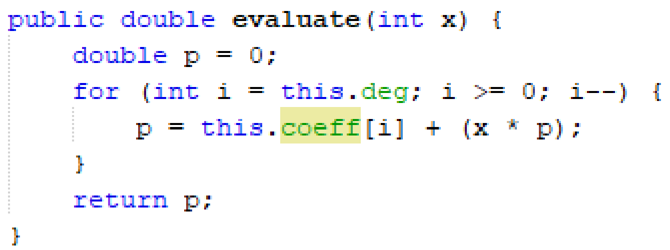
Please “paste” a copy of your code for the methods discussed at the end of this document.

1) For your array implementation of the Polynomial class, what would be the best value to use for the *“size of data” n* that will be used to discuss time/space complexity?

____.

a) What is the runtime of the method *p.evaluate(x) (*in terms of *n) ?*

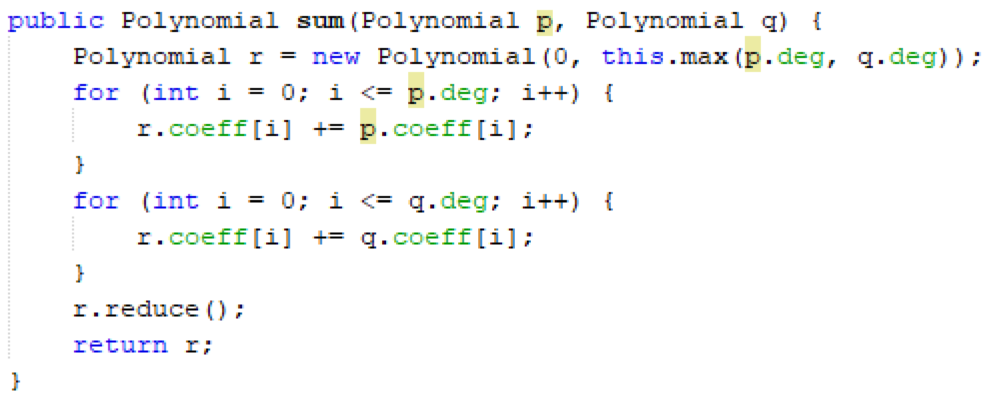
Ans) O(n)



```
public double evaluate(int x) {
    double p = 0;
    for (int i = this.deg; i >= 0; i--) {
        p = this.coeff[i] + (x * p);
    }
    return p;
}
```

b) What is the runtime of the method *sum (p, q)* (in terms of *n)*?

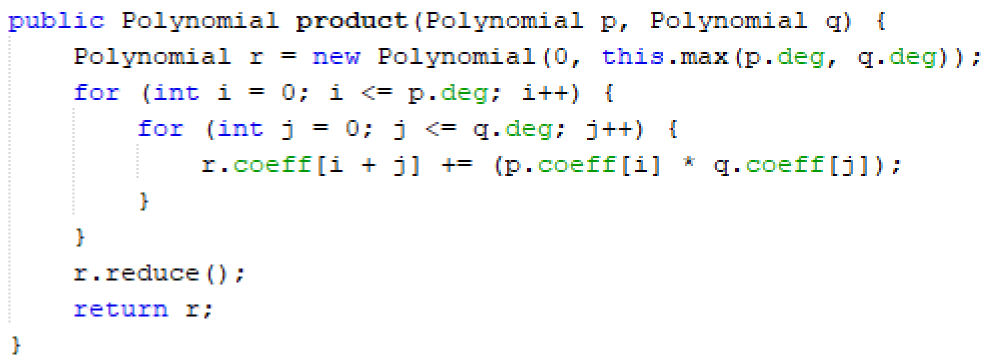
Ans) O(n) + O(n) = 2* O(n) => O(n)



```
public Polynomial sum(Polynomial p, Polynomial q) {
    Polynomial r = new Polynomial(0, this.max(p.deg, q.deg));
    for (int i = 0; i <= p.deg; i++) {
        r.coeff[i] += p.coeff[i];
    }
    for (int i = 0; i <= q.deg; i++) {
        r.coeff[i] += q.coeff[i];
    }
    r.reduce();
    return r;
}
```

c) What is the runtime of the method *product (p, q)* (in terms of *n*)?

