pseudocode

```
addNode(Node head, double coeff, int power):
    if (head = null):
        head = new Node(null, coeff, power)

    else:
        Node cur = head
        Node toAdd = new Node(null, coeff, power)
        if (cur.next = null):
            if(power < cur.power):
                cur.next = toAdd

            else:
                toAdd.next = head
                head = toAdd

        else:
            while (power < cur.next.power):
                cur = cur.next
                if(cur.next = null):
                    break

            toAdd.next = cur.next
            cur.next = toAdd
```

{ if linkedlist is empty

{ if there is only 1 element

{ if there are multiple elements
iterate over the linkedlist until
finding the right place

```
add(Node poly1, Node poly2):
    Node temp1 = poly1.power ≥ poly2.power ? poly1 : poly2
    Node temp2 = poly1.power ≥ poly2.power ? poly2 : poly1
    Node result = null
    while (temp1 != null):
        Boolean found = false
        while (temp2 != null):
            if (temp1.power = temp2.power):
                result = HW2.addNode (result, temp1.coeff + temp2.coeff, temp1.power)
                found = true
            temp2 = temp2.next
```

{ if two elements have same power

```
        if not found:                                   // we only add elements of poly1 to result
            result = HW2.addNode(result, temp1.coeff, temp1.power)   // if the power of node in poly1 does not exist in poly2
        temp2 = poly1.power ≧ poly2.power ? poly2 : poly1
        temp1 = temp1.next
    temp1 = poly1.power ≧ poly2.power ? poly1 : poly2
    temp2 = poly1.power ≧ poly2.power ? poly2 : poly1
    Node temp3 = result
    while (temp2 != null):
        Boolean found = false
        while (temp3 != null):
            if (temp2.power = temp3.power)              // this element in poly2 is already in result
                found = true
            temp3 = temp3.next
        if (!found)
            result = HW2.addNode (result, temp2.coeff, temp2.power)
        temp3 = result
        temp2 = temp2.next
    return result

& multiply (Node poly1, Node poly2):
    Node result = null
    Node temp1 = poly1
    Node temp2 = poly2
    while (temp1 != null):
        while (temp2 != null)
            if (result = null)
                result = HW2.addNode (result, temp1.coeff * temp2.coeff, temp1.power + temp2.power)   // add 1st element to result
            else:
                result = HW2.add (result, new Node(null, temp1.coeff * temp2.coeff, temp1.power + temp2.power))   // multiply each elements and add to result
            temp2 = temp2.next
        temp2 = poly2
        temp1 = temp1.next
    return result
```

Time and Space Complexity

addNode: The time complexity is $O(n)$ because it does $c_1 \cdot n$ operations to find the correct location for the node to be added in the worst case, which is adding to the end. The space complexity is $O(1)$ because it is not recursive and it is returned without any other function call.

add: The time complexity is $O(n^2)$. In the first part of add, for each element of poly1, it iterates all elements of poly2. Let's assume the number of elements of poly1 is i and j for poly2. For i times, it does $c_1 \times j + c_2$ (the length of result) operations. The length of result comes from the complexity of addNode. So our total sum of operations:

$$c_1 j + c_2 1 + c_1 j + c_2 2 + c_1 j + c_2 3 + \dots + c_1 j + c_2 i = c_1 ij + c_2 \frac{(j^2+i)}{2}$$

In the second part, for each element in poly2, it iterates all elements of result. In the worst case, it adds all elements of poly2 to result and the number of operations:

$$c_3 i + c_4 1 + c_3 i + c_4 2 + \dots + c_3 i + c_4 j = c_3 ij + c_4\left(\frac{j^2+j}{2}\right)$$

So, the number of operations is quadratic, therefore time complexity is $O(n^2)$.
The space complexity is $O(1)$ because it is not recursive and there are at most 2 concurrent functions in the stack.

multiply: The time complexity is $O(n^4)$. At first when adding the first element to head, it does constant operations. Then it adds $i \times j$ elements to the linkedlist i.e. it call add function $ij - 1$ times. In each add function, there are quadratic operations, therefore since we also call add quadratic times, the time complexity is $O(n^4)$.
The space complexity is $O(1)$ because it is not recursive and there are at most 2 concurrent functions in the stack.

I have completed this assignment individually, without support from anyone else. I hereby accept that only the below listed sources are approved to be used during this assignment:

(i) Course textbook,

(ii) All material that is made available to me by the professor (e.g., via Blackboard for this course, course website, email from professor / TA),

(iii) Notes taken by me during lectures.

I have not used, accessed or taken any unpermitted information from any other source. Hence, all effort belongs to me.

Umur Berkay Karakaş