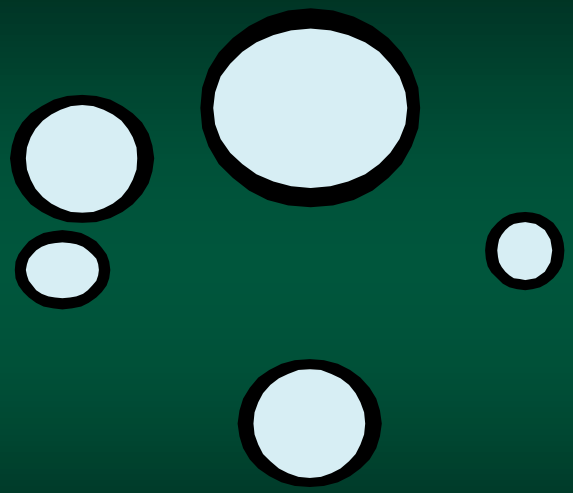
Sorting

Chapter 9



Bubble Sort

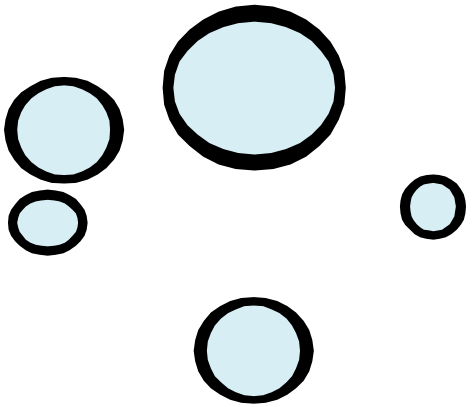
Chapter 9.1

Sorting

- Often, computers needs to *sort* a list – to put it in specific order
- Examples:
  - sorting scores by highest to lowest
  - sorting filenames in alphabetical order
  - sorting students by their student-id
- This can be done for the benefit of the user or, most often, for efficiency

Bubble Sort

- The *bubble sort* is one of the least efficient algorithms
  …but it is easy to understand
- Basic approach
  - "lighter" elements “bubble up” to the top of the array
  - "heavier" items sink to the bottom



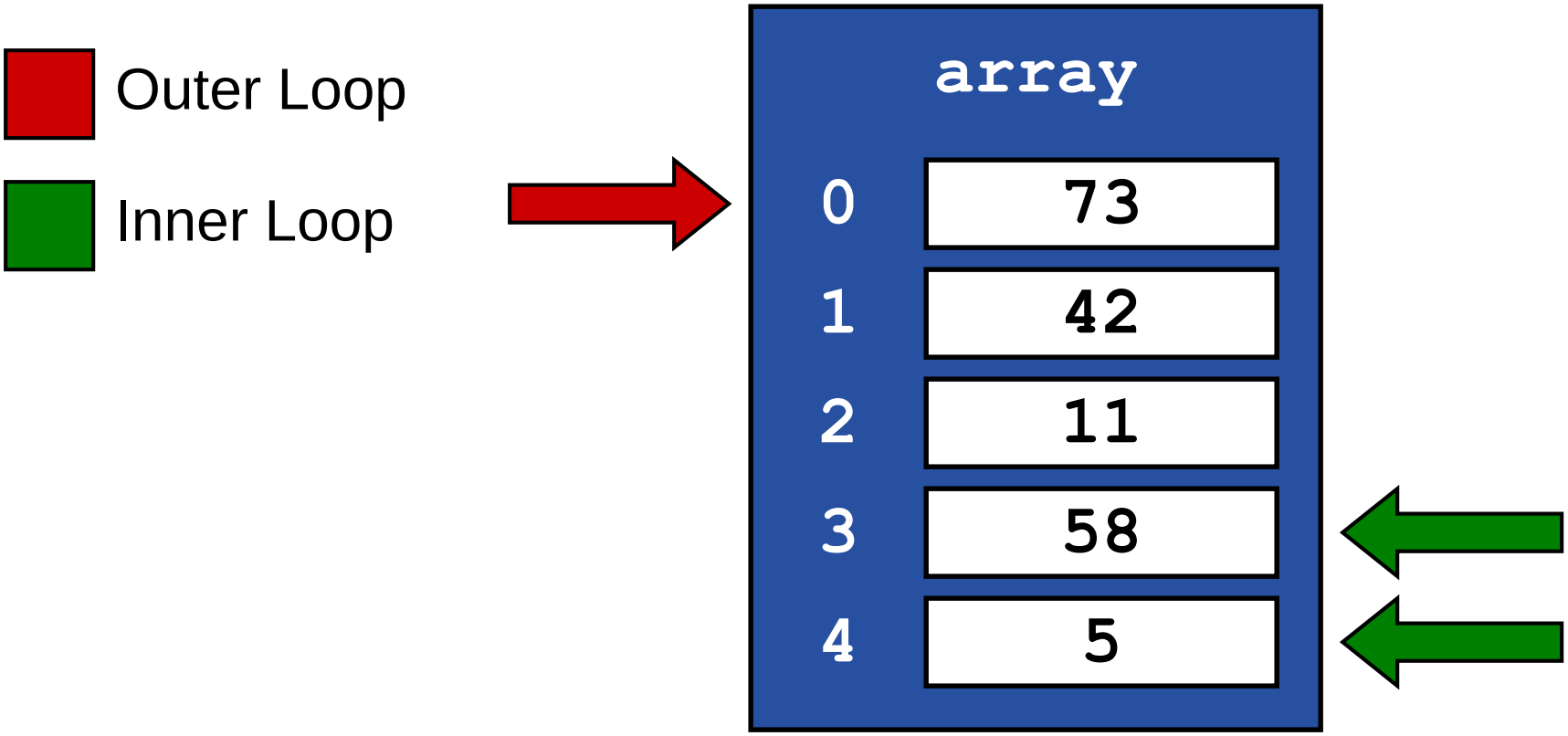
How It Works

- Consists of two For Loops
- Outer loop runs from the first to the last
- Inner loop …
  - runs from the bottom of the array *up* to the top (well, the position of the first loop)
  - it checks every two neighbor elements, if the they are out of order, it swaps them
  - so, the smallest element moves up the array

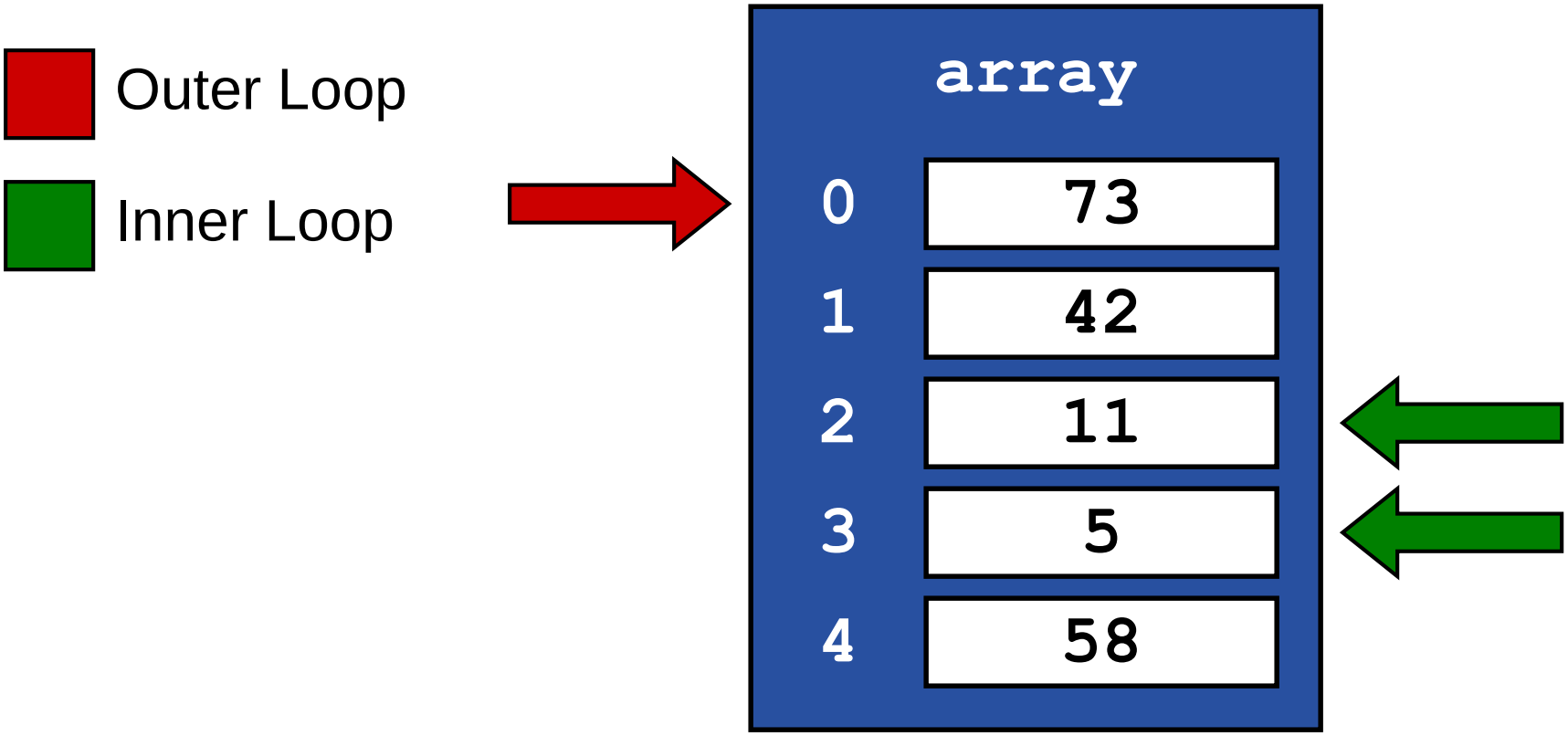
The Bubble Sort

```
For i = 0 to count-1
    For j = count-1 to i+1 Step -1

        If array[j-1] > array[j]
            //swap array[j-1] and array[j]
        End If

    End For
End For
```