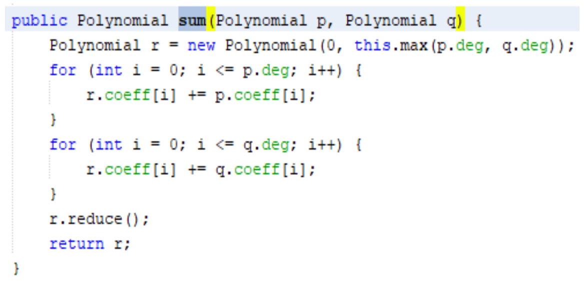
Ans) $O(n^2)$



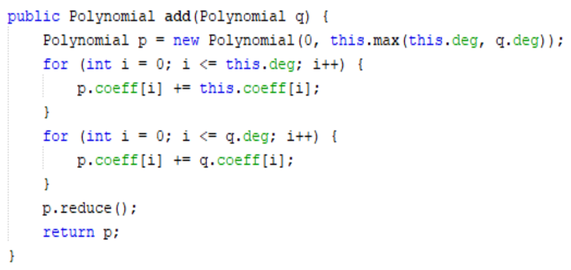
```
public Polynomial product(Polynomial p, Polynomial q) {
    Polynomial r = new Polynomial(0, this.max(p.deg, q.deg));
    for (int i = 0; i <= p.deg; i++) {
        for (int j = 0; j <= q.deg; j++) {
            r.coeff[i + j] += (p.coeff[i] * q.coeff[j]);
        }
    }
    r.reduce();
    return r;
}
```

d) What is the difference between sum and add?

Ans) Add () Returns the resulting polynomial after adding a polynomial q to the given polynomial whereas Sum () Returns the resulting polynomial after summing two polynomials p and q



```
public Polynomial sum(Polynomial p, Polynomial q) {
    Polynomial r = new Polynomial(0, this.max(p.deg, q.deg));
    for (int i = 0; i <= p.deg; i++) {
        r.coeff[i] += p.coeff[i];
    }
    for (int i = 0; i <= q.deg; i++) {
        r.coeff[i] += q.coeff[i];
    }
    r.reduce();
    return r;
}
```



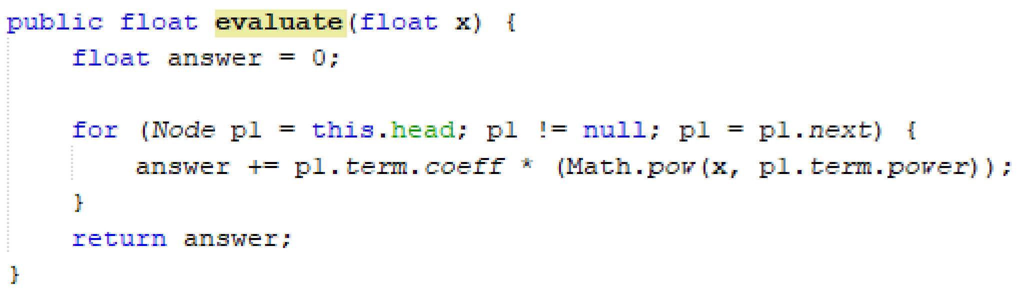
```
public Polynomial add(Polynomial q) {
    Polynomial p = new Polynomial(0, this.max(this.deg, q.deg));
    for (int i = 0; i <= this.deg; i++) {
        p.coeff[i] += this.coeff[i];
    }
    for (int i = 0; i <= q.deg; i++) {
        p.coeff[i] += q.coeff[i];
    }
    p.reduce();
    return p;
}
```

2) For your linked implementation of the Polynomial class, what would be the best value to use for the *"size of data" n* that will be used to discuss time/space complexity?

