Data Structure HW2

Q1.

(A)

100
We could use an array to store, array’s index means power, and coefficient store in it.

(B)

3
We could use an array to store, array’s index means power, and coefficient store in it.

(C)

4
We could use an 2D array to store. array[0] store power, array[1] store coefficient.

Q2.

(1)

B^T	row	col	value
B^T[0]	9	5	8
B^T[1]	0	4	2
B^T[2]	0	8	2
B^T[3]	1	0	1
B^T[4]	1	7	1
B^T[5]	2	8	1
B^T[6]	3	0	1
B^T[7]	3	4	-1
B^T[8]	3	7	1

(2)

A X B	row	col	value
A X B[0]	7	5	12
A X B[1]	0	1	2
A X B[2]	0	3	2
A X B[3]	3	0	12
A X B[4]	3	2	2
A X B[5]	3	3	-4
A X B[6]	4	0	2
A X B[7]	4	3	-1
A X B[8]	5	0	6
A X B[9]	5	2	-1
A X B[10]	5	3	-4
A X B[11]	6	1	-1
A X B[12]	6	3	-1

Q3

(1) (Q3-1.CPP)

manual

If you want to use polynomial object, you need to declare one, and use `init_polynomial(polynomial , &a` to init it.

`void init_polynomial(polynomial &a)`

Init a polynomial a.

`int add_polynomial(polynomial &a, int degree, int coef)`

Add a term with polynomial a, degree and coefficient. If add success, return 1, otherwise, return 0.

`int coeff1(polynomial a, int p)`

Get the coefficient in corresponed degree. If it can't find or over the MAX_degree - 1, it will return 0.

result

```
takala@TakaladeMacBook-Pro HW2 % ./Q3-1
power 0 's cofficient = 10
power 5 's cofficient = 20
power 10 's cofficient = 30
power 15 's cofficient = 40
```

(2) (Q3-2.cpp)

manual

If you want polynomial, you need to declare a polynomial array and a variable. Then, use `void init_polynomial(polynomial a[], int &avail)` to init both.

`void init_polynomial(polynomial a[], int &avail)`
Init the polynomial and variable.

`int add_polynomial(polynomial a[], int &avail, int coef, int expon)`
Add a term with exponential and coefficient. If add success, return 1, otherwise, return 0.

`int coeff2(polynomial a[], int avail, int p)`
Get the coefficient in corresponed exponential. If it can't find or over the MAX_TERMS, it will return 0.

result

```
takala@TakaladeMacBook-Pro HW2 % ./Q3-2
power 0 's cofficient = 10
power 5 's cofficient = 20
power 10 's cofficient = 30
power 15 's cofficient = 40
```

(3)

(1) O(1)
We can get degree data direcly, so it's O(1).
(2) O(n)
We need to search all array which has current data, so it's O(n).

(4)

(1) O(m)
We need to check MAX_Degree times to add each degree, so it’s O(m).
(2) O(n)
We need to add all array which has ordered current data, but we don’t need to check the degree which not have data, so it’s O(n).