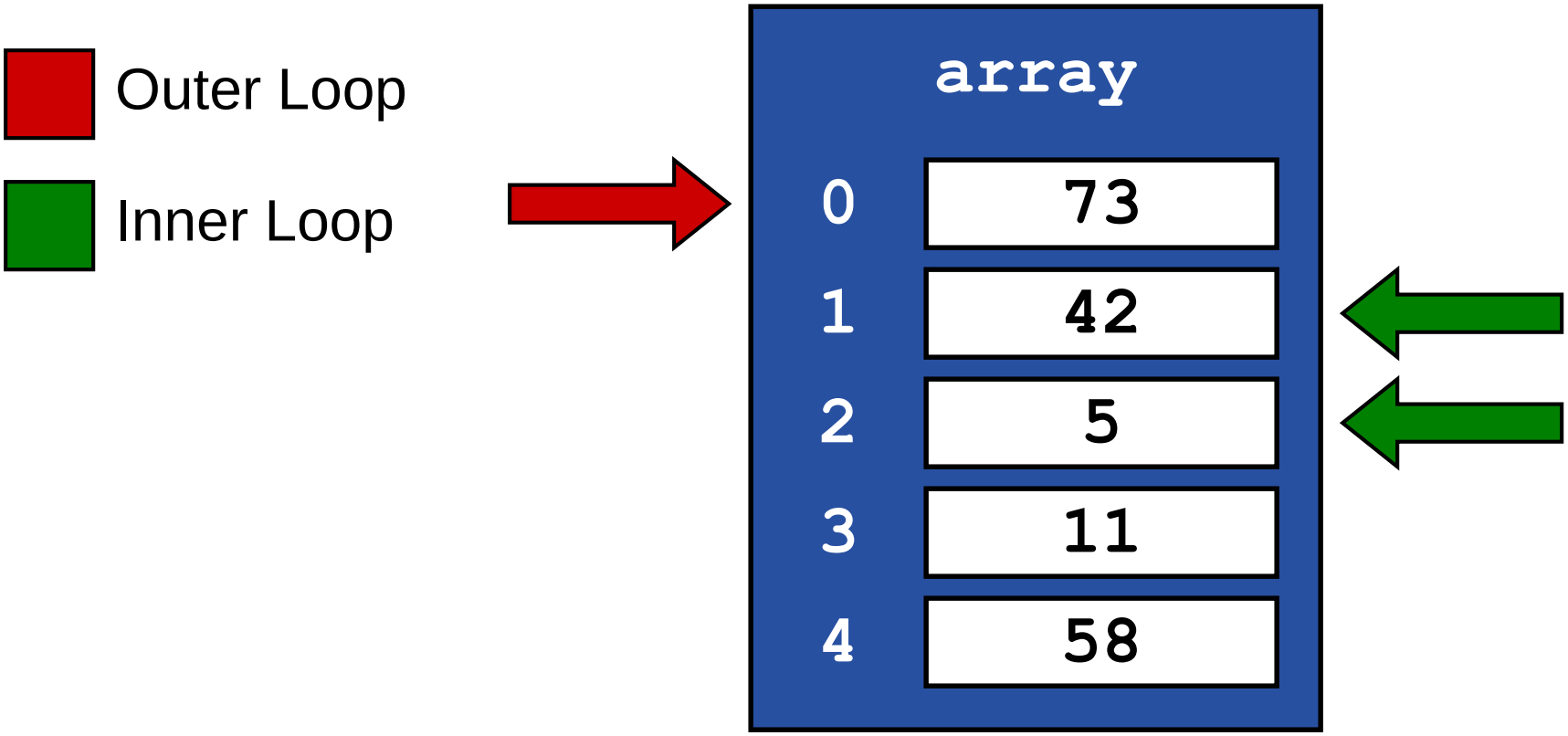
Bubble Sort Example



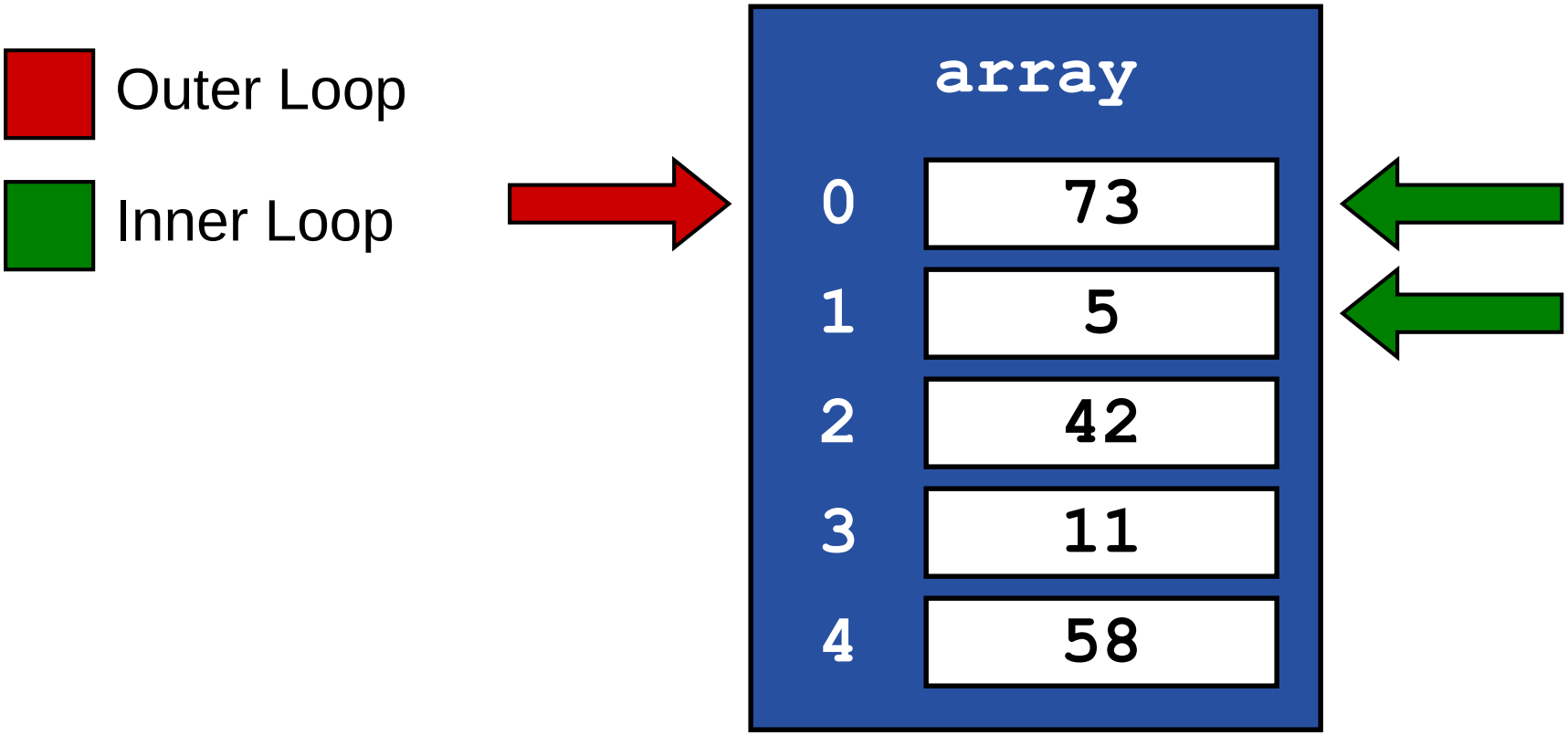
Bubble Sort Example



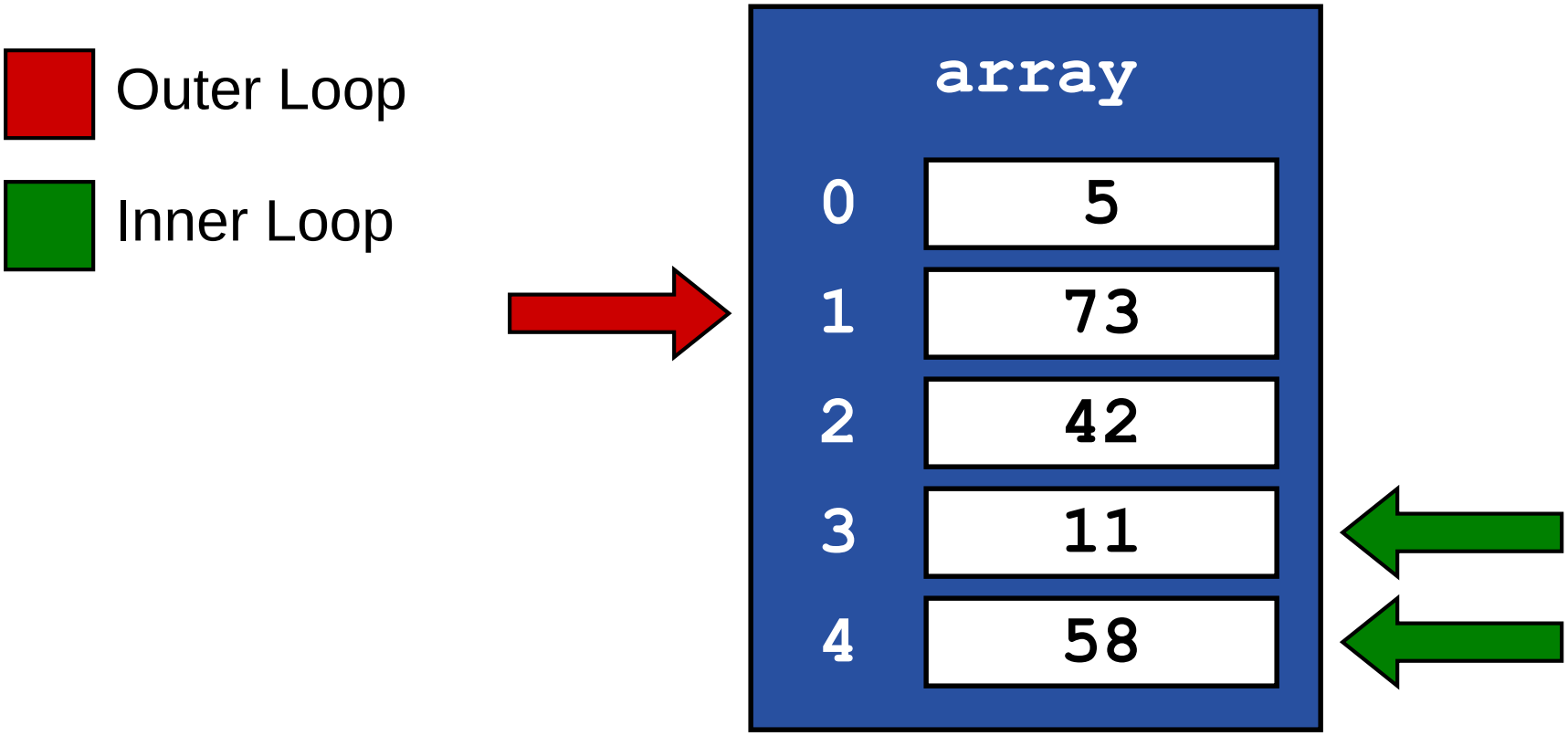
Bubble Sort Example



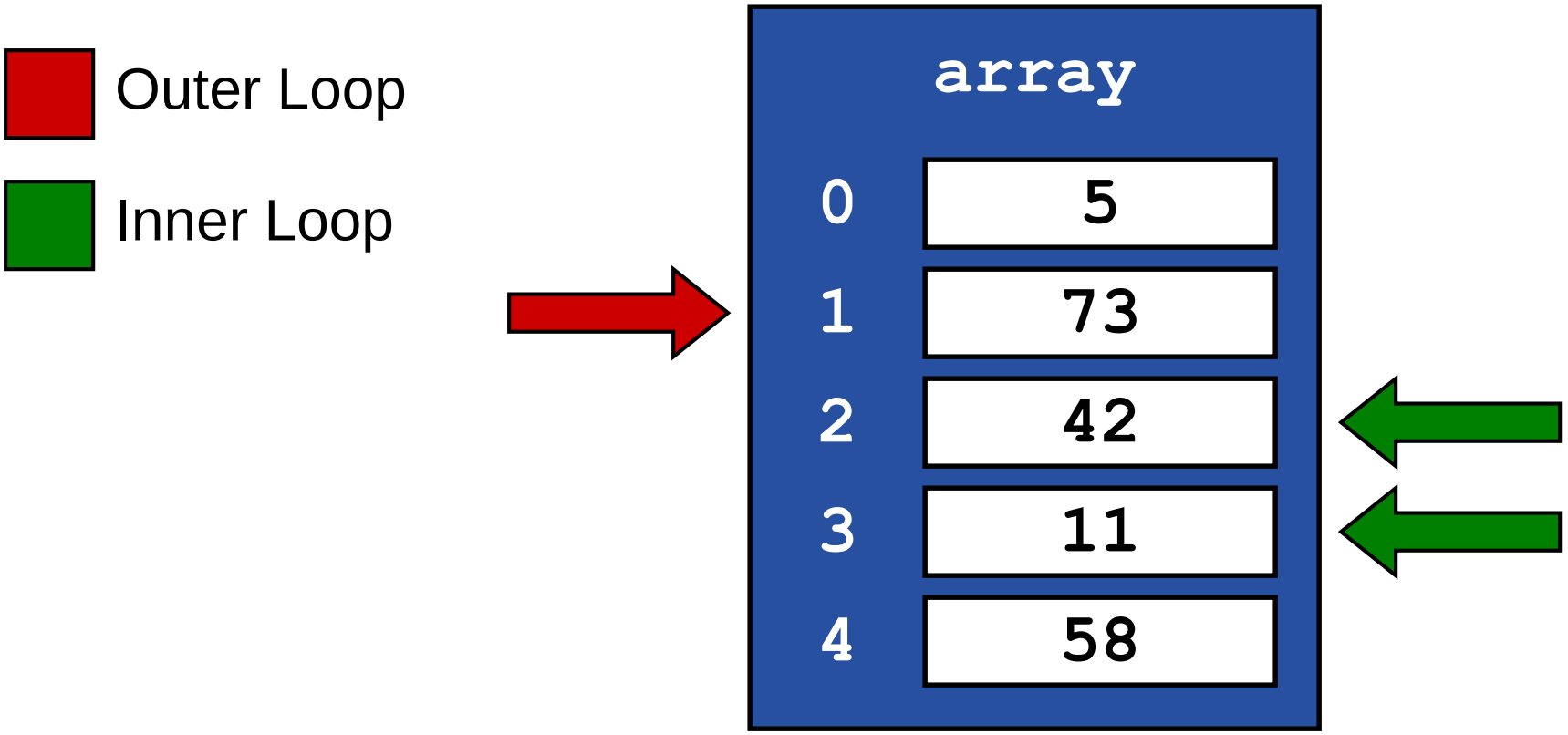
Bubble Sort Example



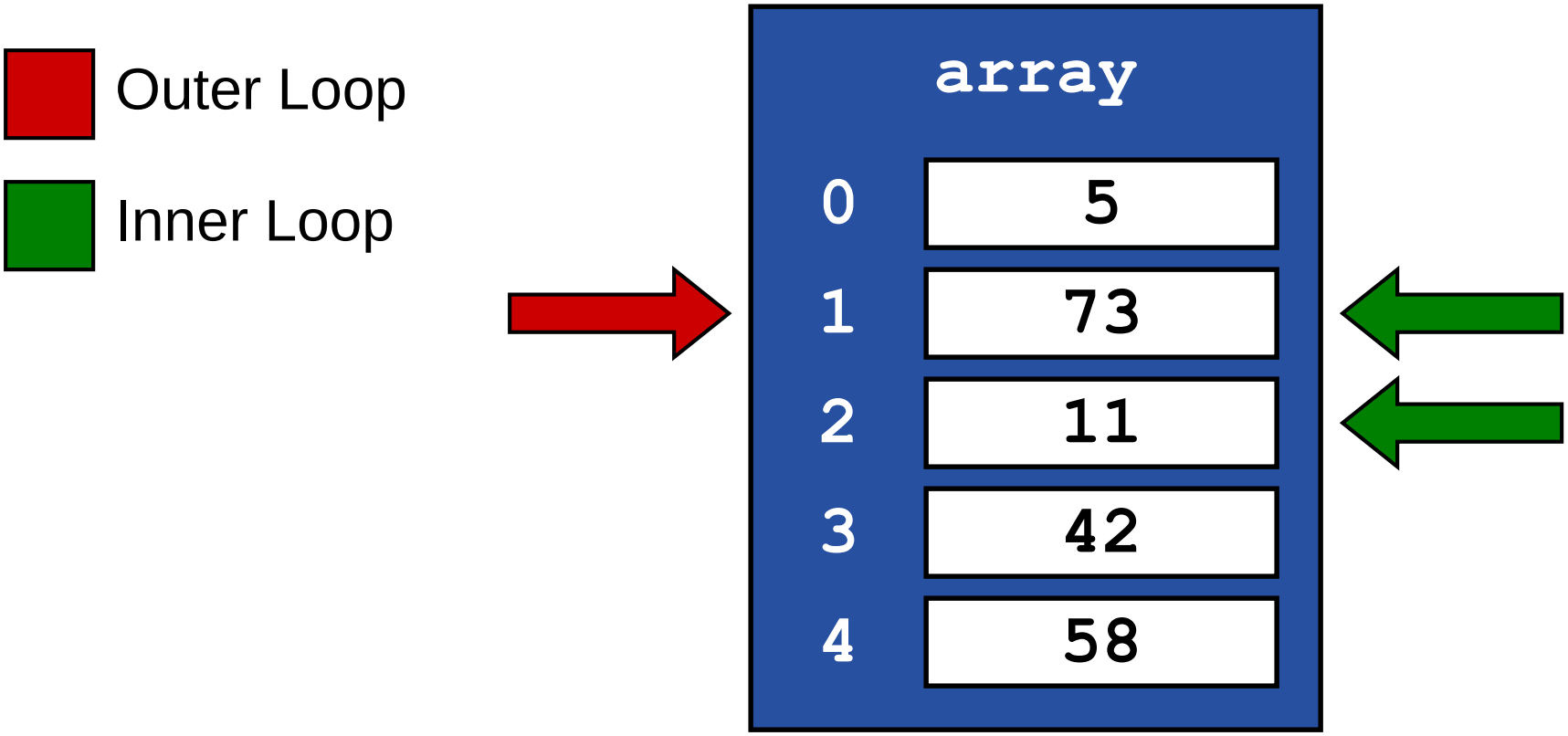
Bubble Sort Example



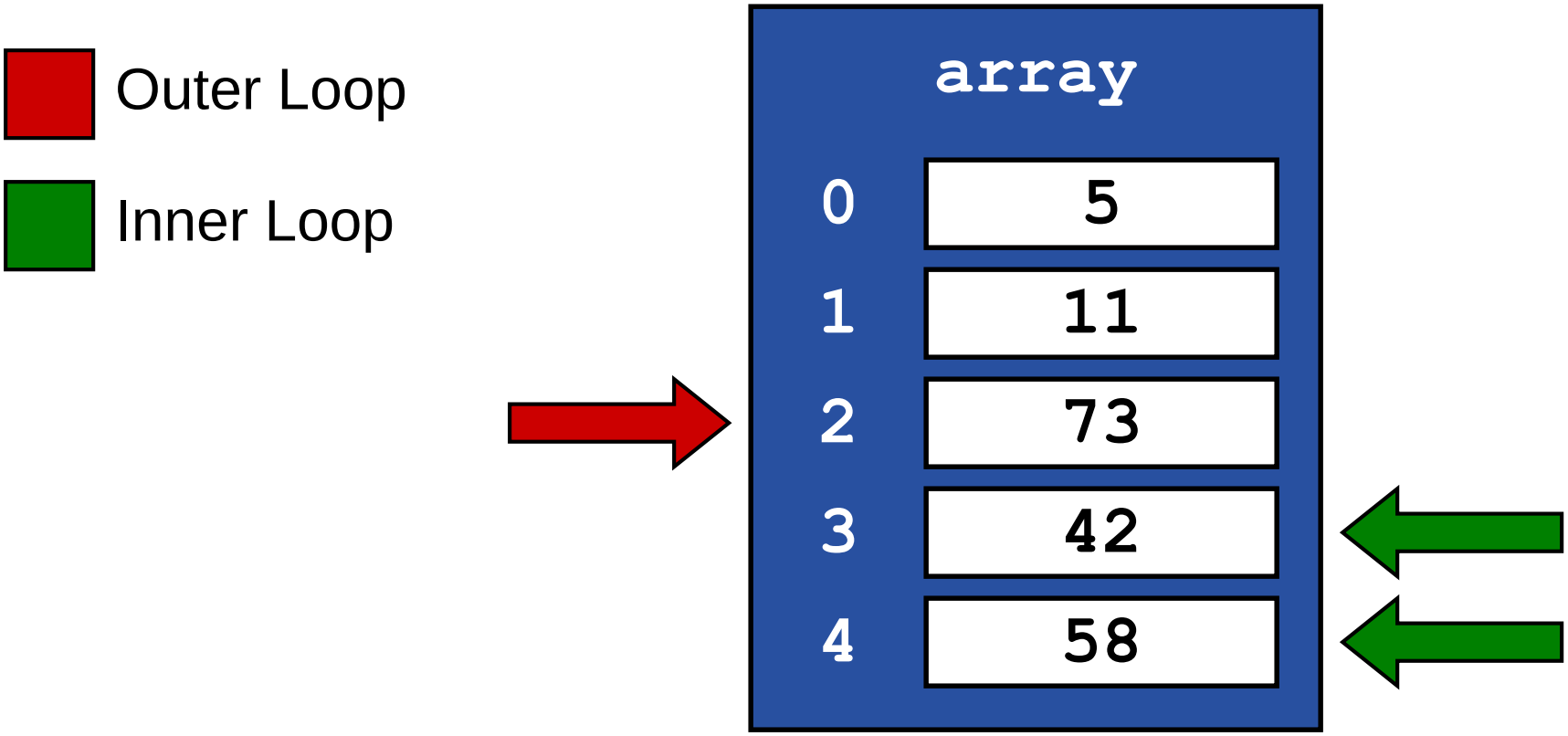
Bubble Sort Example



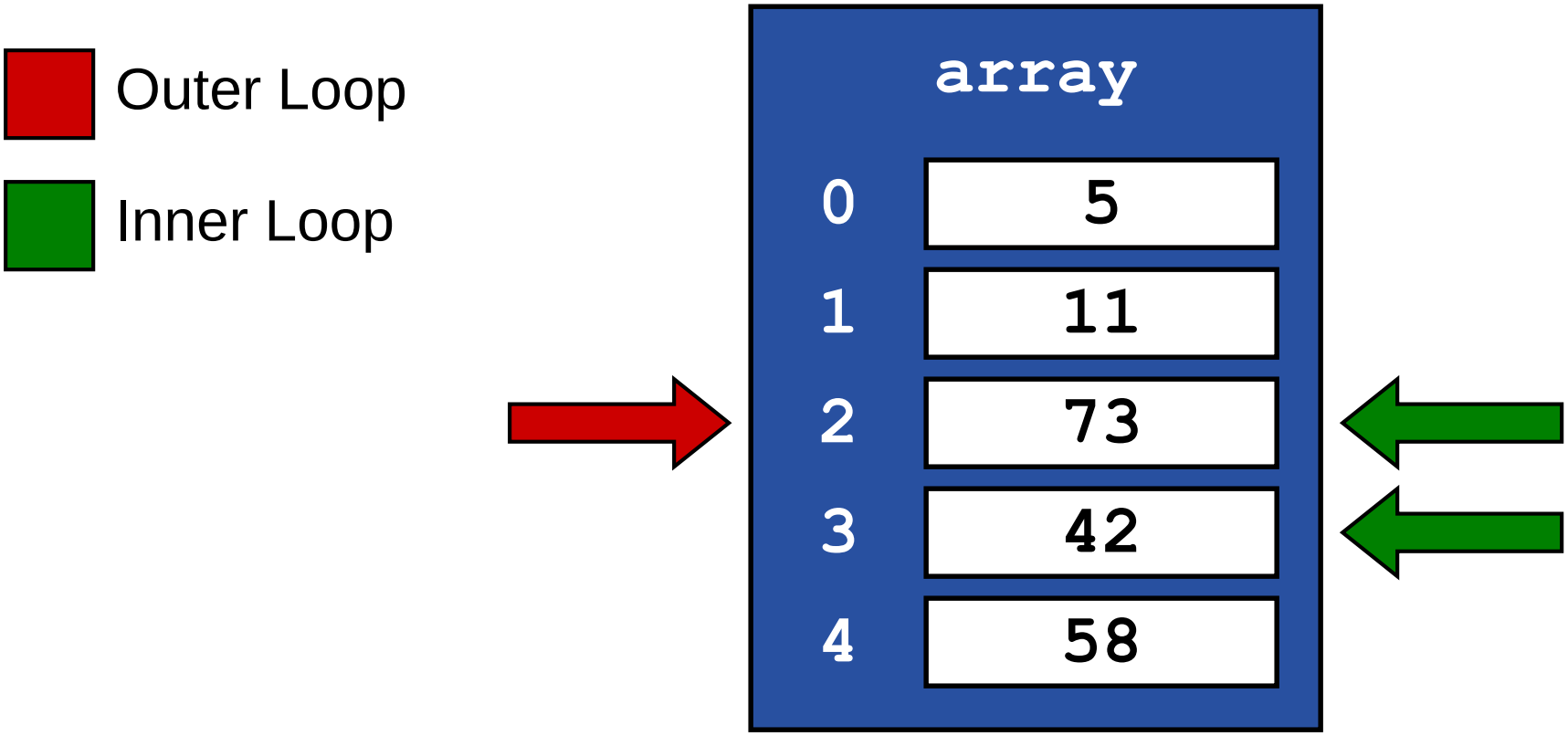