____.

a) What is the runtime of the method *evaluate(x) (*in terms of *n)?*

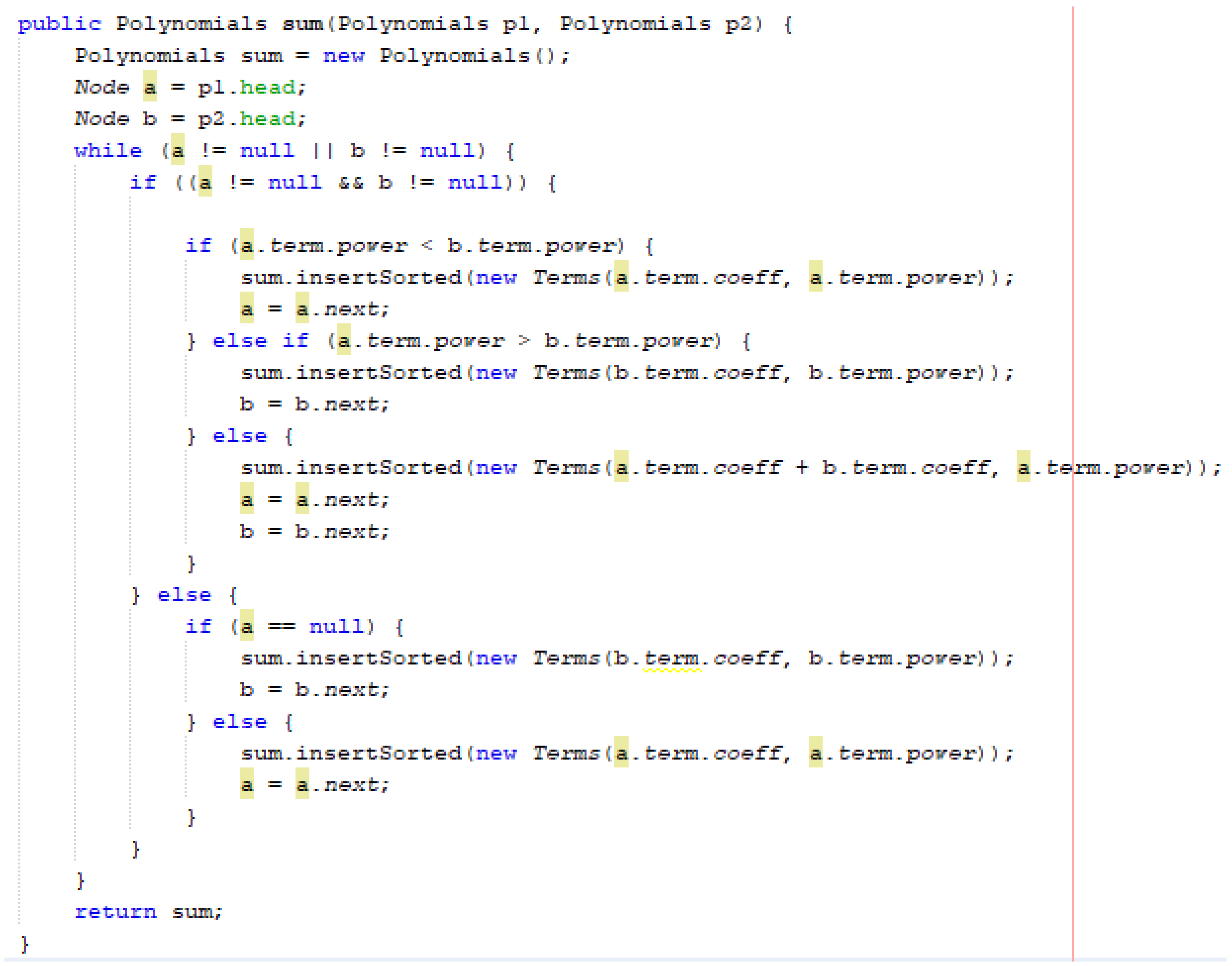
Ans) O(n)____



```
public float evaluate(float x) {
    float answer = 0;

    for (Node p1 = this.head; p1 != null; p1 = p1.next) {
        answer += p1.term.coeff * (Math.pow(x, p1.term.power));
    }
    return answer;
}
```

b) What is the runtime of the method *sum (p, q)* (in terms of *n)*?

