Assignment Problem1-

[redacted]

Pseudo-Code:

Add_Poly and InsertNode functions carry out the addition of two polynomials, a and b.

- InsertNode (head, Coefficient, Exponent)

```
Poly temp
temp->coeff = coefficient
temp->expo = exponent
Poly last = head->prev
temp->next=head
head->prev=temp
temp->prev=last
last->next=temp
```

- Add_Poly (a, b)

```
Poly sumpoly      //sentinel node
Poly poly1 = a
Poly poly2 = b
a = a->next
b = b->next
while a != poly1 and b != poly2 do
    if  polya->expo < polyb->expo
        InsertNode(sumpoly,a->coeff,a->expo)
        a = a->next
    else if  polyb->expo < polya->expo
```

```
                InsertNode(sumpoly,b->coeff,b->expo)
                b = b->next
        else
                if polya->coeff + polyb->coeff != 0
                        InsertNode(sumpoly,a->coeff + b->coeff,a->expo)
                a = a->next
                b = b->next

while a !=poly1 do
        InsertNode(sumpoly,polya->coeff,polya->expo)
        a = a->next
while b != poly2 do
        InsertNode(sumpoly,polyb->coeff,polyb->expo)
        b = b->next

return sumpoly
```

Run-Time Complexity:

- InsertNode takes constant time (8 steps)
- Add_Poly takes
  - 5 steps in constant time
  - The loops together in worst case run (n+m) times, where n and m are the sizes of the polynomials a and b.
  - Each loop contains an if statement, an InsertNode and an increment operation, i.e. (n+m)*(1+8+1)
  - Single step return statement
  - Thus, overall the no. of steps become 5+10*(n+m)+1 = 6+10(n+m)

For simplification, we assume single step constant time operations to take unit time, in such a case the total time taken for execution of the Addition Operation =

6+10(n+m) ≤ **K(n+m)**

Hence the addition operation algorithm has a **Runtime Complexity of O(n+m)**.

Assignment Problem 2-

[redacted]

Pseudo-Code:

Multi_Poly is the function that multiplies the two polynomials using functions Add_Poly and InsertNode

- InsertNode: Defined in the previous question
- Add_Poly(a, b) : Defined in the previous question
- Multi_Poly(p, q)

```
Polynomial Final
while p != p_sentinel do
    Polynomial CurrSum
    Polynomial Iterator = q
    while Iterator != q_sentinel do
        InsertNode(CurrSum, p->coeff*Iterator->coeff, p->expo+Iterator->expo)
        Iterator = Iterator->next
    Final = Add_Poly(Final, CurrSum)
    p = p->next
return Final
```

Run-Time Complexity:

- InsertNode takes constant time (8 steps)
- Add_Poly takes
    - 5 steps in constant time
    - Loops which run for a+b times, where a and b are the sizes of polynomial a and polynomial b, each loop contains an if statement, an InsertNode and an increment operation, i.e. (a+b)*(1+8+1)
    - Single step return statement
    - Thus, overall, the no. of steps become 5+10*(a+b)+1 = 6+10*(a+b)

Multi_Poly contains :

- A single step declaration in constant time
- A loop over polynomial p, i.e., running n times, the loop contains:
    - 2 single step declarations in constant time
    - A loop iterating over q, i.e., running m times, the loop contains:
        - An InsertNode and an increment operation taking 8+1 steps overall
    - An Add_Poly operation, CurrSum has length m and Final can have the length n*m in the worst case, thus total steps in this operation become 6+10(m+mn)
    - A single step increment operation taking constant time
- Single step Return Statement

For simplification, we assume single step constant time operations to take unit time, in such a case the total time taken for execution of the Multiplication Operation =

$1 + n*(2 + m*(8+1) + 6 + 10*(m+mn) + 1) + 1 =$

$1 + 2n + 9mn + 6n + 10mn + 10mn^2 + n + 1 =$

$10mn^2 + 9mn + 9n + 2 \leq \mathbf{K(mn^2)}$ [for a sufficiently large constant K]

Hence the multiplication operation algorithm has a **Runtime Complexity of $O(mn^2)$.**