```
Algorithm insertionSort (A,n)
In: Array A storing n values
Out: {Sort A in increasing order}
for i = 1 to n-1 do {
    temp = A[i]
    j = i – 1
    while (j >= 0) and (A[j] > temp) do {
        A[j+1] = A[j]
        j = j – 1
    }
    A[j+1] = temp
}
```

Iteration	Operations
$i=1$	$c_2 + c_1 \times 1$
$i=2$	$c_2 + 2c_1$
$i=3$	$c_2 + 3c_1$
$\vdots$	$\vdots$
$i=n-1$	$c_2 + (n-1)c_1$

$+ + + \cdots +$

c_2

c_1

$j=i-1$	$+c_1$
$j=i-2$	$+c_1$
$j=i-3$	$+c_1$
$\vdots$	$+\vdots$
$j=0$	$+c_1$
$j=-1$	

ic_1

$$f(n) = (n-1)c_2 + c_1 \sum_{i=1}^{n-1} i = (n-1)c_2 + \frac{n(n-1)}{2}c_1$$

is $O(n^2)$

Algorithm selectionSort (A,n)

In: Array A storing n values

Out: {Sort A in increasing order}

```
Q = new queue
for i = n-1 downto 0 do {
    smallest = 0                                        c4
    for j = 1 to i do                                   i c1
        if A[j] < A[smallest] then smallest = j         c1
    Q.enqueue(A[smallest])                              c3
    for j = smallest+1 to i do                          i c2
        A[j-1] = A[j]                                   c2
}
for i = 0 to n-1 do
    A[i] = Q.dequeue                                    c4      c4 n
```

Worst case analysis

Loop body cost: $\underbrace{c_3 + c_4}_{c'} + \underbrace{(c_1 + c_2)}_{c''} i = c' + c'' i$

	#ops
$i = n-1$	$c' + c''(n-1)$ +
$i = n-2$	$c' + c''(n-2)$ +
$i = n-3$	$c' + c''(n-3)$ +
$\vdots$	$\vdots$
$i = 1$	$c' + c''(1)$ +
$i = 0$	$c' + c''(0)$

$$c' n + c'' \sum_{i=0}^{n-1} i$$

$$\sum_{i=0}^{n-1} i = 0 + 1 + 2 + \cdots + n-1 = 1 + 2 + \cdots + n-1 = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

$$f(n) = c' n + c'' \frac{n(n-1)}{2} + c_4 n$$

is $O(n^2)$

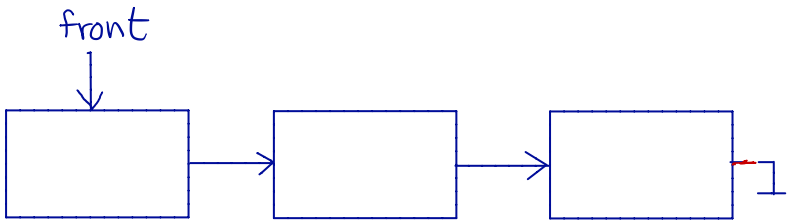
The above analysis uses this equality

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

```
//-----------------------------------------------------------
//  Adds the specified element to the rear of the queue.
//-----------------------------------------------------------
public void enqueue (T element) {
   LinearNode<T> newNode = new LinearNode<T> (element);

   if (isEmpty( )) {
        front = newNode;
        rear = newNode;
   }
   else {
        rear.setNext (newNode);
        rear = newNode;
   }
   count++;
}
```