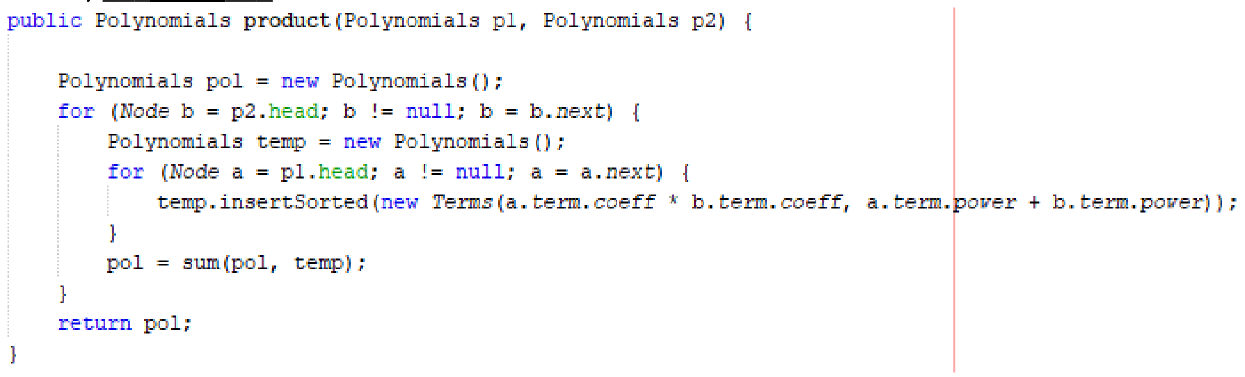
Ans) ___O(n)_____



```
public Polynomials sum(Polynomials p1, Polynomials p2) {
    Polynomials sum = new Polynomials();
    Node a = p1.head;
    Node b = p2.head;
    while (a != null || b != null) {
        if ((a != null && b != null)) {

            if (a.term.power < b.term.power) {
                sum.insertSorted(new Terms(a.term.coeff, a.term.power));
                a = a.next;
            } else if (a.term.power > b.term.power) {
                sum.insertSorted(new Terms(b.term.coeff, b.term.power));
                b = b.next;
            } else {
                sum.insertSorted(new Terms(a.term.coeff + b.term.coeff, a.term.power));
                a = a.next;
                b = b.next;
            }
        } else {
            if (a == null) {
                sum.insertSorted(new Terms(b.term.coeff, b.term.power));
                b = b.next;
            } else {
                sum.insertSorted(new Terms(a.term.coeff, a.term.power));
                a = a.next;
            }
        }
    }
    return sum;
}
```

c) What is the runtime of the method *product (p, q)* (in terms of *n*)?

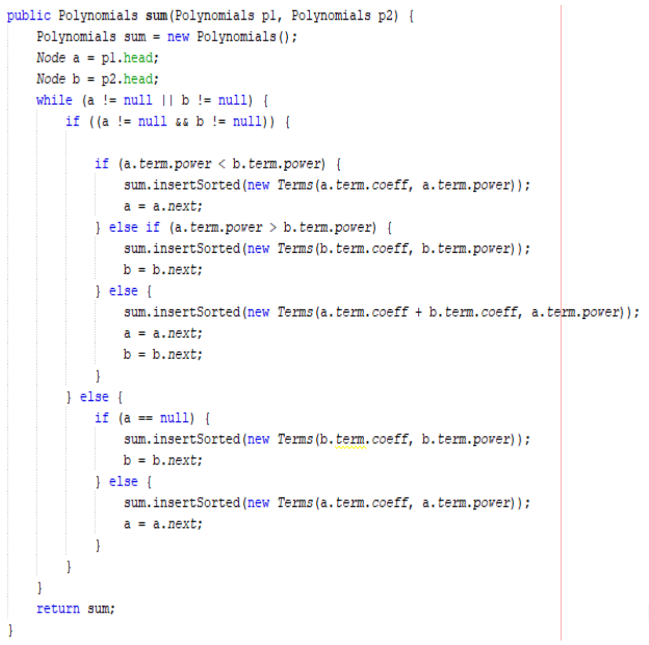
Ans) ___O(n^2)____



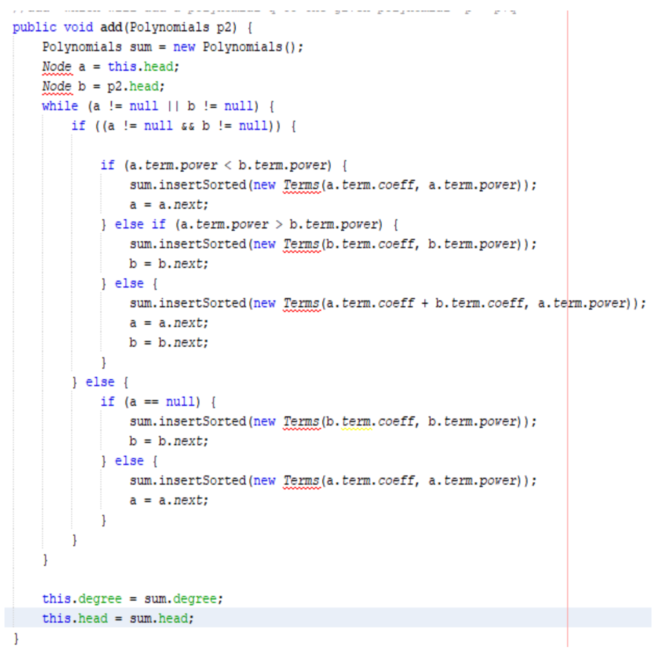
```
public Polynomials product(Polynomials p1, Polynomials p2) {

    Polynomials pol = new Polynomials();
    for (Node b = p2.head; b != null; b = b.next) {
        Polynomials temp = new Polynomials();
        for (Node a = p1.head; a != null; a = a.next) {
            temp.insertSorted(new Terms(a.term.coeff * b.term.coeff, a.term.power + b.term.power));
        }
        pol = sum(pol, temp);
    }
    return pol;
}
```

