B Running Time of Naïve Polynomial Algorithm

The following code fragment computes each term of a polynomial of degree *n* from scratch

```
 1   y = 0;
 2   i = n;
 3   while (i >= 0) {
 4        z = 1;
 5        for (k = 0; k < i; k++) {
 6             z = z * x;
 7        }
 8        y = y + a[i] * z;
 9        i = i - 1;
10   }
```

Following the process described on pages 24 and 25 of the textbook, the table below shows the cost and number of times that each step in the code fragment takes:

Line	Statement	Cost	Times
1	`y = 0`	c_1	1
2	`i = n`	c_2	1
3	`while (i >= 0) {`	c_3	n + 2
4	`z = 1`	c_4	n + 1
5	`for (k = 0; k < i; k++) {`	c_5 c_6	n + 1 ((n * (n – 1) / 2) + 1
6	`z = z * x`	c_7	(n * (n – 1) / 2
7	`}`	0	---
8	`y = y + a[i] * z`	c_8	n + 1
9	`i = i – 1`	c_9	n + 1
10	`}`	0	---

Lines 1 and 2 are straightforward assignments.

Line 3, when combined with the decrement operation in line 9, provides a loop that runs from `n` down to 0. In addition, `i` is tested one more time than the loop actually runs.

Line 4 runs as many times as the loop itself, as described for line 3.

Line 5 is a complicated construct. It is an inner loop that will run `n + 1` times, just like line 4. Each time it runs, `k` will be set to 0, which is a constant time operation. The first time the loop runs, it will run `n` times. The second time it runs, it will run `n – 1` times. This will continue until `i = 1`. This is merely an arithmetic series, the solution of which is `(n * (n – 1)) /2`. In addition, the loop variable k will be tested one more time than the loop itself.

Line 6 will run as many times as the inner loop itself, as described for line 5.

Line 7 is merely an inner loop semantic that transfers control back to line 5.

Lines 8 and 9 run the same number of times as the outer loop itself, as described for line 3.

Line 10 is merely an outer loop semantic that transfers control back to line 3.

Adding the cost of all statements gives the following equation:

$$T(n) = c_1(1) + c_2(1) + c_3(n+2) + c_4(n+1) + c_5(n+1) + c_6\left(\frac{n(n-1)}{2} + 1\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n+1) + c_9(n+1)$$

$$T(n) = (c_1 + c_2 + 2c_3 + c_4 + c_5 + c_6 + c_8 + c_9) + \left(c_3 + c_4 + c_5 + \frac{c_6}{2} + \frac{c_7}{2} + c_8 + c_9\right)n + \left(\frac{c_6}{2} + \frac{c_7}{2}\right)n^2$$

Combine constants:

let: $a = \frac{c_6}{2} + \frac{c_7}{2}$

and: $b = c_3 + c_4 + c_5 + \frac{c_6}{2} + \frac{c_7}{2} + c_8 + c_9$

and: $c = c_1 + c_2 + 2c_3 + c_4 + c_5 + c_6 + c_8 + c_9$

Then:

$$T(n) = an^2 + bn + c$$

This running time is directly proportional to n^2. In addition, it can be bounded on both the upper and lower sides by proper selection of arbitrary constants. Therefore, the asymptotic running time of this code fragment for the naïve polynomial calculation is:

$$T(n) = \Theta(n^2)$$