Algorithm:
Insertion Sort

Insertion Sort

Key Aspects:

- In-place comparison sort.
- The algorithm divides the list into two portions: sorted and unsorted.
- Selects the minimum element and “inserts it” at the right position in the sorted portion of the list.
- Inefficient for large lists.
- Efficient for small or mostly sorted lists.

The list is sorted in ascending order

Algorithm:

- The sorted portion of the list initially contains one element (at index 0).
- Select the first element of the unsorted portion of the list.
- Insert this element at the right position in the sorted portion of the list.
- To do this, you might need to “move” to the right all the elements that are greater than the element that is being inserted.
- Expand the sorted portion.
- Repeat these steps until the process is completed. The final output is a sorted list.

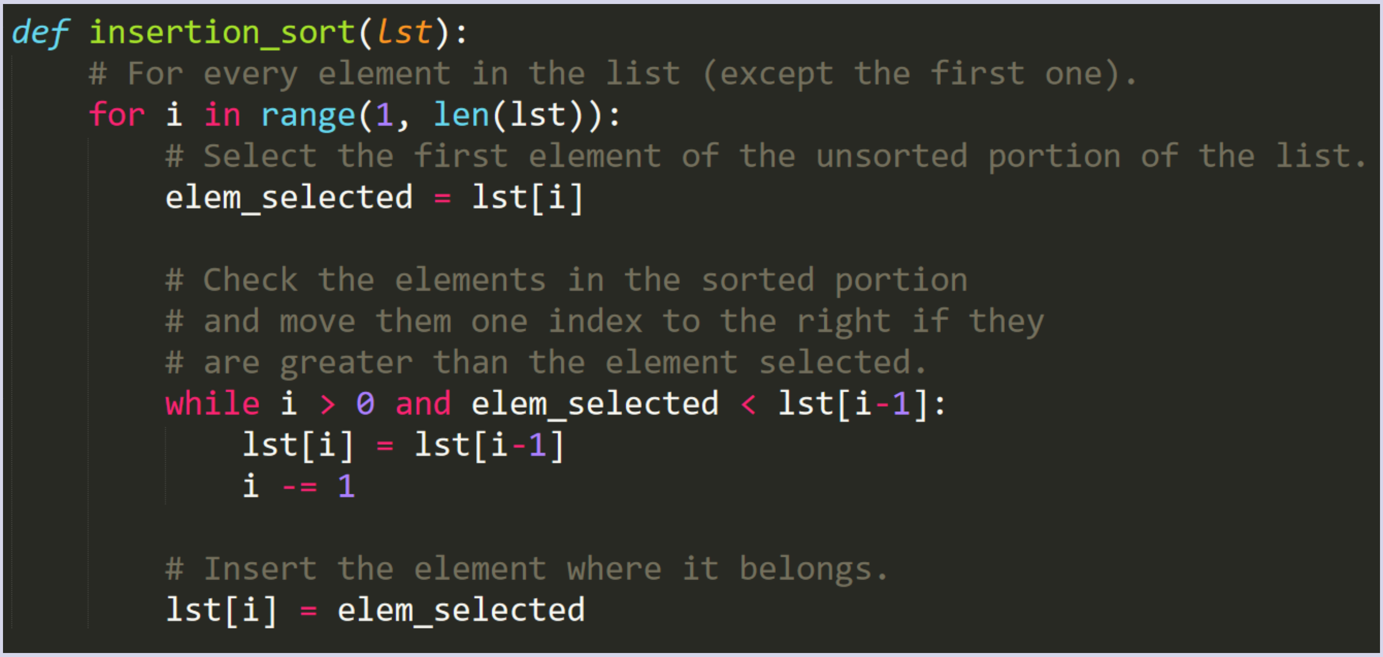
Time Complexity:

- Worst-Case Time Complexity: Quadratic O(n^2)
- Average-Case Time Complexity: Quadratic O(n^2)
- Best-Case Time Complexity: Linear O(n)