d) What is the difference between *sum* and *add*?

Ans) Sum () adds 2 polynomials and creates a new polynomial whereas Add () will add a polynomial q to the given polynomial p = p + q.



```
public Polynomials sum(Polynomials p1, Polynomials p2) {
    Polynomials sum = new Polynomials();
    Node a = p1.head;
    Node b = p2.head;
    while (a != null || b != null) {
        if ((a != null && b != null)) {

            if (a.term.power < b.term.power) {
                sum.insertSorted(new Terms(a.term.coeff, a.term.power));
                a = a.next;
            } else if (a.term.power > b.term.power) {
                sum.insertSorted(new Terms(b.term.coeff, b.term.power));
                b = b.next;
            } else {
                sum.insertSorted(new Terms(a.term.coeff + b.term.coeff, a.term.power));
                a = a.next;
                b = b.next;
            }
        } else {
            if (a == null) {
                sum.insertSorted(new Terms(b.term.coeff, b.term.power));
                b = b.next;
            } else {
                sum.insertSorted(new Terms(a.term.coeff, a.term.power));
                a = a.next;
            }
        }
    }
    return sum;
}
```

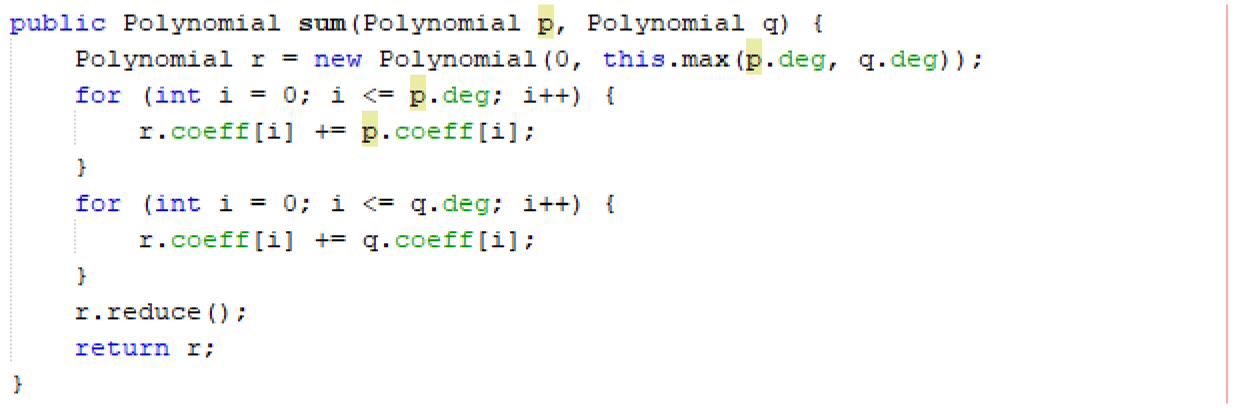


```
public void add(Polynomials p2) {
    Polynomials sum = new Polynomials();
    Node a = this.head;
    Node b = p2.head;
    while (a != null || b != null) {
        if ((a != null && b != null)) {

            if (a.term.power < b.term.power) {
                sum.insertSorted(new Terms(a.term.coeff, a.term.power));
                a = a.next;
            } else if (a.term.power > b.term.power) {
                sum.insertSorted(new Terms(b.term.coeff, b.term.power));
                b = b.next;
            } else {
                sum.insertSorted(new Terms(a.term.coeff + b.term.coeff, a.term.power));
                a = a.next;
                b = b.next;
            }
        } else {
            if (a == null) {
                sum.insertSorted(new Terms(b.term.coeff, b.term.power));
                b = b.next;
            } else {
                sum.insertSorted(new Terms(a.term.coeff, a.term.power));
                a = a.next;
            }
        }
    }

    this.degree = sum.degree;
    this.head = sum.head;
}
```

3) For the following questions, give an example (using *5th* degree polynomials) of:

a) When an array implementation is "better" for the method *sum* and why?

