 and i<m then heap[insert]([i+1,1,f[i+1][2]+g[1][2]
],H); fi;
         if j<n then heap[insert]([i,j+1,f[i][2]+g[j+1][2]],H) fi;
         c := c+f[i][1]*g[j][1]; M++;
      od;
      if c<>0 then k++; h[k] := [c,Z]; fi;
   od;
   [seq( h[i], i=1..k )];
end:
```

```
> u,v := 'u,v';
  X := [u,v,w,x,y,z];
  #a := 3*x*y+5*y:
  #b := 2*x*y+3*x+17*z^2+5*z:
  a := randpoly(X,degree=10,terms=20):
  b := randpoly(X,degree=10,terms=1000):
  c := expand(a*b): # the answer
  A := Maple2SDMP(a,X):
  B := Maple2SDMP(b,X):
  C := Maple2SDMP(c,X):
  IsSorted(A);
  IsSorted(B);
  IsSorted(C);
```

$$u, v := u, v$$

$$X := [u, v, w, x, y, z]$$

$$true$$

$$true$$

$$true \qquad (3)$$

```
> N,M := 0,0: H := MulPolM(A,B):
  N,M; S := evalf(N/M,4);
  evalb(H=C);
  N,M := 0,0: H := MulPolDC(A,B):
  N,M; S := evalf(N/M,4);
  evalb(H=C);
  N,M := 0,0: H := HeapMult(A,B):
  N,M; S := evalf(N/M,4);
  evalb(H=C);
```

195435, 19880

$S := 9.831$

true

82722, 19880

$S := 4.161$

true

130998, 19880

$S := 6.589$

true **(4)**

If #A is small compared to #B then the divide an conquer algorithm is very good; it does the fewest comparisons. Let's see what happens if we interchange A and B

```
> N,M := 0,0: H := MulPolM(B,A):
  N,M; S := evalf(N/M,4);
  evalb(H=C);
  N,M := 0,0: H := MulPolDC(B,A):
  N,M; S := evalf(N/M,4);
  evalb(H=C);
  N,M := 0,0: H := HeapMult(A,B): N,M;
  S := evalf(N/M,4);
  evalb(H=C);
```

9283423, 19880

$S := 467.0$

true

189297, 19880

$S := 9.522$

true

130998, 19880

$S := 6.589$

true **(5)**

It is quite surprising how bad the repeated merging is. It is approaching 100 times slower than the heap algorithm which is the best.