Bubble Sort Example



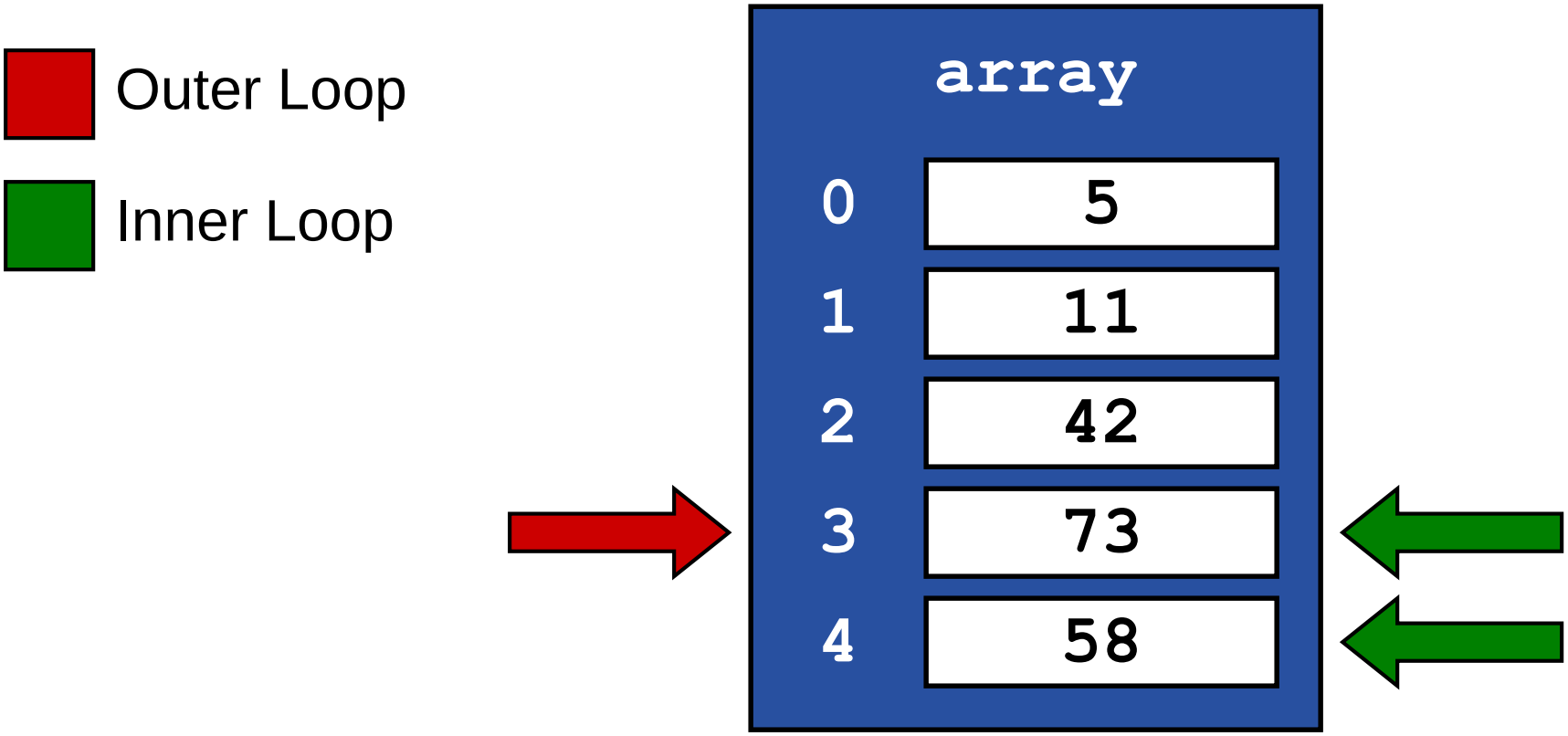
Bubble Sort Example



Bubble Sort Example



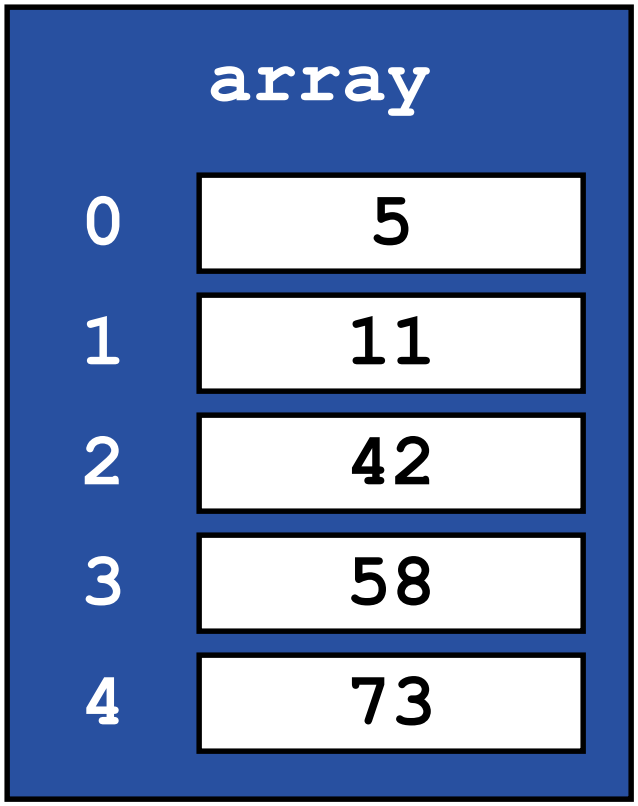
Bubble Sort Example



Bubble Sort Example

Outer Loop

Inner Loop

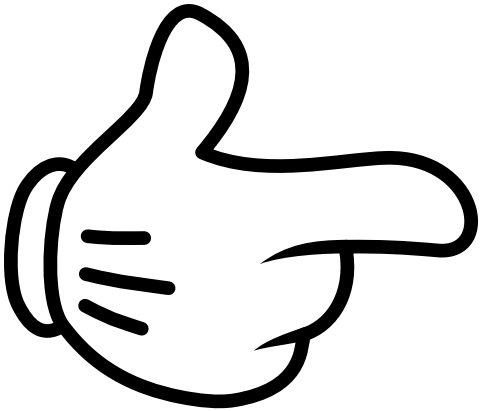


Selection Sort

Chapter 9.2

Selection Sort

- The *Selection Sort* is a similar to the Bubble Sort
- However…
  - rather than "bubble up" smaller items, it scans the entire array