Insertion Sort
=
Insert the
elements one by one

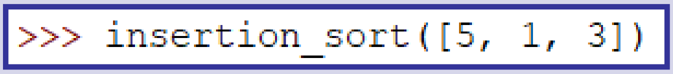
Insertion Sort

Code:

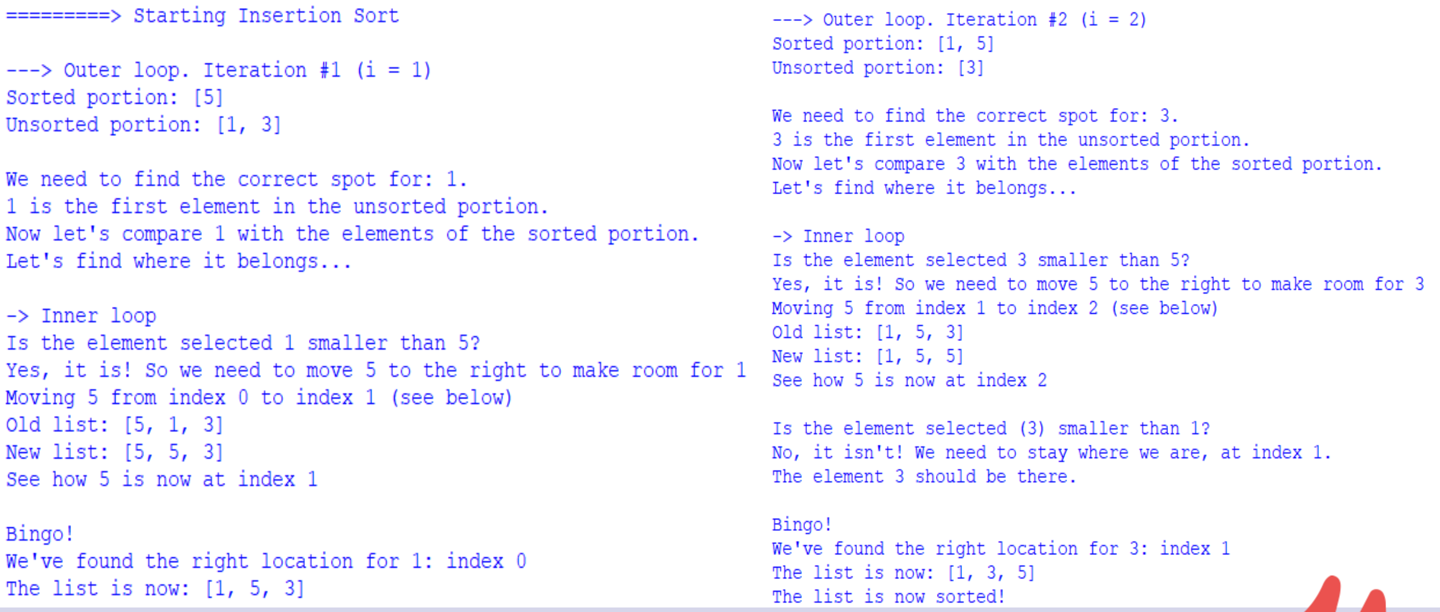


```python
def insertion_sort(lst):
    # For every element in the list (except the first one).
    for i in range(1, len(lst)):
        # Select the first element of the unsorted portion of the list.
        elem_selected = lst[i]

        # Check the elements in the sorted portion
        # and move them one index to the right if they
        # are greater than the element selected.
        while i > 0 and elem_selected < lst[i-1]:
            lst[i] = lst[i-1]
            i -= 1

        # Insert the element where it belongs.
        lst[i] = elem_selected
```

Example:



```
>>> insertion_sort([5, 1, 3])
```



```
=========> Starting Insertion Sort

---> Outer loop. Iteration #1 (i = 1)
Sorted portion: [5]
Unsorted portion: [1, 3]

We need to find the correct spot for: 1.
1 is the first element in the unsorted portion.
Now let's compare 1 with the elements of the sorted portion.
Let's find where it belongs...

-> Inner loop
Is the element selected 1 smaller than 5?
Yes, it is! So we need to move 5 to the right to make room for 1
Moving 5 from index 0 to index 1 (see below)
Old list: [5, 1, 3]
New list: [5, 5, 3]
See how 5 is now at index 1

Bingo!
We've found the right location for 1: index 0
The list is now: [1, 5, 3]
```

