1. a. $i = n$ $\frac{n}{2^0}$

$i = \frac{n}{2}$ $\frac{n}{2^1}$

$i = \frac{n}{4}$ $\frac{n}{2^2}$

$\vdots$

$i = \frac{n}{2^{k-1}}$

$\vdots$

$i = 1$ → end

operations	cost
Variable clafication	2
assignment	$2 + \log_2 n + 1$
comparision	$\log n + 1$
incrementation	$\log n + 1$

$2^{k-1} = n \Rightarrow k = \log_2 n + 1$

$\Rightarrow T(n) = 2 + 2 + \log_2 n + \log_2 n + \log_2 n + 3$

$= 3\log n + 7$

$T(n) = 3\log n + 7 \leq 3\log n + 7\log n = 10\log n$

$n_0 = 2,\ c = 10,\ g(n) = \log n$

$\Rightarrow T(n) = O(\log n)$

$T(n) = 3\log n + 7 \geq 3\log n$

$\Rightarrow T(n) = \Omega(\log n)$

$\Rightarrow T(n) = \Theta(\log n)$

b. outer loop: $j = n, \frac{n}{2}, \frac{n}{4} \ldots \frac{n}{2^{k-1}} \ldots 1$ → $\log n + 1$ times

inner loop: $i = 0, 1, 2 \ldots n-1$ → n times

operations	cost
Variable clafication	$2 + \log n + 1$

assignment	$2 + \log n + 1$
comparision	$\log n + 1 + n(\log n + 1)$
incrementation	$2n(\log n + 1)$

$\Rightarrow T(n) = 3\log n + 7 + 3n(\log n + 1)$

$= 3n\log n + 3n + 3\log n + 7 \leq 3n\log n + 3n\log n + 3n\log n + 7n\log n$

$\leq 16 n \log n$

$\Rightarrow T(n) = O(n\log n)$

$T(n) \geq 3n\log n \Rightarrow T(n) = \Omega(n\log n)$

$\Rightarrow T(n) = \Theta(n\log n)$

C. outter loop: $n = n, \frac{n}{2}, \frac{n}{4}, \cdots \frac{n}{2^{k-1}} \cdots 1 \rightarrow \log n + 1$ times

inner loop: $i = 0, 1, 2, \cdots n-1 \rightarrow n$ times

operations	cost
Variable clafication	$\log n + 1$
assignment	$2(\log n + 1)$
comparision	$\log n + 1 + n(\log n + 1)$
incrementation	$n(\log n + 1)$
function	$\frac{n}{2} \times n(\log n + 1)$

$\Rightarrow T(n) = 4\log n + 4 + \frac{1}{2}n^2(\log n + 1) + 2n(\log n + 1)$

$= \frac{1}{2}n^2\log n + \frac{1}{2}n^2 + 2n\log n + 4\log n + 2n + 4$

$$\leq \frac{1}{2}n^2\log n + \frac{1}{2}n^2\log n + 2n\log n + 4n$$

$$= 13n^2\log n$$

$$\Rightarrow T(n) = O(n^2\log n)$$

$$T(n) \geq \frac{1}{2}n^2\log n \Rightarrow T(n) = \Omega(n^2\log n)$$

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

d. let C_1 = the runtime of anotherMethod(). then

$$T(n) = 2T(\tfrac{n}{2}) + C_1$$

$$= 2(2T(\tfrac{n}{4}) + C_1) + C_1 = 4T(\tfrac{n}{4}) + 3C_1$$

$$= 2(2(2(T(\tfrac{n}{8}) + C_1)) + C_1) + C_1 = 8T(\tfrac{n}{8}) + 7C_1$$

$$\vdots$$

$$= 2^k T(\tfrac{n}{2^k}) + (2^k - 1)C_1$$

when $\frac{n}{2^k} = 1$, the loop will end, then

$$k = \log n$$

$$\Rightarrow T(n) = 2^{\log n} T(1) + (2^{\log n} - 1)C_1$$

$$= nT(1) + (n-1)C_1 \qquad T(1) = 1$$

$$\Rightarrow T(n) = (C_1 + 1)n - C_1$$

$$\Rightarrow T(n) = O(n) = \Omega(n) = \Theta(n)$$

2. a. growth rath of $O(n^2)$

① special case : if one of the lists is empty list,
then return (No intersections)