  - it finds the smallest element
  - only *then* does it swap the values



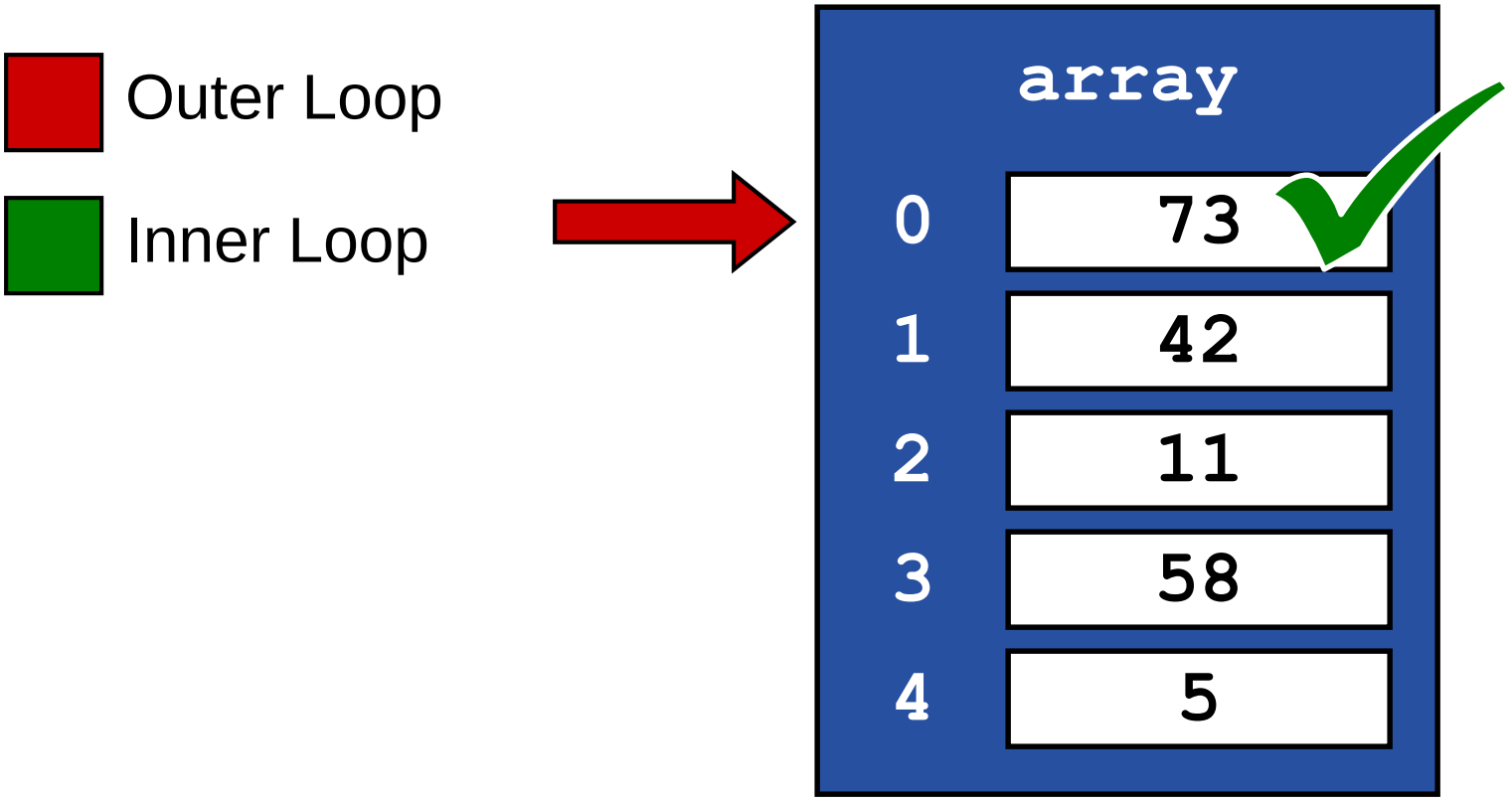
Selection Sort

- Like the Bubble Sort, it consists of two For Loops – one outer and one inner
- Outer loop runs from the first to the last
- Inner loop …
  - starts at the position of the outer loop
  - scans down and finds the *smallest* value
- Then, after the scan, do a single swap

