Question 1.

```
struct Node
    coeff
    exp
    prev
    next
function makeSentinel()
    temp
    temp.next=temp
    temp.prev=temp
    return temp
function insertNodeAtEnd(sentinel, currentNode, coeff, exp)
    temp
    temp.coeff=coeff
    temp.exp=exp
    temp.prev=currentNode
    temp.next=sentinel
    currentNode.next=temp
    sentinel.prev=temp
function add(pSentinel,p,qSentinel,q,rSentinel,r)
    while(p!=pSentinel and q!=qSentinel)
        if(p.exp < q.exp)
            insertNodeAtEnd(rSentinel,r,p.coeff,p.exp)
            r=r.next
            p=p.next
        else if(p.exp>q.exp)
            insertNodeAtEnd(rSentinel,r,q.coeff,q.exp)
            r=r.next
            q=q.next
        else
```

```
            if(p.coeff+q.coeff != 0)
                insertNodeAtEnd(rSentinel,r,p.coeff+q.coeff,p.exp)
                r=r.next
                p=p.next
                q=q.next
    while(p!=pSentinel)
        insertNodeAtEnd(rSentinel,r,p.coeff,p.exp)
        r=r.next
        p=p.next
    while(q!=qSentinel)
        insertNodeAtEnd(rSentinel,r,q.coeff,q.exp)
        r=r.next
        q=q.next
    //Reset the pointers to point to the first element after the Snetinel node or possibly it before returning
    r=rSentinel.next
    p=pSentinel.next
    q=qSentinel.next
    return r
function main()
    Input(n,m)
    //Initialising sentinel nodes of p and q polynomials
    pSentinel=makeSentinel()
    qSentinel=makeSentinel()
    //Making copies of pointers to traverse the list
    p=pSentinel
    q=qSentinel
    for 1 to n
        Input(coeff,exp)
        insertNodeAtEnd(pSentinel,p,coeff,exp)
        p=p.next
```

```
p=pSentinel.next //Resetting pointer to the first element after Sentinel node
for 1 to m
    Input(coeff,exp)
    insertNodeAtEnd(qSentinel,q,coeff,exp)
    q=q.next
q=qSentinel.next //Resetting pointer to the first element after Sentinel node
rSentinel =makeSentinel//Making a node to store final polynomial
r=rSentinel
r=add(pSentinel,p,qSentinel,q,rSentinel,r)
x=0
while(r!=rSentinel)
    print(r.coeff,r.exp)
    x=1
if(x==0) //When the sum was zero polynomial
    print("0")
```