② use two while loop to find the intersections :
if there is n elements in list A and m elements

```
while ( loopA : traverse listA) {
    while ( loopB : traverse list B)
        if ( A == B ) {
            return;
        }
}
```

$\Rightarrow T(n) = n \cdot m \leq n^2 = O(n^2)$

b. Since we assume the two linked lists don't have duplicates, then we can use a Treeset to hold the elements of one of the lists, then see whether Set A contains any element from list B

① Create an empty tree set :

```
Set < ListNode > set1 = new TreeSet < > ( );
```

② add the reference of each node in list A to set1

```
while ( ~ ) {
    set1.add ( ~ );   // O(log n)
```

}

the add elements to TreeSet costs O(logn)

then the while loop cost O(nlogn).

③ compare the elements from set1 with listB,

-if set1.contains(listB.current), then return.

this part cost O(n) time.

⇒ the whole process costs O(nlogn)

C. If we use HashSet rather than TreeSet data structure, the growth rate will drop down to O(n) since HashSet.add(n) opperation only cost O(1), better than the previous one.

```
while (loopA : listA) {
        set1.add(listA.current); // O(1)
}
}
```

⇒ T(n) = O(n)

The steps are almost the same as previous one

3. a. let f(n) represent method1, and g(n) represent method2

then $f(4) = g(3,3) - (4-1) = g(3,3) - 3$

$= 1 + g(2,3) - 3 = 1 + 1 + g(1,3) - 3$

$= 1 + 1 + 1 + g(0,3) - 3 = g(0,3)$

$g(0,3) = f(3) = g(2,2) - (3-1) = g(2,2) - 2$

$= 1 + g(1,2) - 2 = 1 + 1 + g(0,2) - 2 = g(0,2)$

$g(0,2) = f(2) = g(1,1) - (2-1) = g(1,1) - 1$

$= 1 + g(0,1) - 1 = g(0,1)$

$g(0,1) = f(1) = g(0,0) - (1-1) = g(0,0)$

$g(0,0) = f(0) = 0$

$\Rightarrow f(4) = g(0,3) = f(3) = g(0,2) = f(2) = g(0,1)$

$= f(1) = g(0,0) = f(0) = 0$

$\Rightarrow$ The output will be 0

b. let $T_1(n)$ represent the runing time function of $f(n)$

let $T_2(m,k)$ represent the runing time function of $g(n)$

$f(n) = g(n-1, n-1) - (n-1)$ let C_1 = time to caculate $n-1$

$\Rightarrow T_1(n) = T_2(m,k) + C_1$ $\quad m = n-1,\ k = n-1$

$\Rightarrow T_1(n) = T_2(n-1, n-1) + C_1$

$T_2(n-1, n-1) = T_2(n-2, n-1) + C_2 = T_2(n-3, n-1) + 2C_2$

$= T_2(n-4, n-1) + 3C_2 \ldots = T_2(n-k, n-1) + (k-1)C_2$

When $n-k=0$, it ends $\Rightarrow \quad k=n$

$\Rightarrow T_2(n-1, n-1) = T_2(0, n-1) + (n-1)\, C_2$

$= T_1(n-1) + (n-1)\, C_2$

$\Rightarrow T_1(n) = T_1(n-1) + (n-1)\, C_2 + C_1$

$= T_1(n-2) + (n-2)\, C_2 + (n-1)\, C_2 + C_1$

$= T_1(n-3) + (n-3)\, C_2 + (n-2)\, C_2 + (n-1)\, C_2 + C_1$

$= T(0) + \sum_{i=0}^{n-1} i\, C_2 + C_1$

$= \frac{n(n-1)}{2} C_2 + C_1 = \frac{1}{2} C_2 n^2 - \frac{1}{2} n C_2 + C_1$

if $C_1 = C_2 = 1$, then

$T_1(n) = \frac{1}{2} n^2 - \frac{1}{2} n + 1 = O(n^2)$

From above, we get $T_2(n-1, n-1) = T_1(n-1) + (n-1) C_2$

use k, m replace $n-1$, then

$T_2(m, k) = T_1(k) + k C_2$

$= \frac{1}{2} k^2 - \frac{1}{2} k + 1 + k C_2$

$= \frac{1}{2} k^2 + (C_2 - \frac{1}{2}) k + 1$

$\Rightarrow T_2(m, k) = O(n^2)$

C. The growth rate is n^2.