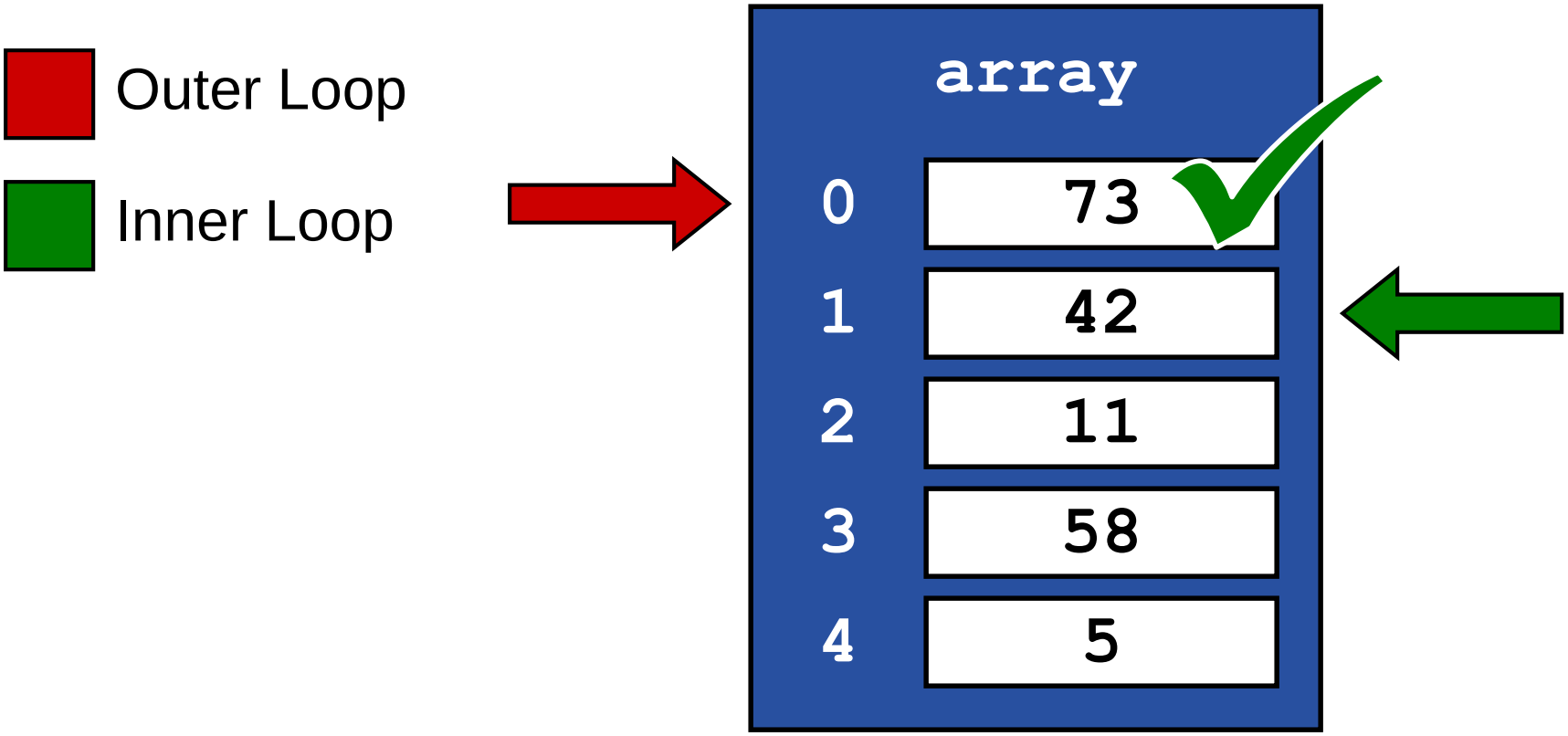
The Selection Sort

```
For i = 0 To count-1

   Set best = i;

   For j = i to count - 1
      If array[j] < array[best]
        Set best = j
      End If
   End For
   //swap array[i] and array[best]
End For
```