Question 2.

```
struct Node
    coeff
    exp
    prev
    next
function makeSentinel()
    temp
    temp.next=temp
    temp.prev=temp
    return temp
function insertNodeAtEnd(sentinel, currentNode, coeff, exp) //Same as question1
    temp
    temp.coeff=coeff
    temp.exp=exp
    temp.prev=currentNode
    temp.next=sentinel
    currentNode.next=temp
    sentinel.prev=temp
function add(pSentinel,p,qSentinel,q,rSentinel,r) //Same as question1
    while(p!=pSentinel and q!=qSentinel)
        if(p.exp < q.exp)
            insertNodeAtEnd(rSentinel,r,p.coeff,p.exp)
            r=r.next
            p=p.next
        else if(p.exp>q.exp)
            insertNodeAtEnd(rSentinel,r,q.coeff,q.exp)
            r=r.next
            q=q.next
        else
```

```
                if(p.coeff+q.coeff != 0)
                    insertNodeAtEnd(rSentinel,r,p.coeff+q.coeff,p.exp)
                    r=r.next
                    p=p.next
                    q=q.next
    while(p!=pSentinel)
        insertNodeAtEnd(rSentinel,r,p.coeff,p.exp)
        r=r.next
        p=p.next
    while(q!=qSentinel)
        insertNodeAtEnd(rSentinel,r,q.coeff,q.exp)
        r=r.next
        q=q.next
    //Reset the pointers to point to the first element after the Snetinel node or possibly it before returning
    r=rSentinel.next
    p=pSentinel.next
    q=qSentinel.next
    return r
//Multiplies a polynomial p with a monomial whose coefficient and exponent has been passed to the function
function multiply(pSentinel,p,coeff,exp,sSentinel,s)
    while(pSentinel!=p)
        insertNodeAtEnd(sSentinel,s,p.coeffcp.exp+exp)
        s=s.next
        p=p.next
    p=pSentinel.next
    s=sSentinel.next
    return s
function main()
    Input(n,m)
    //Initialising sentinel nodes of p and q polynomials and pointers to traverse them
```

```
pSentinel=makeSentinel()
p=pSentinel
qSentinel=makeSentinel()
q=qSentinel
for 1 to n
        Input(coeff,exp)
        insertNodeAtEnd(pSentinel,p,coeff,exp)
        p=p.next
p=pSentinel.next //Resetting pointer to the first element after Sentinel node
for 1 to m
        Input(coeff,exp)
        insertNodeAtEnd(qSentinel,q,coeff,exp)
        q=q.next
q=qSentinel.next //Resetting pointer to the first element after Sentinel node
//Making 3 new nodes r,s,t and pointers to traverse them for the computation process
rSentinel =makeSentinel()
r=rSentinel
sSentinel =makeSentinel()
s=sSentinel
tSentinel =makeSentinel()
t=tSentinel
while(q!=qSentinel)
        s=multiply(pSentinel,p,q.coeff,q,exp,sSentinel,s)
        t=add(sSentinel,s,rSentinel,r,tSentinel,t)
        r=t //Copying the result of initial addition for next iteration
        rSentinel=r.prev //Resetting position of rSentinel after r is returned
        q=q.next //Moving to next element in polynomial q
        Emptying lists s and t before next iteration
        sSentinel.next=sSentinel
        sSentinel.prev=sSentinel
        s=sSentinel
        tSentinel.next=t
        tSentinel.prev=t
        t=tSentinel
x=0
while(r!=rSentinel)
        print(r.coeff,r.exp)
        x=1
if(x==0) //When the product was zero polynomial
        print("0")
```

Runtime Analysis:

1. We input the lists p and q which takes O(n) and O(m) time respectively.
2. The while loop i.e. while(q!=qSentinel) {…}

   Runs for m number of iterations i.e. O(m) since in each iteration we necessarily traverse to the next element of the linked list q and there are exactly m elements in the list q and we exit the list as soon as we reach the sentinel node(which comes immediately after the last i.e mth element). Also, note that we do not modify the elements of q at any time.

   a. The multiply function takes O(|p|)=O(n) time since we traverse through all the monomials of the polynomial i.e. n and multiplying them once to the coefficient and exp and inserting the node to end of s which takes constant amount of time in each iteration.

   b. The add function takes O(|s|+|r|) = O(n + |r|) <= O(n + mn) = O(m*n) time. We can see this as follows:

      The first loop in add function traverses through the lists p and q and increments the value of either p or q or both in each iteration (and does not modify them). The loop exits as soon as p==pSentinel or q==qSentinel or both so it runs for a maximum of O(|p|+|q|) time (in our case p and q are s and t).

      The second loop traverses through the list p and for the same reason as above takes O(|p|) time. The third loop, similarly, takes O(|q|) time.

      Therefore the add function takes O(|p|+|q|)+O(|p|)+O(|q|) =O(|p|+|q|) time.

      Now we pass s and r as the parameters p and q. s is the product of the polynomial p and a term from polynomial q and hence contains |p| = n elements. r is the product of first i monomials of q with polynomial p and hence contains at most i*n elements which is less than or equal to m*n elements (since we do not allow duplicate elements in our lists).

      Hence, from above the add function takes O(n + n*m)=O(n*m) time.

      (Note that for a more tight argument we may take the its complexity to be O(i*p) and do a summation over i to m for each iteration . This however simply results in an A.P. and the order remains O(m*n) and only a constant factor changes.)
3. The printing step takes O(r) = O(m*n) time.

Therefore, the total order = O(m) + O(n) + O(m)*[O(n) + O(n*m))] = O (n* m^2) time.

(Neglecting the lower order terms for asymptotic analysis)

Also note that we may add a condition to check which of n or m is lower say x and the higher one being y then make the code O(x^2*y) just by changing the looping variable to x.