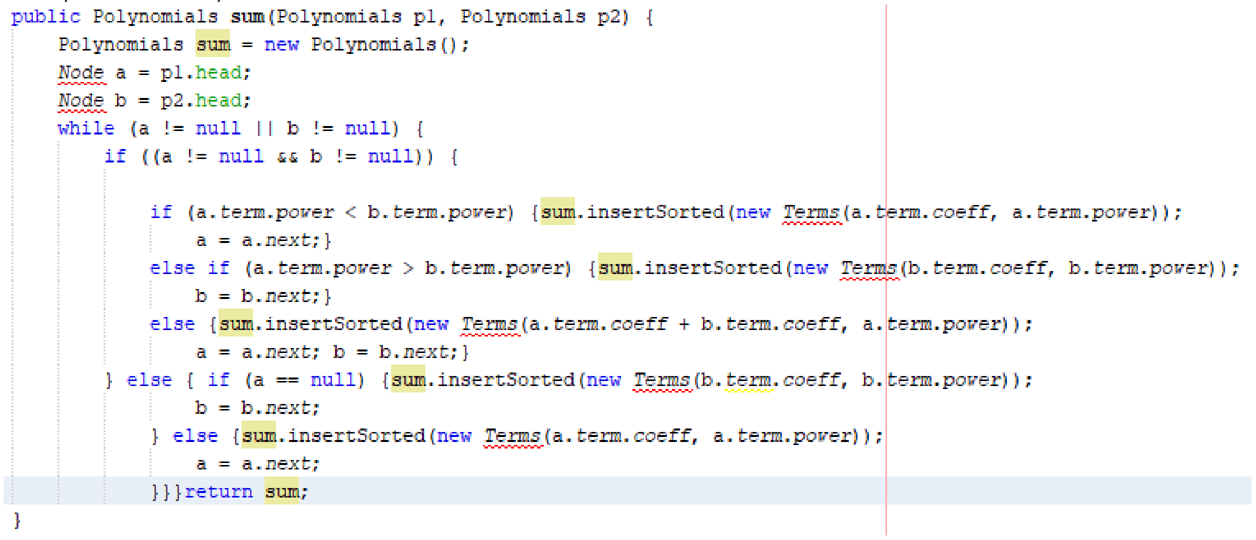
Ans) Array implementation will be better if the degree of the polynomial is predefined.



```
public Polynomial sum(Polynomial p, Polynomial q) {
    Polynomial r = new Polynomial(0, this.max(p.deg, q.deg));
    for (int i = 0; i <= p.deg; i++) {
        r.coeff[i] += p.coeff[i];
    }
    for (int i = 0; i <= q.deg; i++) {
        r.coeff[i] += q.coeff[i];
    }
    r.reduce();
    return r;
}
```

b) When a linked implementation is “better” for the method *sum* and why?

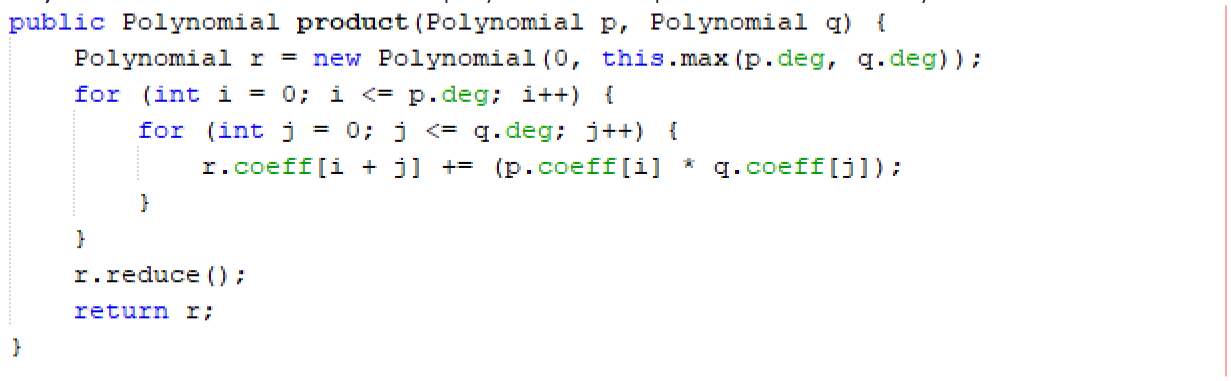
Ans) There is a size limit on arrays, which is a limitation. The number of elements in the array must be specified ahead of time. In contrast, the linked list allows you to add as many entries as you like.



```
public Polynomials sum(Polynomials p1, Polynomials p2) {
    Polynomials sum = new Polynomials();
    Node a = p1.head;
    Node b = p2.head;
    while (a != null || b != null) {
        if ((a != null && b != null)) {

            if (a.term.power < b.term.power) {sum.insertSorted(new Terms(a.term.coeff, a.term.power));
                a = a.next;}
            else if (a.term.power > b.term.power) {sum.insertSorted(new Terms(b.term.coeff, b.term.power));
                b = b.next;}
            else {sum.insertSorted(new Terms(a.term.coeff + b.term.coeff, a.term.power));
                a = a.next; b = b.next;}
        } else { if (a == null) {sum.insertSorted(new Terms(b.term.coeff, b.term.power));
                b = b.next;
            } else {sum.insertSorted(new Terms(a.term.coeff, a.term.power));
                a = a.next;
            }}}return sum;
}
```