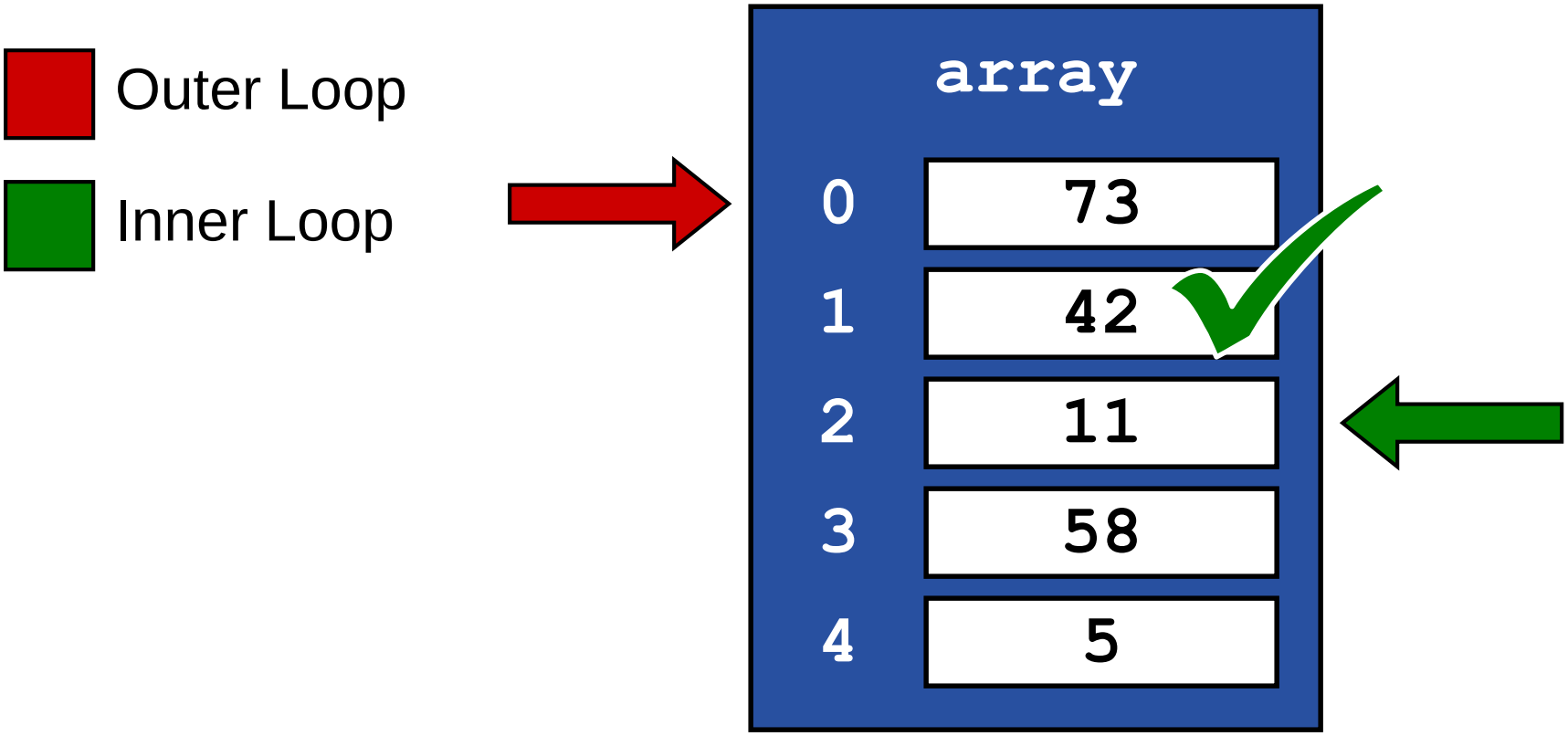
Selection Sort Example



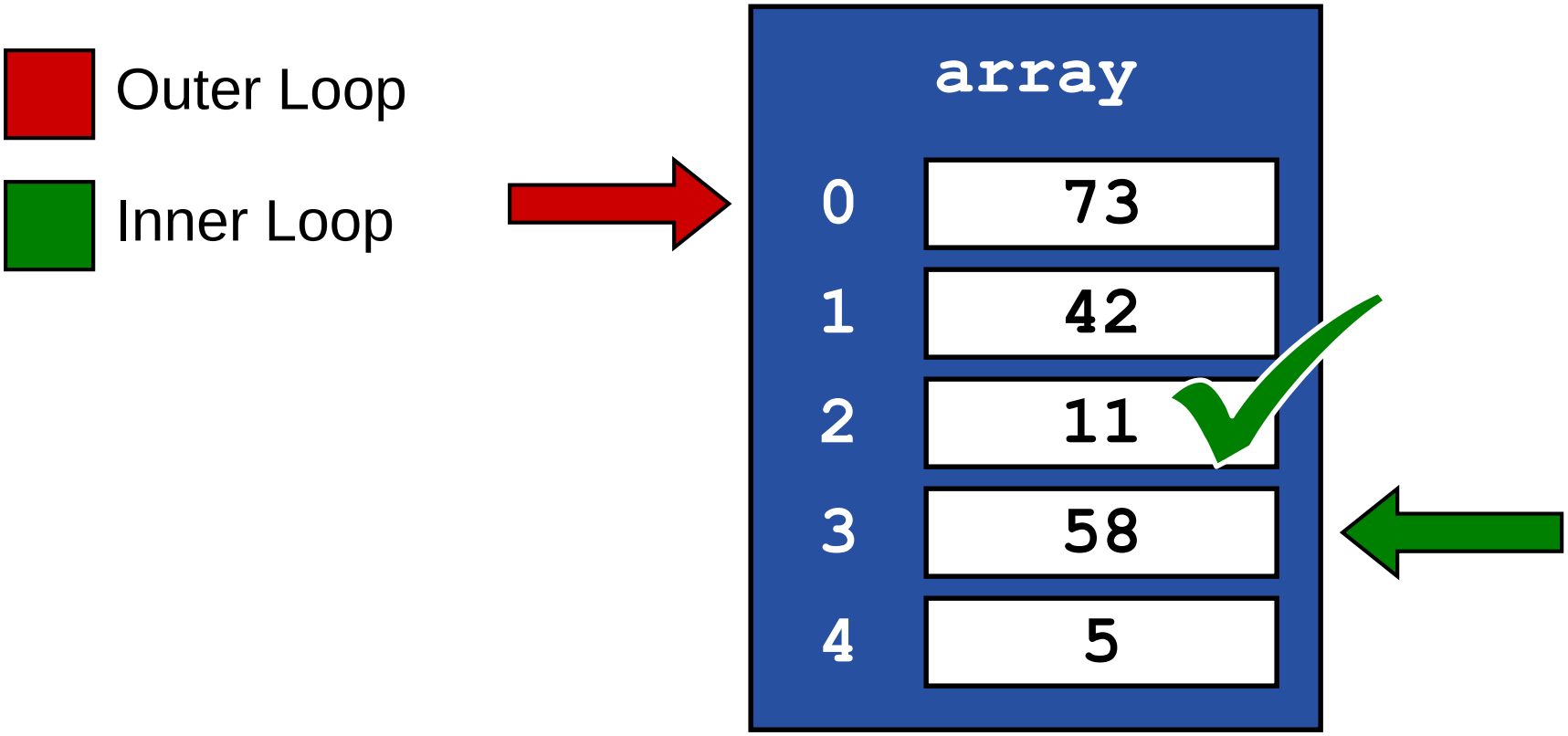
Selection Sort Example



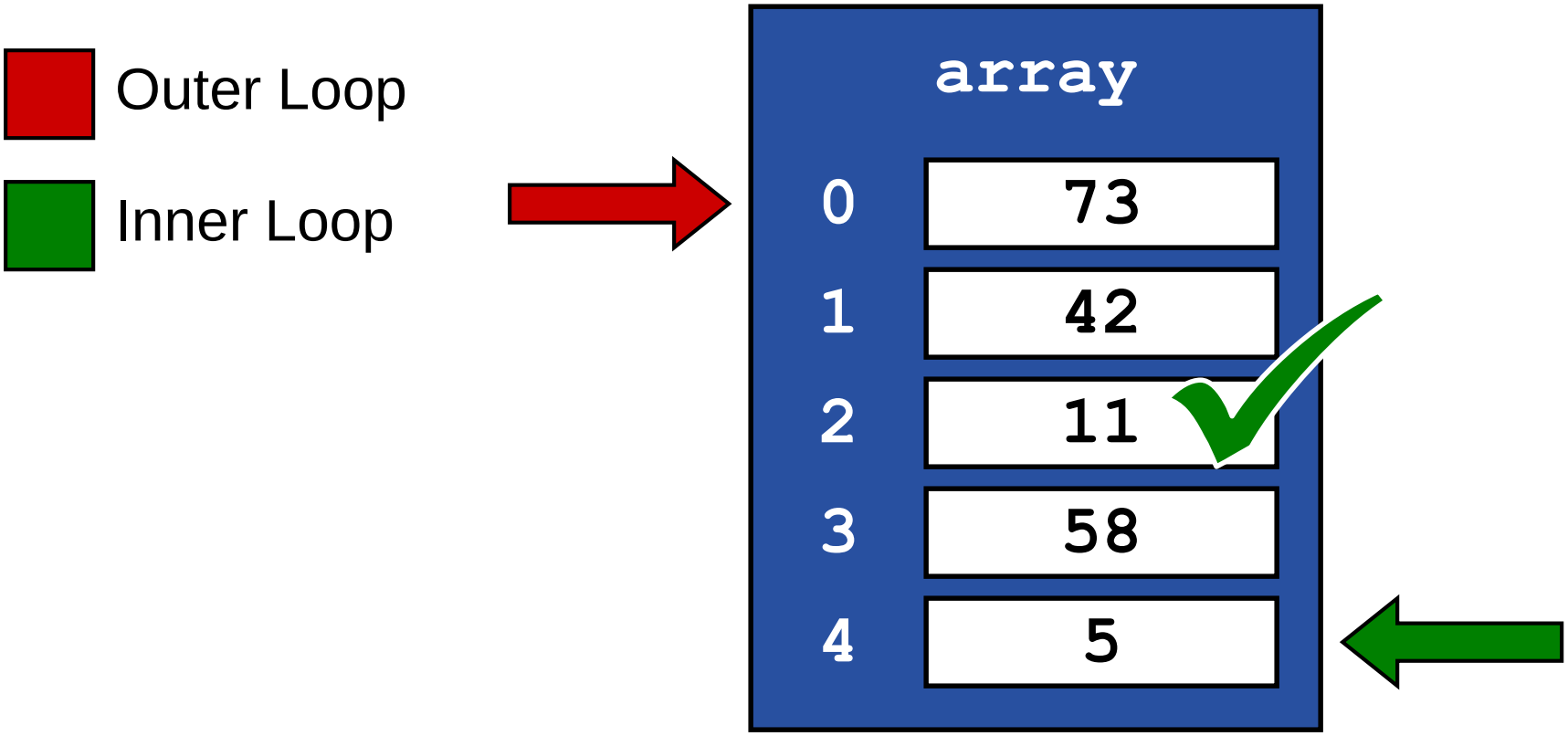
Selection Sort Example



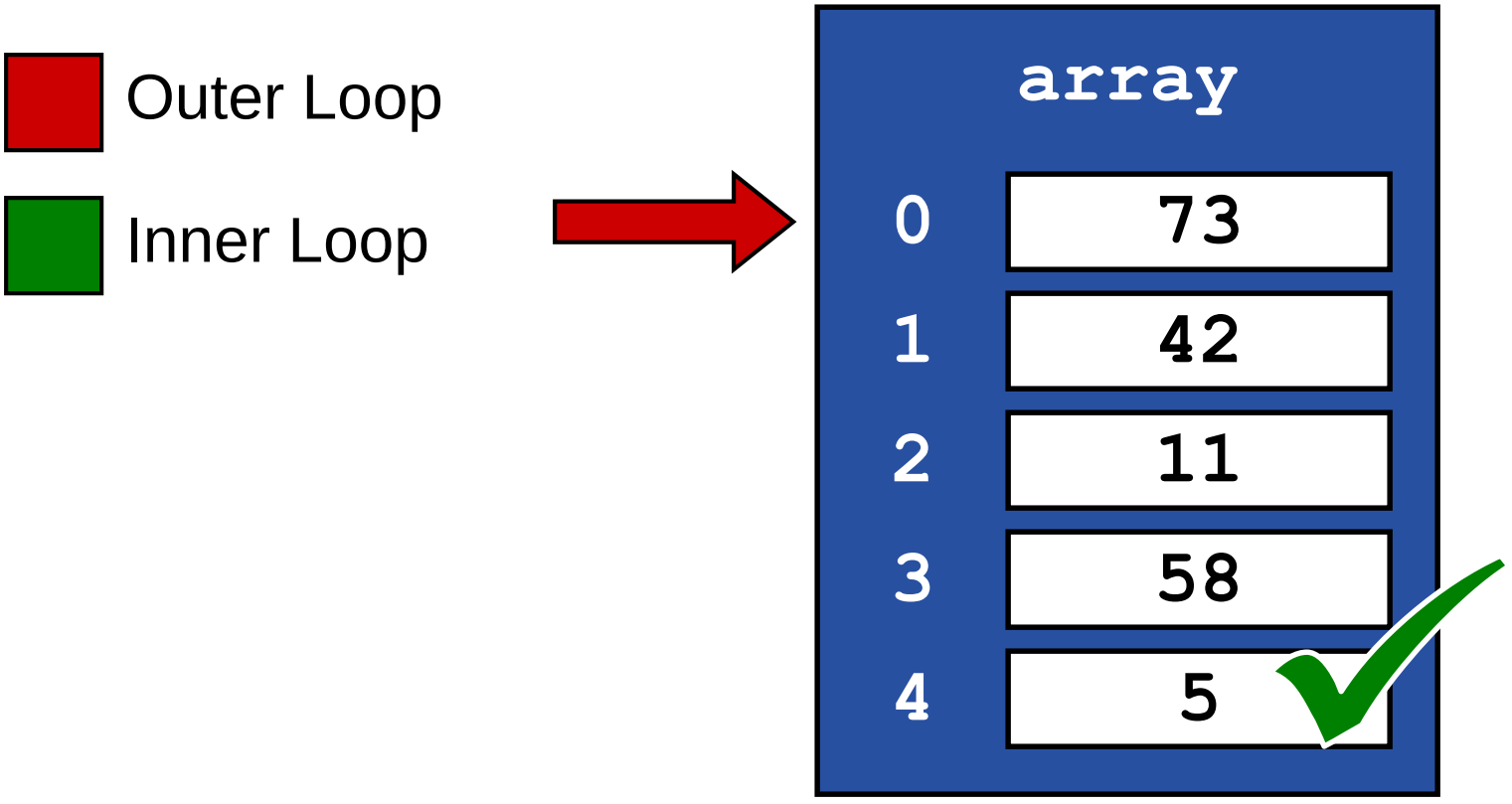
