1. Analyze: n is data.length

if (manyItems == data.length)

operation: 2

int biggerArray[ ];

operation: 1

biggerArray = new int[manyItems\*2 + 1];

operation: 4

for(int i=0;i < manyItems;i++) //n iterations, each iteration has <=7 operations

operation: 1+n+2n

biggerArray [i] = data[i];

operation: 3n

data = biggerArray;

operation: 1

data[manyItems] = element;

operation: 2

manyItems++;

operation: 2

**total number of operations <= 6n+13 >> time complexity: O(n)**

1. Analyze: n is manyItems(The actual number of elements in the int array bag)

int answer = 0

operation: 1

int index;

operation: 1

answer = 0;

operation: 1

for (index = 0; index < manyItems; index++)//n iterations, each iteration has <=8 operations

operation: 1+n+2n

if (target == data[index])

operation: 2n

answer++;

operation: 2n

return answer;

operation: 1

**total number of operations <= 7n+5 >> time complexity: O(n)**

1. Analyze: n is the given parameter, position

IntNode cursor;

operation: 1

int i;

operation: 1

if (position <= 0)

throw new IllegalArgumentException("position is not positive");

operation: 2

cursor = head;

operation: 1

for(i=1;(i < position) && (cursor != null); i++)//n-1 iterations, each iteration has <=8 operations

operation: 1+3(n-1)+2(n-1)

cursor = cursor.link;

operation: 2n

return cursor;

operation: 1

**total number of operations <= 7n+2 >> time complexity: O(n)**

1. Analyze: n is the actual number of nodes in the linked list starting from the given head

IntNode cursor = null;

operation: 1

int answer = 0;

operation: 1

for (cursor = head; cursor != null; cursor = cursor.link)

//n iterations, each iteration has <=6 operations

operation: 1+n+2n

answer++;

operation: 2n

return answer;

operation: 1

**total number of operations <= 5n+4 >> time complexity: O(n)**