c) When an array implementation is “better” for the method *product* and why?

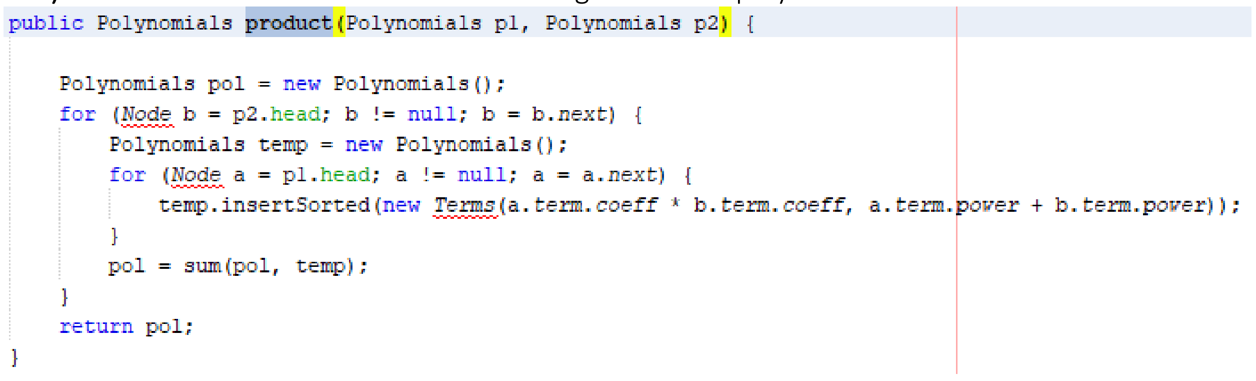
Ans) It is better when the terms in polynomials are predefined or already know.



```
public Polynomial product(Polynomial p, Polynomial q) {
    Polynomial r = new Polynomial(0, this.max(p.deg, q.deg));
    for (int i = 0; i <= p.deg; i++) {
        for (int j = 0; j <= q.deg; j++) {
            r.coeff[i + j] += (p.coeff[i] * q.coeff[j]);
        }
    }
    r.reduce();
    return r;
}
```

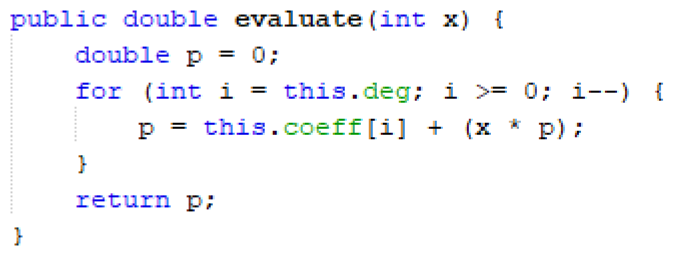
d) When a linked implementation is “better” for the method *product* and why?

Ans) It is better when we do not have the degrees of the polynomials.