Selection Sort Example



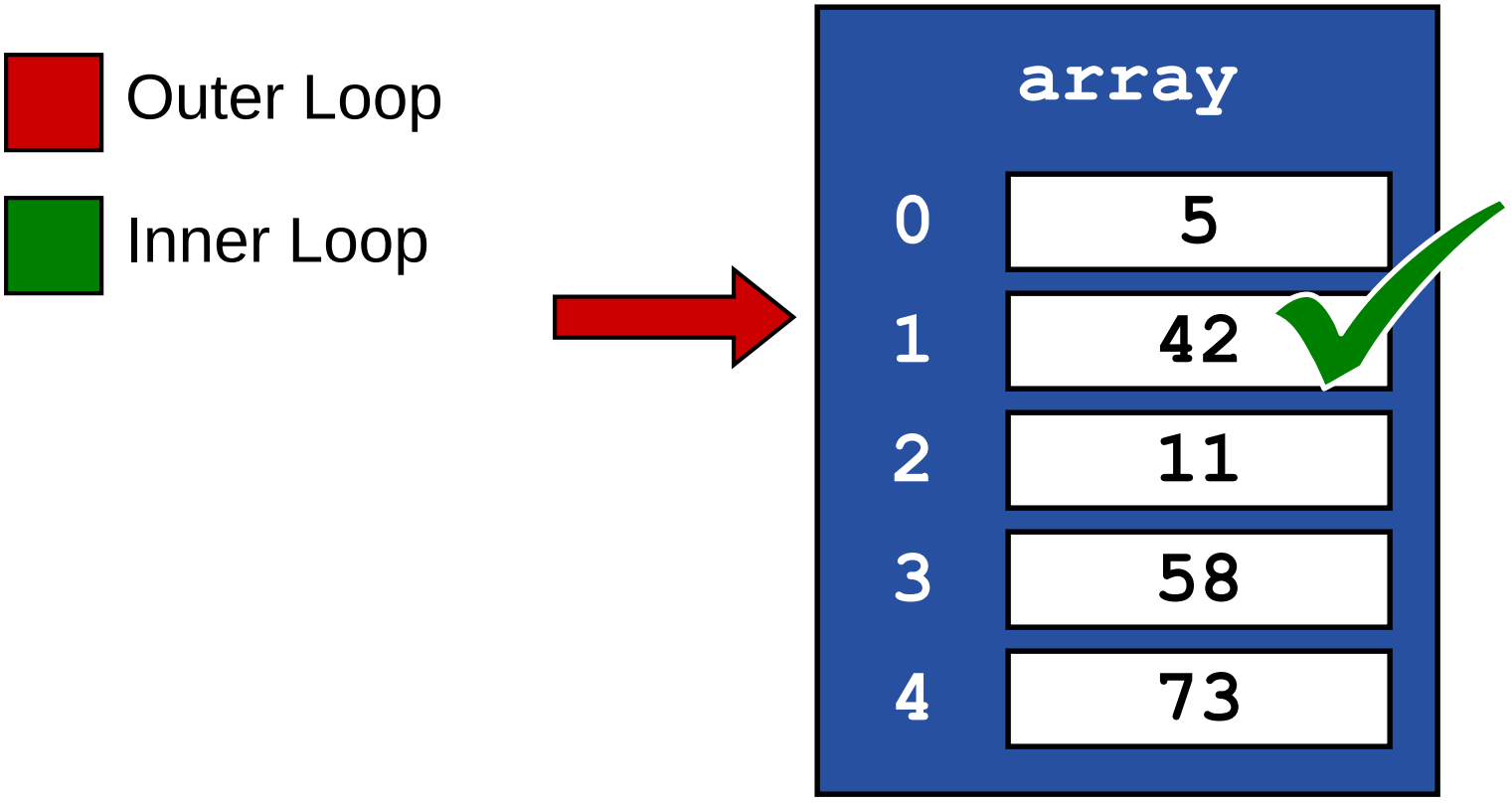
Selection Sort Example



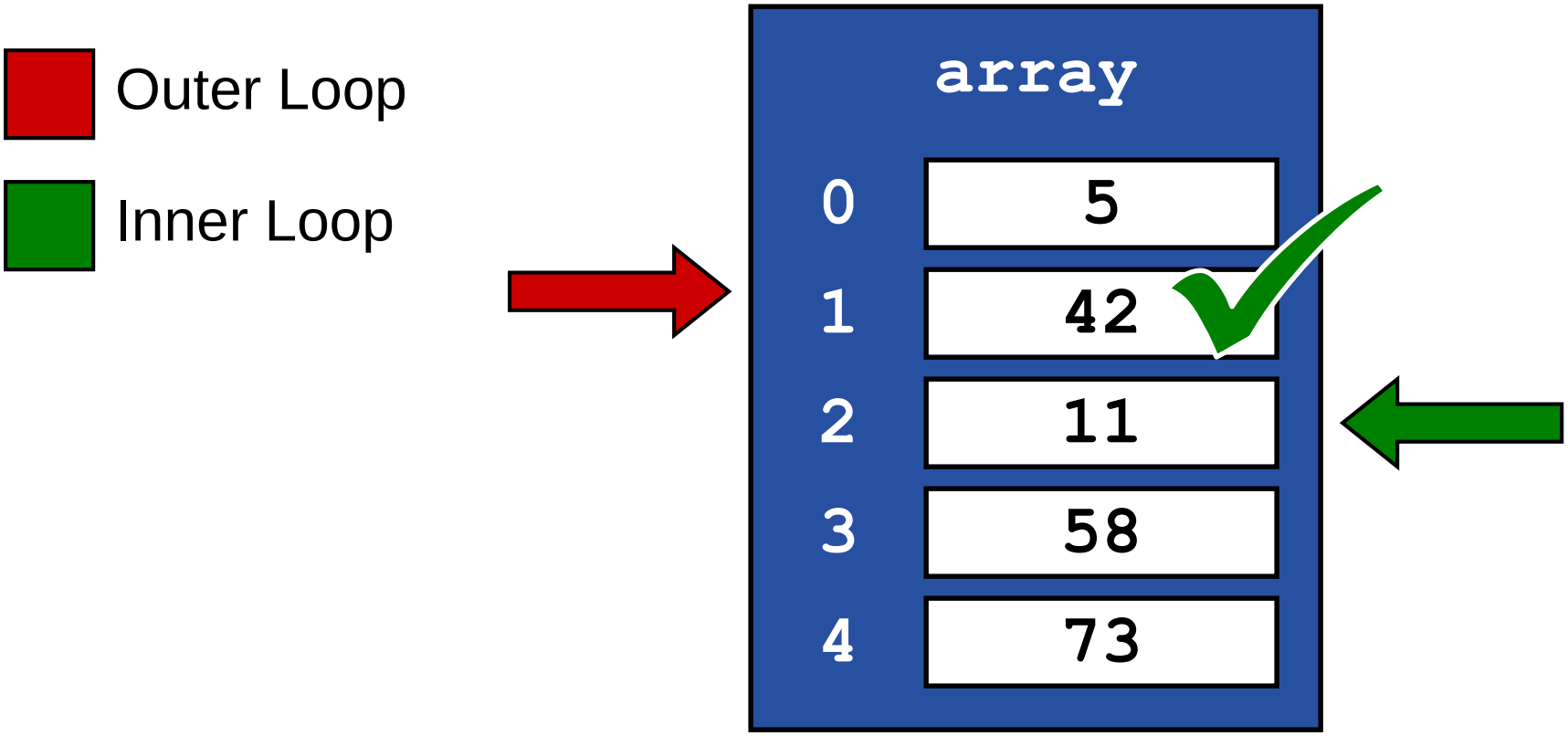
Selection Sort Example



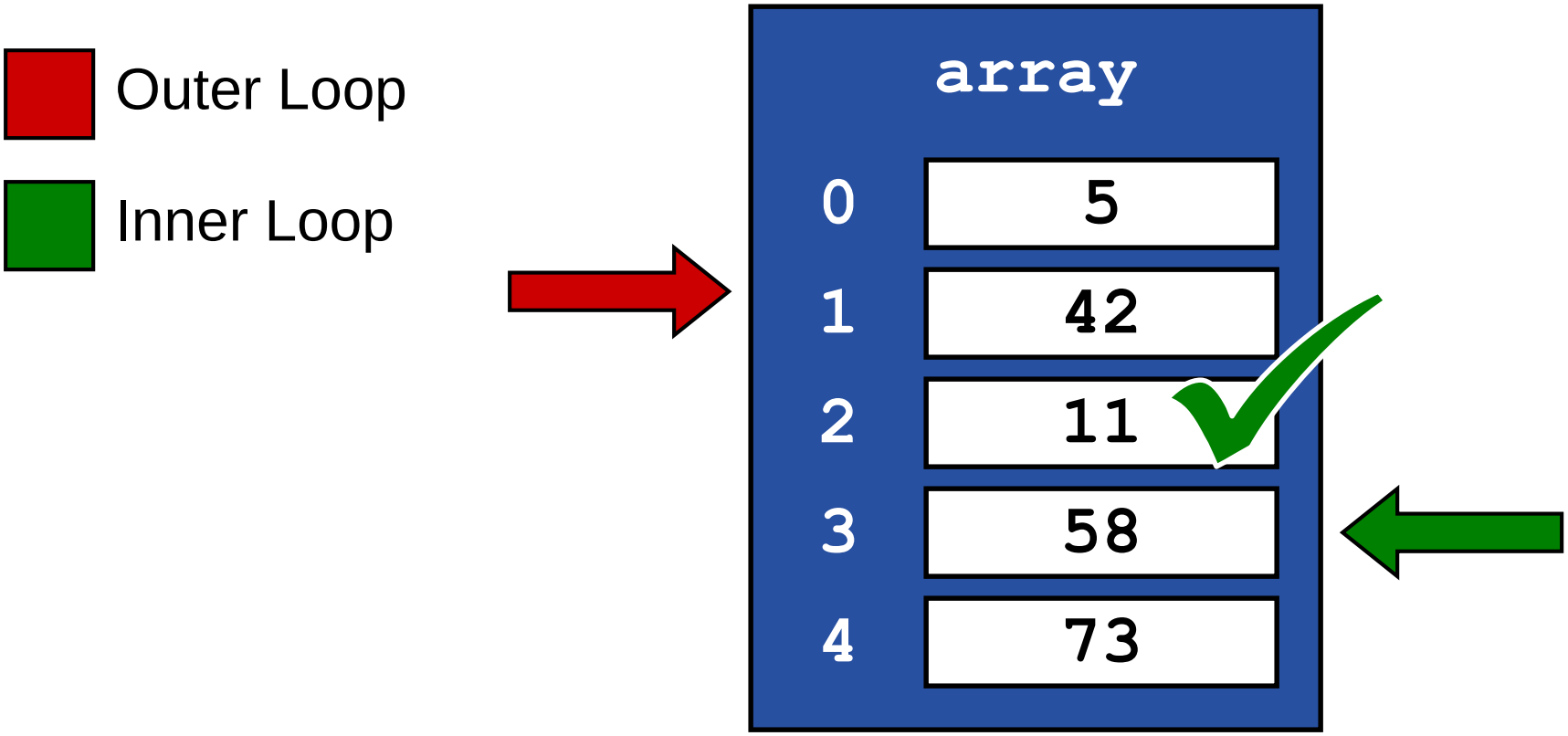
Selection Sort Example



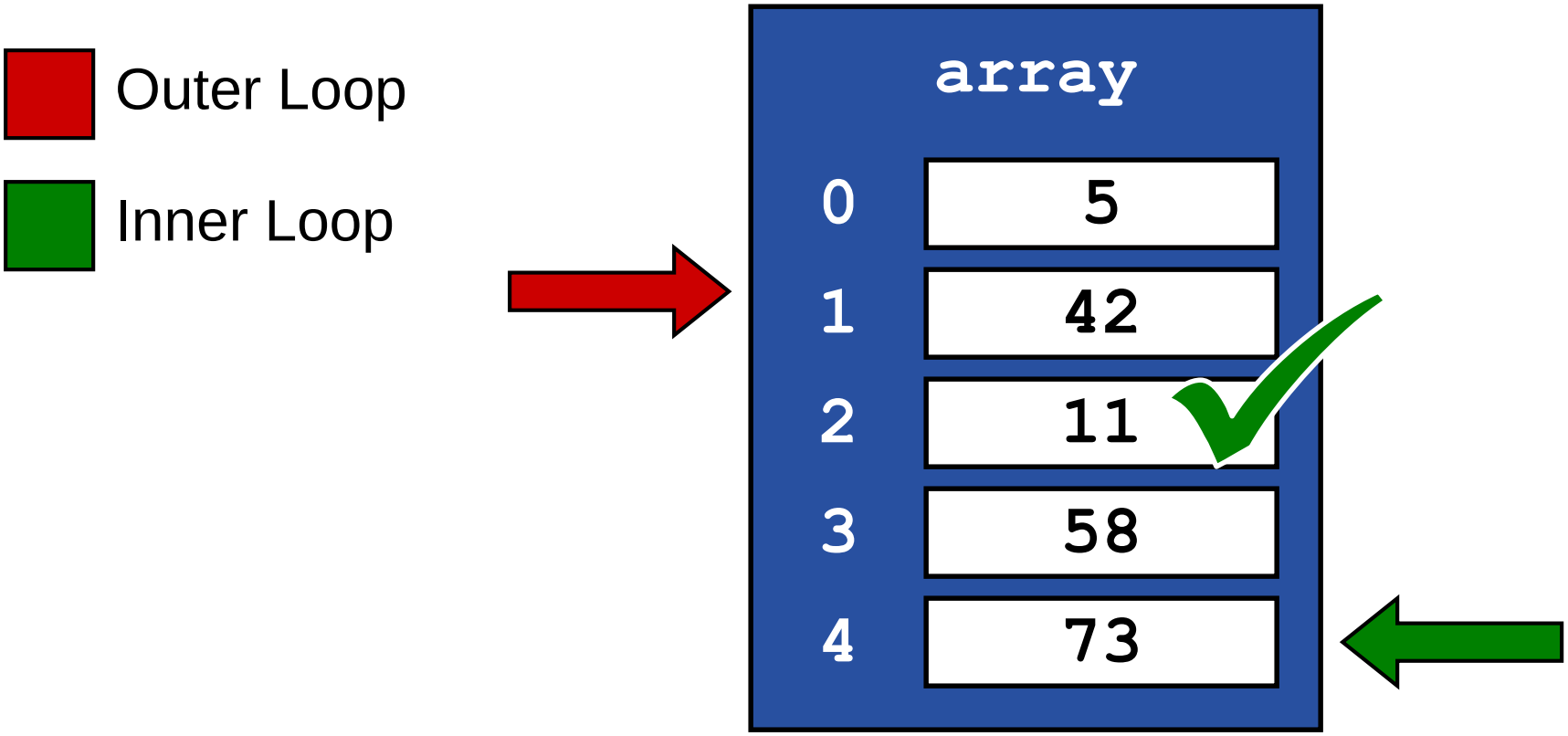