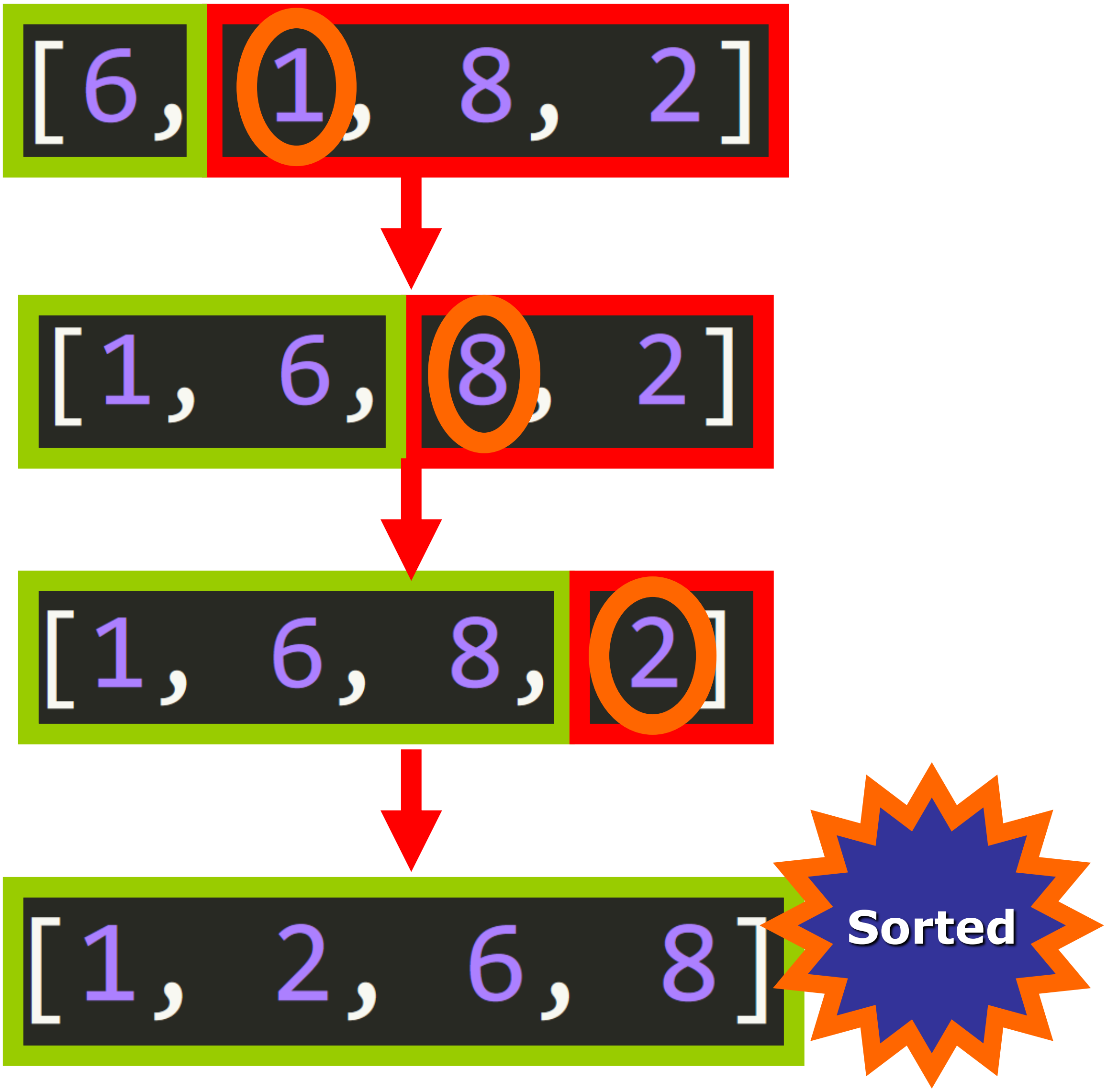
```
---> Outer loop. Iteration #2 (i = 2)
Sorted portion: [1, 5]
Unsorted portion: [3]

We need to find the correct spot for: 3.
3 is the first element in the unsorted portion.
Now let's compare 3 with the elements of the sorted portion.
Let's find where it belongs...

-> Inner loop
Is the element selected 3 smaller than 5?
Yes, it is! So we need to move 5 to the right to make room for 3
Moving 5 from index 1 to index 2 (see below)
Old list: [1, 5, 3]
New list: [1, 5, 5]
See how 5 is now at index 2

Is the element selected (3) smaller than 1?
No, it isn't! We need to stay where we are, at index 1.
The element 3 should be there.

Bingo!
We've found the right location for 3: index 1
The list is now: [1, 3, 5]
The list is now sorted!
```