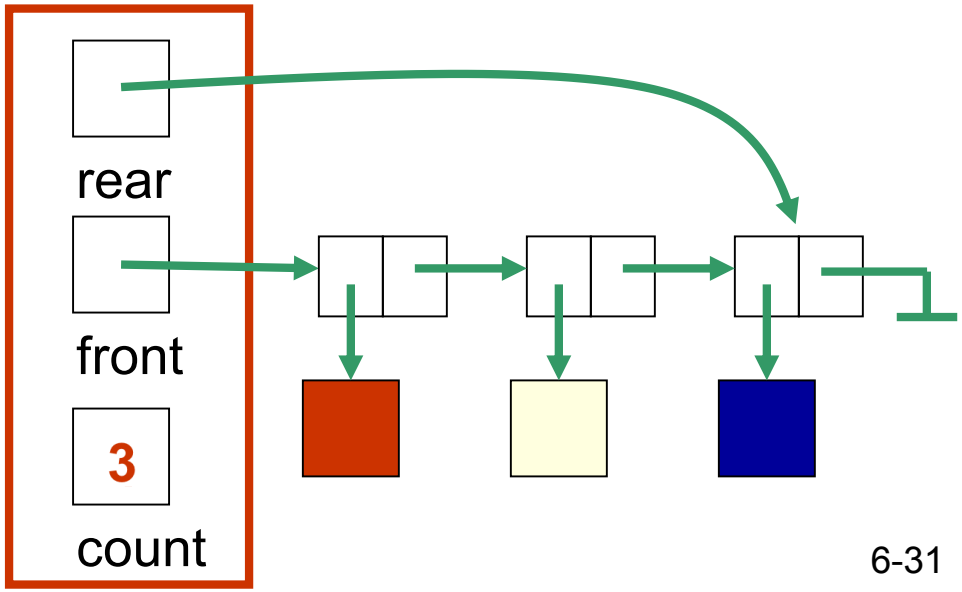
C3



count = 3

```
//-----------------------------------------------------------
//  Removes the element at the front of the queue and returns a
//  reference to it. Throws an EmptyCollectionException if the
//  queue is empty.
//-----------------------------------------------------------
public T dequeue ( ) throws EmptyCollectionException {
    if (isEmpty( ))
        throw new EmptyCollectionException ("queue");
    T result = front.getElement( );
    front = front.getNext( );
    count--;
    if (count == 0) rear = null;
    return result;
}
```

C4



```
Algorithm selectionSort (A,n)
In: Array A storing n values
Out: {Sort A in increasing order}
for i = 0 to n-2 do {
    smallest = i
    for j = i + 1 to n - 1do {
        if A[j] < A[smallest] then
            smallest = j
    }
    temp = A[smallest]
    A[smallest] = A[i]
    A[i] = temp
}
```

C_2

C_1

$C_1(n-i-1)$

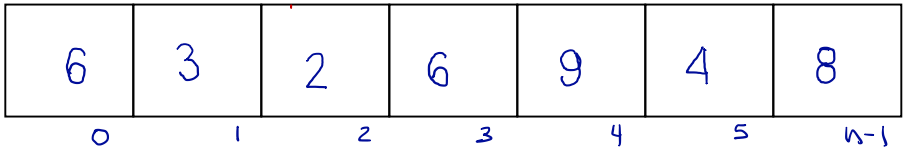
$C_2 + C_1(n-i-1)$

#OPS

$i=0$	$C_2 + C_1(n-1)$ +
$i=1$	$C_2 + C_1(n-2)$ +
$i=2$	$C_2 + C_1(n-3)$ +
$i=3$	$C_2 + C_1(n-4)$ +
⋮	⋮
$i=n-3$	$C_2 + C_1(2)$ +
$i=n-2$	$C_2 + C_1(1)$

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

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



pivot = 6

Algorithm quicksort(A,n)

In: Array A storing n values

Out: {Sort A in increasing order}

```
if n > 1 then {
    pivot = A[0]
    smaller, equal, larger = new arrays of length n
    n_s = n_e = n_l = 0
    for i = 0 to n-1 do
        if A[i] < pivot then {
            smaller[n_s] = A[i]
            n_s = n_s + 1
        }
        else if A[i] = pivot then {
            equal[n_e] = A[i]
            n_e = n_e + 1
        }
        else {
            larger[n_l] = A[i]
            n_l = n_l + 1
        }
    quicksort(smaller, n_s)
    quicksort(larger, n_l)
    i = 0
    for j = 0 to n_s do {
        A[i] = smaller[j]
        i = i + 1
    }
    for j = 0 to n_e do {
        A[i] = equal[j]
        i = i + 1
    }
    for j = 0 to n_l do {
        A[i] = larger[j]
        i = i + 1
    }
}
```

Worst case

$O(n^2)$

Average case

$O(n \log n)$