Find the number of statements executed by each algorithm in the worst case. Our rules for counting statements are on the following page.

1. Linear search: search an unsorted array for a key. If it’s in the list, return its subscript. If it’s not in the list, return -1. Also characterize the input that gives the worst case performance.

```c
// A:   the list (input)
// n:   the number of elements in the list (input)
// key: the value being searched for (input)
int Lin_search(int A[], int n, int key) {
    int i;
    for (i = 0; i < n; i++)
        if (A[i] == key) return i;
    return -1;
} /* Lin_search */
```

2. Vector dot product: find the dot-product of two n-element arrays: find the product of corresponding elements and return their sum.

```c
// x:    the first n-element input array (input)
// y:    the second n-element input array (input)
// n:    the number of elements in x and y.
int Dot_product(int x[], int y[], int n) {
    int i, dot;
    dot = 0;
    for (i = 0; i < n; i++)
        dot = dot + x[i]*y[i]; // add, mult and assign = 3 statements
    return dot;
} /* Dot_prod */
```
Some Rules for Counting Statements

1. Declaring a variable requires no statements.

2. Assigning to a variable requires one statement.

3. A_for statement usually requires
   (a) One initialization
   (b) $k$ increments
   (c) $k$ tests
   (d) One branch

4. An if statement requires
   (a) At least one statement for the test.
   (b) (Technically, we should allow a statement for a branch, but we’ll include this with the test.)