Insertion Sort

Example:

```
>>> insertion_sort([6, 1, 8, 2])
```

🟩 sorted 🟥 unsorted ○ element



[6, 1, 8, 2]

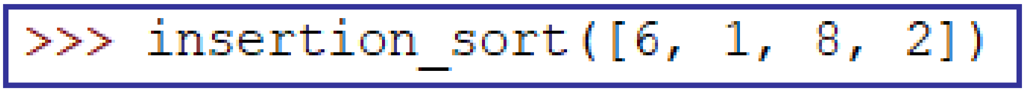
[1, 6, 8, 2]

[1, 6, 8, 2]

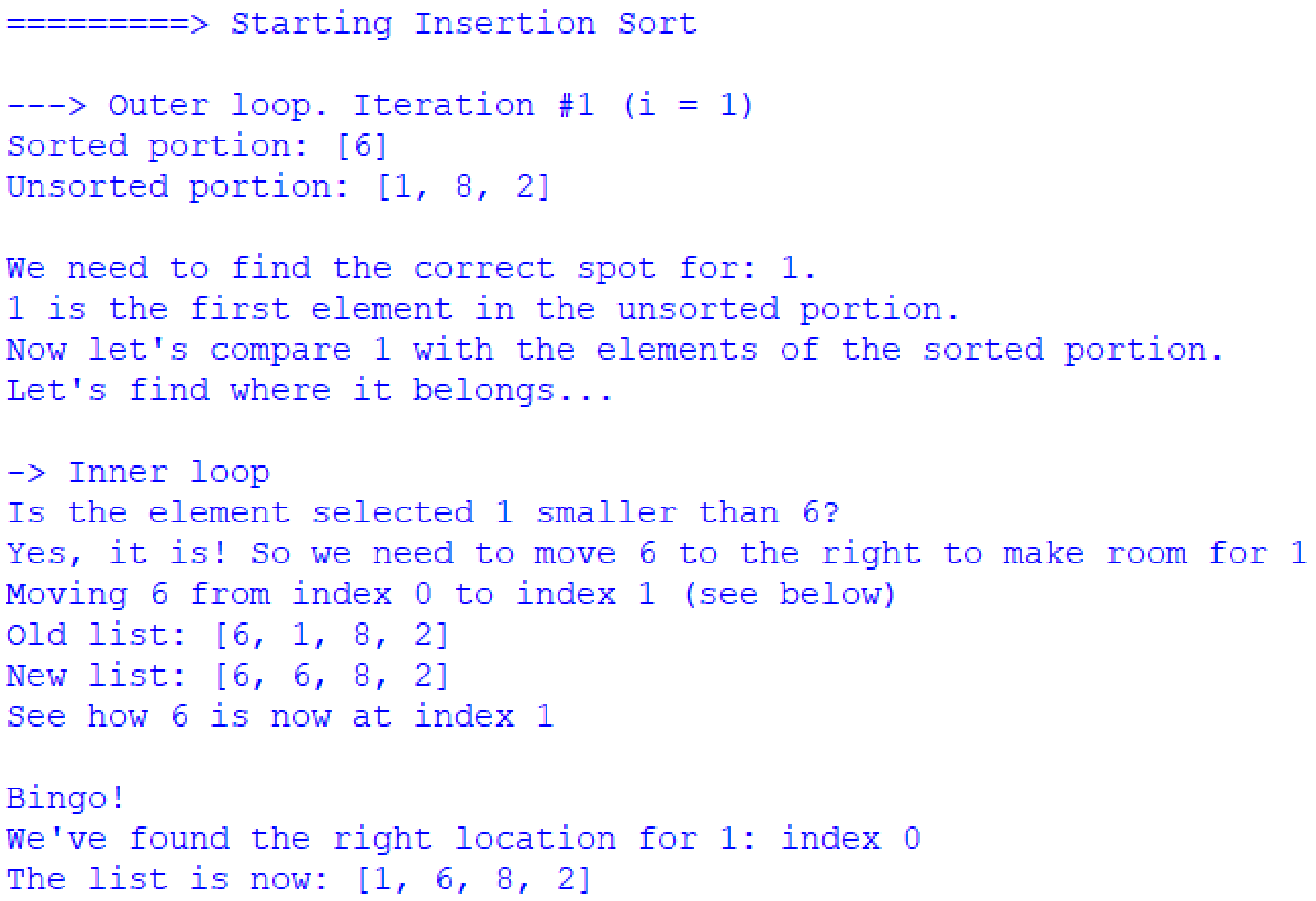
[1, 2, 6, 8]