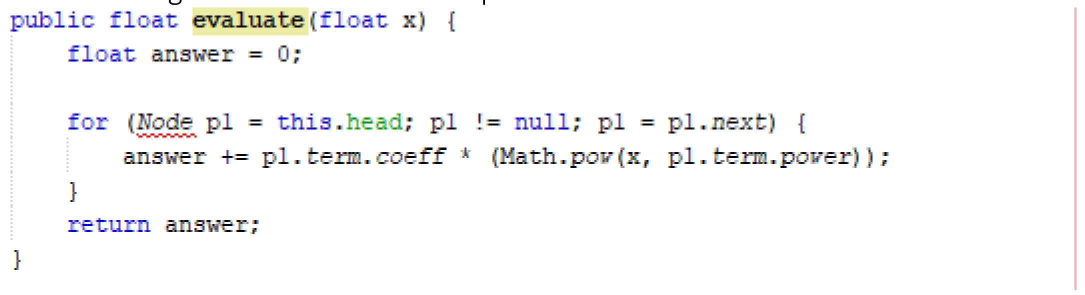
```
public Polynomials product(Polynomials p1, Polynomials p2) {

    Polynomials pol = new Polynomials();
    for (Node b = p2.head; b != null; b = b.next) {
        Polynomials temp = new Polynomials();
        for (Node a = p1.head; a != null; a = a.next) {
            temp.insertSorted(new Terms(a.term.coeff * b.term.coeff, a.term.power + b.term.power));
        }
        pol = sum(pol, temp);
    }
    return pol;
}
```

e) When an array implementation is “better” for the method *evaluate* and why?
Ans) Array implementation is better to use for evaluate method when we know the degree and coefficients beforehand.



```
public double evaluate(int x) {
    double p = 0;
    for (int i = this.deg; i >= 0; i--) {
        p = this.coeff[i] + (x * p);
    }
    return p;
}
```

f) When a linked implementation is “better” for the method *evaluate* and why?
Ans) In evaluate method of polynomials in linked implementation it will perform better when the degree and coefficient are predefined and the coefficient is not a fraction.



```
public float evaluate(float x) {
    float answer = 0;

    for (Node pl = this.head; pl != null; pl = pl.next) {
        answer += pl.term.coeff * (Math.pow(x, pl.term.power));
    }
    return answer;
}
```

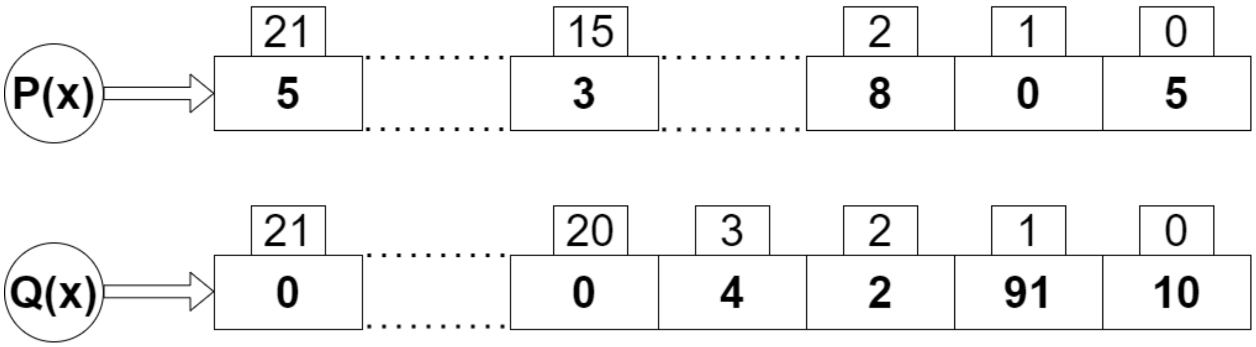
4) **Briefly discuss the differences in the space complexity (memory) of the array and linked implementations of the Polynomial class.**

Ans) When performing polynomial operations, arrays take up a lot of space. When solving polynomial equations, two parallel data structures are required. Both data structures' indexes advance to perform the desired operation.

$P(x)= 5x^{21}+3x^{15}+8x^2+5$

$Q(x)=4x^3+2x^2+91x+10$

Let's take an example of adding these two.



N values are used to store only four values, as you can see (N being the highest power). There will now be a single pointer that moves from index 21 to 0 and adds elements from both arrays. If we use polynomials with higher powers, there will be a significant amount of memory waste.

The index of any element in an array can be used to access it directly. In the case of a linked list, however, all previous elements must be traversed before any element can be reached. Additionally, better cache locality in arrays (due to contiguous memory allocation) can improve performance significantly. As a result, some operations (like modifying a specific element) are faster in arrays, while others (like inserting/deleting a data element) are faster in linked lists.