Attendance Quiz on April 25, 18

	To sort an array *A*[0..*N*-1]:	**Primitive Operations**
(1)	for (int last = *N* -1; last >= 1; last --) {	**(1) 2N**
	// Move the largest entry in *A*[0…last] to *A*[last]	
(2)	for (int index = 0; index <= last-1; index++) {	**(2) (2N) (3(N-1))**
	//swap adjacent elements if necessary	
(3)	if (A[index] > A[index+1]) {	**(3) 3 [(2N) (3(N-1))]**
(4)	int temp = A[index];	**(4) 2[(2N) (3(N-1))]**
(5)	A[index] = A[index+1];	**(5) 3[(2N) (3(N-1))]**
(6)	A[index + 1] = temp;	**(6) 2[(2N) (3(N-1))]**
(7)	} // index++	**(7) 2(2N) (N-1)**
(8)	} // last--	**(8) 2(2N)**
	}	

(1) 2N

1 Fetch Last + 1 Comparison = 2 Steps

Do these 2 steps for N times

(2) **(2N) (3(N-1))**

3 Ops = 1 fetch index + 1 fetch last + 1 comparison, do this for N-1 times

Do this 2N times (Nested loop)

(3) **3 [(2N) (3(N-1))]**

3 Ops = 2 Fetches A[] + 1 Compare, do this for **(2N) (3(N-1))**

(4) **2[(2N) (3(N-1))]**
2 Ops = Fetch A[] + 1 assignment, do this for **[(2N) (3(N-1))]**
(5) **3[(2N) (3(N-1))]**
2 Ops = Fetch A[] + 1 assignment, do this for **[(2N) (3(N-1))]**
(6) **2[(2N) (3(N-1))]**
2 Ops = Fetch A[] + 1 assignment, do this for **[(2N) (3(N-1))]**
(7) **2(2N) (N-1)**
2 Ops = Fetch index + 1 assignment, do this for **[(2N) (N-1)]**
(8) **2(2N)**
2 Ops = Fetch last + 1 assignment, do this for **(2N)**

Total Primitive Operations:

2N + (2N) (3(N-1)) + 2(3 [(2N) (3(N-1))]) + 4[(2N) (3(N-1))] + 2(2N) (N-1) + 2(2N) =
70 N^2 - 64 N
(Computed from Wolframalpha)

Asymptotic Algorithm Analysis : O(N^2)