Ques 2.)

Main Driver Code

```
//function that returns the pointer which points to the middle element of
the list
// Function to find the pointer to the central element in a list.
// This is done using two pointers one which moves twice as fast
// as the other. When the faster pointer reaches the end of the
// list,the slower pointer points to the central element.

function middlePtr(list):
    slow=list.next   //Initialize the slow pointer
    fast=list.next.next  //Initialize the faster pointer

    while(fast.next!=list and fast.next.next!=list) do:
        slow=slow.next
        fast=fast.next.next
    return slow

//function to merge two sorted lists
function merge(p,q):
    c1=p.next
    c2=q.next

    sentinel=make_node(0,0)
    sentinel.next=sentinel.prev=sentinel

    while(c1!=p and c2 !=q) do:
        if(c1.exp<c2.exp):
            append(c1,sentinel)
            c1=c1.next
        else if(c2.exp<c1.exp):
            append(c2,sentinel)
            c2=c2.next
        else :
            a=c1.coeff+c2.coeff
            if(a):
                c=make_node(a,c1.exp)
                append(c,sentinel)
            c1=c1.next
            c2=c2.next

    c = (c1==p)?c2:c1
```

```
    while(c!=p and c!=q) do:
        append(c,sentinel)

    return sentinel

//function to sort the list
function sort(r):
    if(r.next==r or r.next.next==r) return r //If one element,then sorted
    centerPtr=middlePtr(r)  //Get the middle pointer
    alt_sentinel=make_node(0,0) //Split the list at the middle pointer
    alt_sentinel.next=centerPtr.next
    alt_sentinel.prev=r.prev
    centerPtr.next.prev=alt_sentinel
    r.prev=centerPtr
    centerPtr.next=r

    final=merge(sort(r),sort(alt_sentinel)) //Merge the two sorted lists
    return final

//function for multiplying two polynomial and return a list which contains
all the multiplied terms
function multiply(p,q):
    r=make_node(0,0)
    r.next=r.prev=r
    for(i=p.next;i!=p;i=i.next):
        for(j=q.next;j!=q;j=j.next):
            temp=make_node(p.coeff*q.coeff,p.exp+q.exp)
            append(temp,r)
    return r

//Main function
function main():
    n,m
    cin>>n>>m
    p=input(n)
    q=input(m)
    r=multiply(p,q)
    finalAns=sort(r)
    print(finalAns)
```

Analysis of time Complexity

Scanning the input list and Printing the output list:

Scanning a input list of size n takes linear time .So time taken to scan p and q polynomial will be O(2m+2n) or O(m+n).

Similarly the final output list can have maximum of 2*(mn) elements considering the fact that no two terms had the same exponent. So time taken to print the list will be O(2mn).

So these both times will be added to the final time.

Multiplication Part:

In the multiply function we multiply every term of polynomial p with every term of polynomial q with the help of two nested for loops (outer one runs n times) and for every iteration of outer loop inner loop runs m times. So overall time taken will be O(m*n).We append all the multiplied terms in another list.

Merge Sort Part : For sorting the multiplied array(contains all the multiplied terms)

Now we apply the merge sort algorithm on the linked list we have now obtained. We will consider the time required for this operation alone and then add it to the time calculated earlier.

Let us suppose the list to be sorted has k elements.

Merge Sort is a standard algorithm used for sorting unsorted arrays and linked lists.:

Step 1: An unsorted linked list is divided into two halves and those two halves are sorted separately.(Divide and Sort)

Step 2: the two separate sorted half parts of the lists are merged together such that the entire list becomes sorted. (Union)

So lets assume that step 1 of sorting a list takes time T(n) where n is the no of elements in the list.

Step 2 of merging two sorted list(each containing n/2 elements) takes O(n) time because we iterate over every element of those two lists and place the elements in ascending(or descending) order in the third list.

So let's say we get a list containing n elements to be sorted. So this will involve mainly four processes.

1.Finding the pointer which points to the middle of the list.

2.Sorting of first half of the list.

3.Sorting of second half of the list.

4.Merging the list obtained in 1 and 2.

So if the list contains n elements then the time take by the function middle ptr will be some constant times n/2 and there will be certain more steps after that which will take constant time.

Hence time taken by step 1 is A*(n/2) + B.

So the expression obtained is:

T(n) = A*(n/2) + B + 2*T(n/2) + O(n) where T(n) is the time taken to sort n elements list

We also know that a list containing n elements can be divided into maximum $\log_2 n$ parts as

$2^k <= n \Rightarrow k <= \log_2 n$

So by solving above equation we will get :

$T(n)=O(n) + 2*O(n/2) + 2^2 * O(n/2^2) + \ldots\ldots\ldots\ldots\ldots + 2^k * O(n/2^k) + kB + A * (n/2 + n/4 + \ldots\ldots + n/2^k)$

Where $k=\log_2 n$

$T(n)<c1*n + 2 * c2 * n/2 + 2^2 * c3 * n/2^2 + \ldots\ldots\ldots\ldots\ldots + 2^k * ck * n/2^k + B * \log_2 n + A * n * \log_2 n$

Let c =max(c1,c2,.....ck)

$T(n)<c * n * k + B * \log_2 n + A * n * \log_2 n$

$T(n)< c* n * \log_2 n + B * \log_2 n + A * n * \log_2 n$

$T(n) < c1 * n * \log_2 n + B * \log_2 n$

$T(n) < c1 * n * \log_2 n + B * n * \log_2 n$

$T(n) < c2 * n * \log_2 n$

Hence time complexity of merge sort is $O(n*\log_2 n)$ of list containing n elements.

So our above lists contain mn elements. So merge Sort will take O(mn *log(mn)) time.

Total time taken= O(mn)+ O(mn *log(mn)) + O(m+n) + O(mn) [last two terms for print and input]

Total time taken= 2* O(mn)+ O(mn *log(mn)) + O(m+n)

For large values of m and n dominating term will be O(mn *log(mn)).

Hence overall time complexity of our algorithm is O(mn *log(mn)).