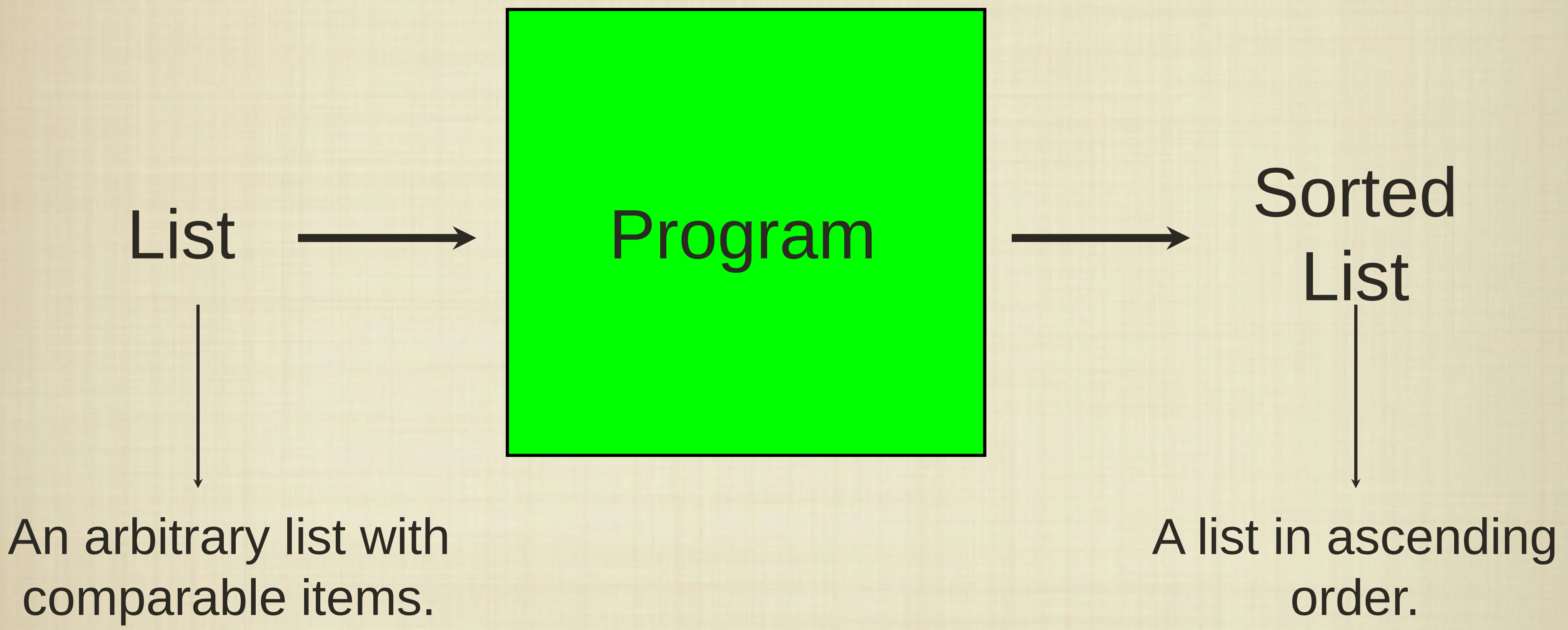
Algorithm Analysis Sorting

Fall 2013

Carola Wenk

Sorting

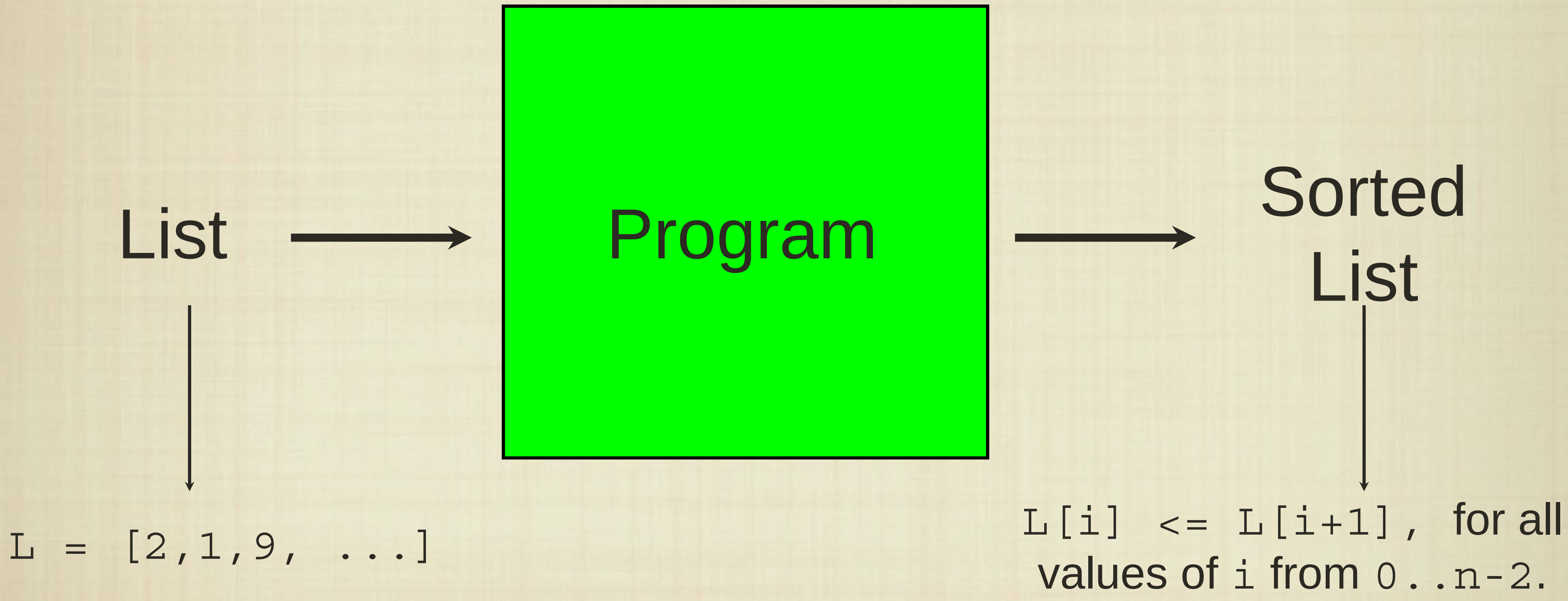
Let’s consider the problem of sorting a list of numbers.


List
Program
Sorted List
An arbitrary list with comparable items.
A list in ascending order.

Can we make the specifications more concrete?

Sorting

Let’s consider the problem of sorting a list of numbers.



How do we sort a list?

A Sorting Algorithm

Let’s consider the problem of sorting a list of numbers.

Algorithm:

1. Find the minimum element in the list.

2. Swap it with the first element.

3. Repeat with the rest of the list.

What is the running time?

A Sorting Algorithm

Let’s consider the problem of sorting a list of numbers.

Algorithm:

1. Find the minimum element in the list.

2. Swap it with the first element.

3. Repeat with the rest of the list.

What is the running time? How many times do we find the minimum?

Algorithm Analysis

This approach to sorting a list is often called “selection” sorting. For a list with i elements, we perform about $c \cdot i$ operations to find the minimum.

Each time we find a minimum, we are reducing the time spent on searching for “future” minima. The list sizes are:

$$n, n-1, n-2, \ldots, 2, 1$$

The corresponding number of operations is:

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

Algorithm Analysis

This approach to sorting a list is often called “selection” sorting. For a list with i elements, we perform about $c \cdot i$ operations to find the minimum.

Each time we find a minimum, we are reducing the time spent on searching for “future” minima. The list sizes are:

$$n, n-1, n-2, \ldots, 2, 1$$

The corresponding number of operations is:

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

Algorithm Analysis

This approach to sorting a list is often called “selection” sorting. For a list with i elements, we perform about $c \cdot i$ operations to find the minimum.

Each time we find a minimum, we are reducing the time spent on searching for “future” minima. The list sizes are:

$$n, n-1, n-2, \ldots, 2, 1$$

The corresponding number of operations is:

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

An Implementation

```
# find the index of the minimum in L
def my_min_index(L):
  curr_min_index = 0
  for i in range(1,len(L)):
    if (L[i] < L[curr_min_index]):
      curr_min_index = i
  return curr_min_index

# swap the contents of L[i] and L[j]
def swap(L, i, j):
  temp = L[i]; L[i] = L[j]; L[j] = temp

# sort a list in O(n^2) time
def selection_sort(L):
```

[redacted]

An Implementation

```
# find the index of the minimum in L
def my_min_index(L):
  curr_min_index = 0
  for i in range(1,len(L)):
    if (L[i] < L[curr_min_index]):
      curr_min_index = i
  return curr_min_index

# swap the contents of L[i] and L[j]
def swap(L, i, j):
  temp = L[i]; L[i] = L[j]; L[j] = temp

# sort a list in O(n^2) time
def selection_sort(L):
  n = len(L)
  for i in range(n):

    [redacted]
```

An Implementation

```
# find the index of the minimum in L
def my_min_index(L):
  curr_min_index = 0
  for i in range(1,len(L)):
    if (L[i] < L[curr_min_index]):
      curr_min_index = i
  return curr_min_index

# swap the contents of L[i] and L[j]
def swap(L, i, j):
  temp = L[i]; L[i] = L[j]; L[j] = temp

# sort a list in O(n^2) time
def selection_sort(L):
  n = len(L)
  for i in range(n):
    j = i + my_min_index(L[i:])
    swap(L, i, j)
```

$\longleftarrow$ “List Slicing”