Insertion Sort

Example:



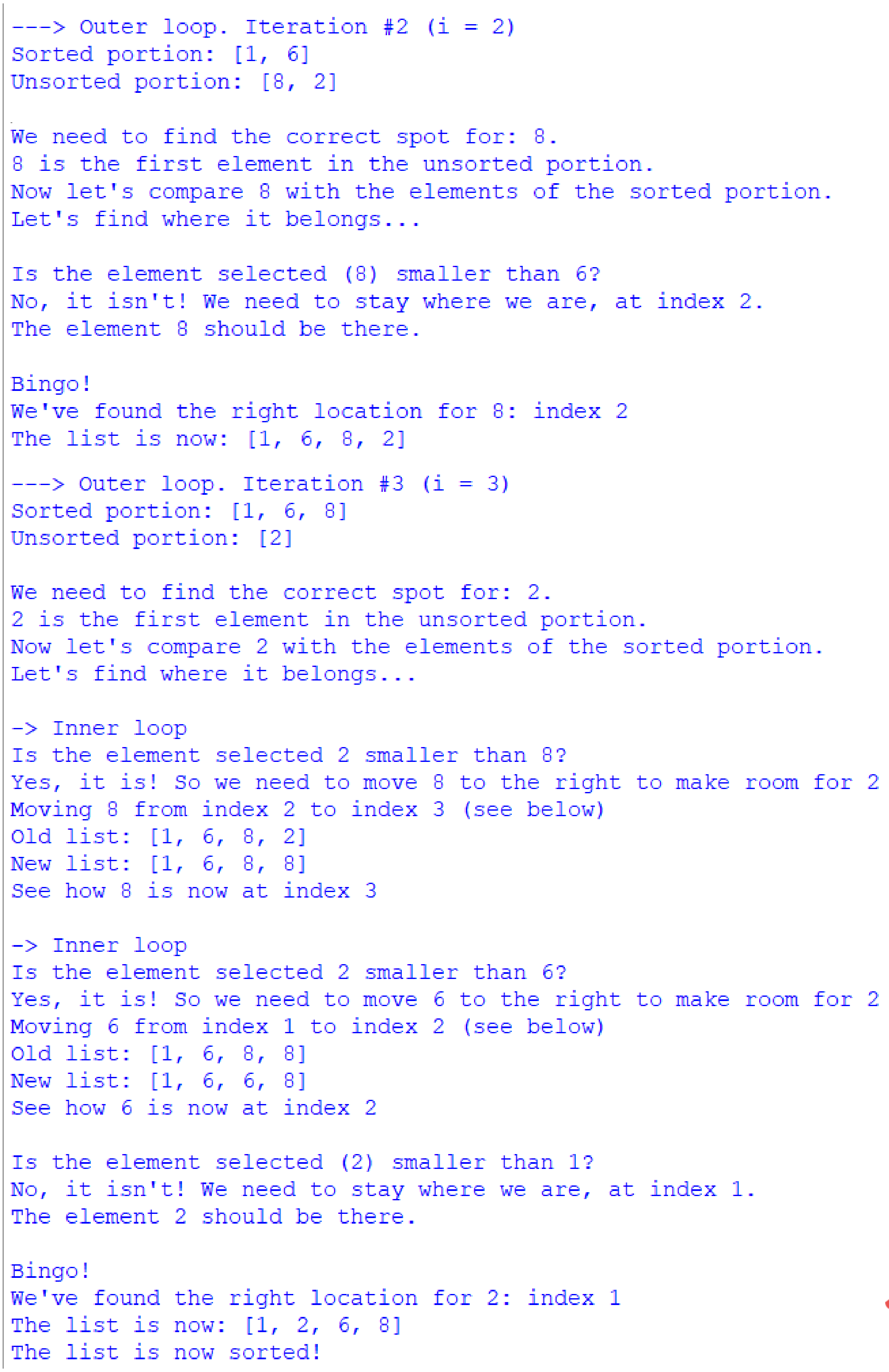
```
>>> insertion_sort([6, 1, 8, 2])
```



```
==========> Starting Insertion Sort

---> Outer loop. Iteration #1 (i = 1)
Sorted portion: [6]
Unsorted portion: [1, 8, 2]

We need to find the correct spot for: 1.
1 is the first element in the unsorted portion.
Now let's compare 1 with the elements of the sorted portion.
Let's find where it belongs...

-> Inner loop
Is the element selected 1 smaller than 6?
Yes, it is! So we need to move 6 to the right to make room for 1
Moving 6 from index 0 to index 1 (see below)
Old list: [6, 1, 8, 2]
New list: [6, 6, 8, 2]
See how 6 is now at index 1

Bingo!
We've found the right location for 1: index 0
The list is now: [1, 6, 8, 2]
```

Insertion Sort

Example:



```
---> Outer loop. Iteration #2 (i = 2)
Sorted portion: [1, 6]
Unsorted portion: [8, 2]

We need to find the correct spot for: 8.
8 is the first element in the unsorted portion.
Now let's compare 8 with the elements of the sorted portion.
Let's find where it belongs...

Is the element selected (8) smaller than 6?
No, it isn't! We need to stay where we are, at index 2.
The element 8 should be there.

Bingo!
We've found the right location for 8: index 2
The list is now: [1, 6, 8, 2]

---> Outer loop. Iteration #3 (i = 3)
Sorted portion: [1, 6, 8]
Unsorted portion: [2]

We need to find the correct spot for: 2.
2 is the first element in the unsorted portion.
Now let's compare 2 with the elements of the sorted portion.
Let's find where it belongs...

-> Inner loop
Is the element selected 2 smaller than 8?
Yes, it is! So we need to move 8 to the right to make room for 2
Moving 8 from index 2 to index 3 (see below)
Old list: [1, 6, 8, 2]
New list: [1, 6, 8, 8]
See how 8 is now at index 3

-> Inner loop
Is the element selected 2 smaller than 6?
Yes, it is! So we need to move 6 to the right to make room for 2
Moving 6 from index 1 to index 2 (see below)
Old list: [1, 6, 8, 8]
New list: [1, 6, 6, 8]
See how 6 is now at index 2

Is the element selected (2) smaller than 1?
No, it isn't! We need to stay where we are, at index 1.
The element 2 should be there.

Bingo!
We've found the right location for 2: index 1
The list is now: [1, 2, 6, 8]
The list is now sorted!
```