Selection Sort Example



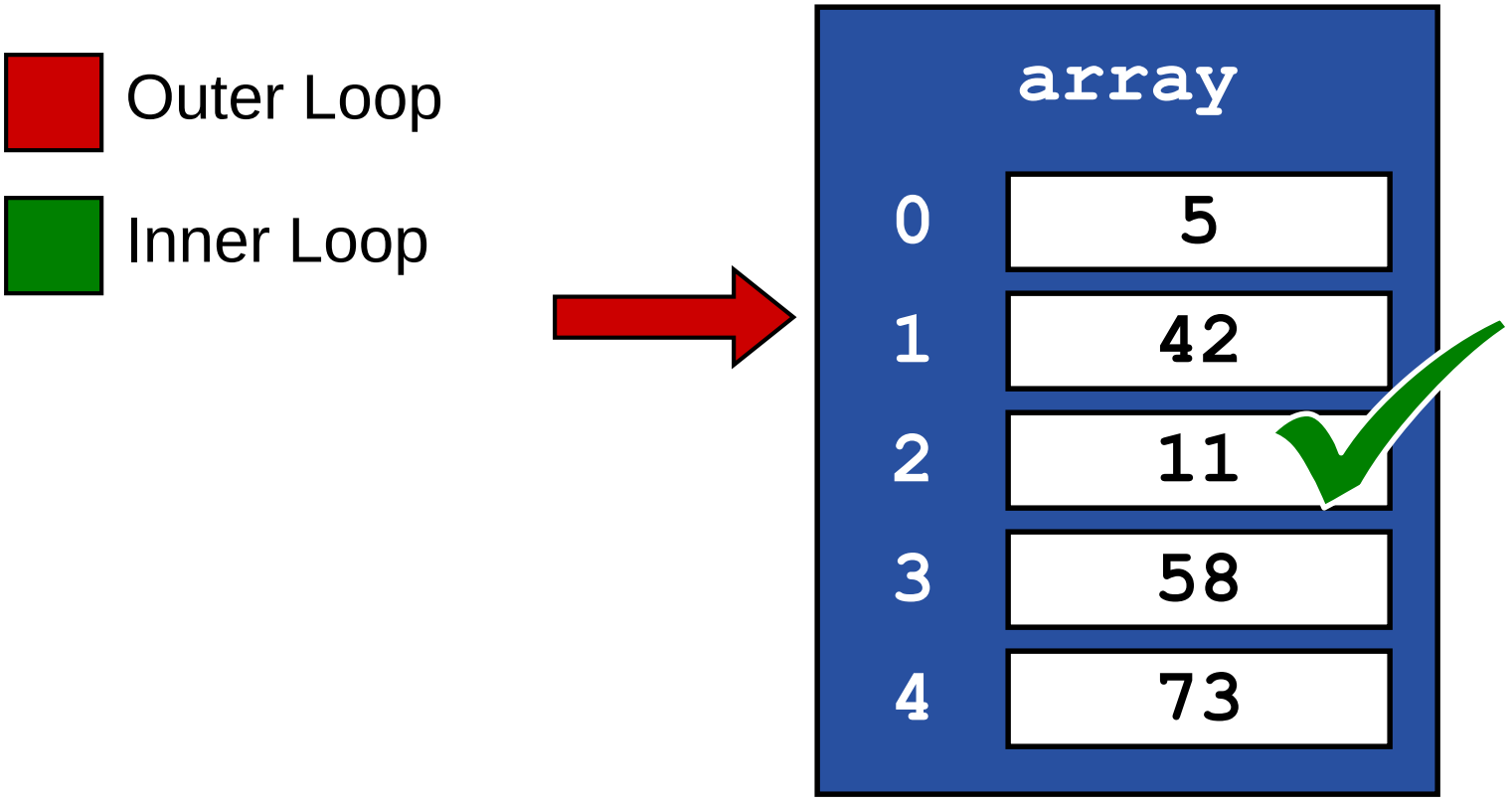
Selection Sort Example



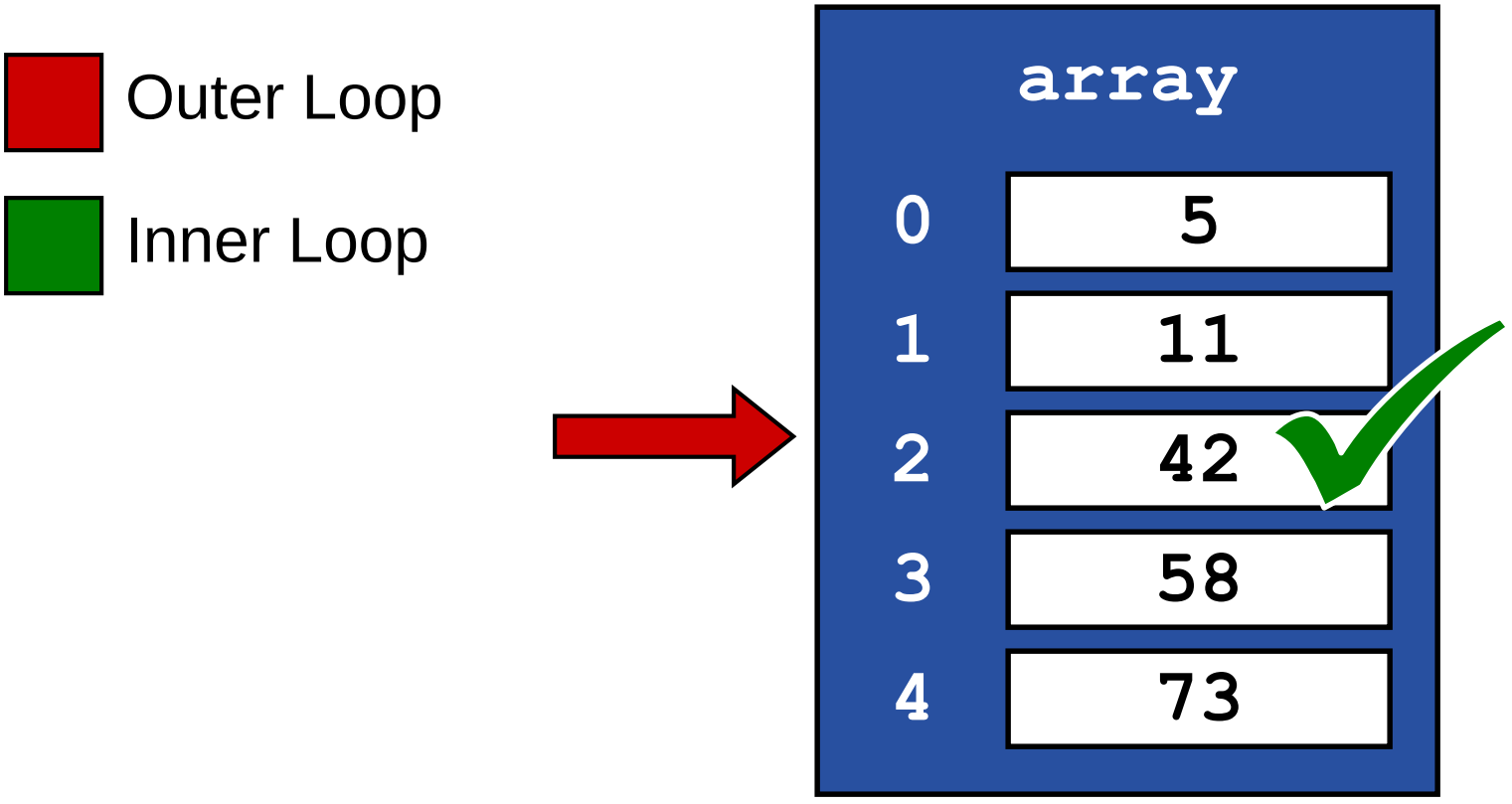
Selection Sort Example



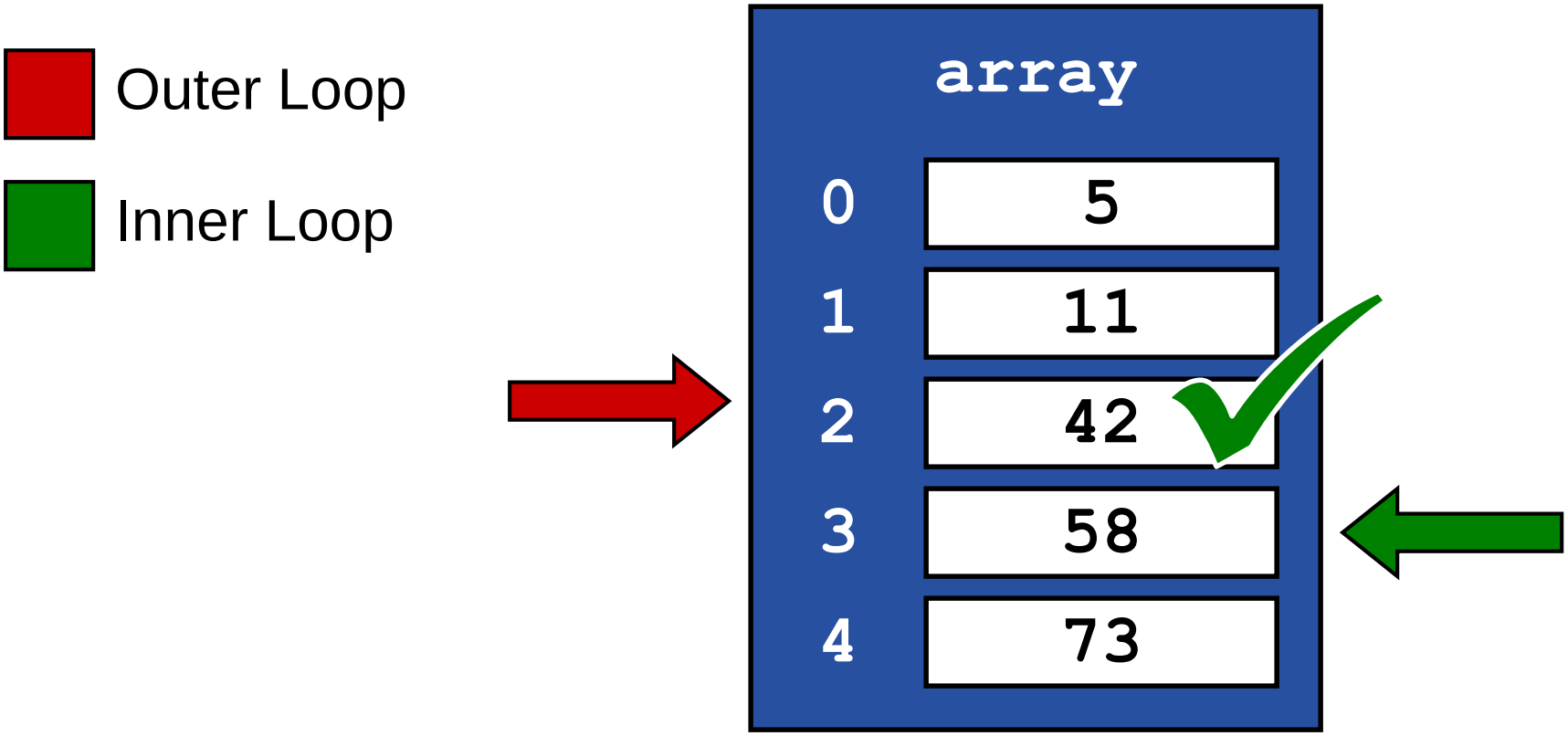
Selection Sort Example



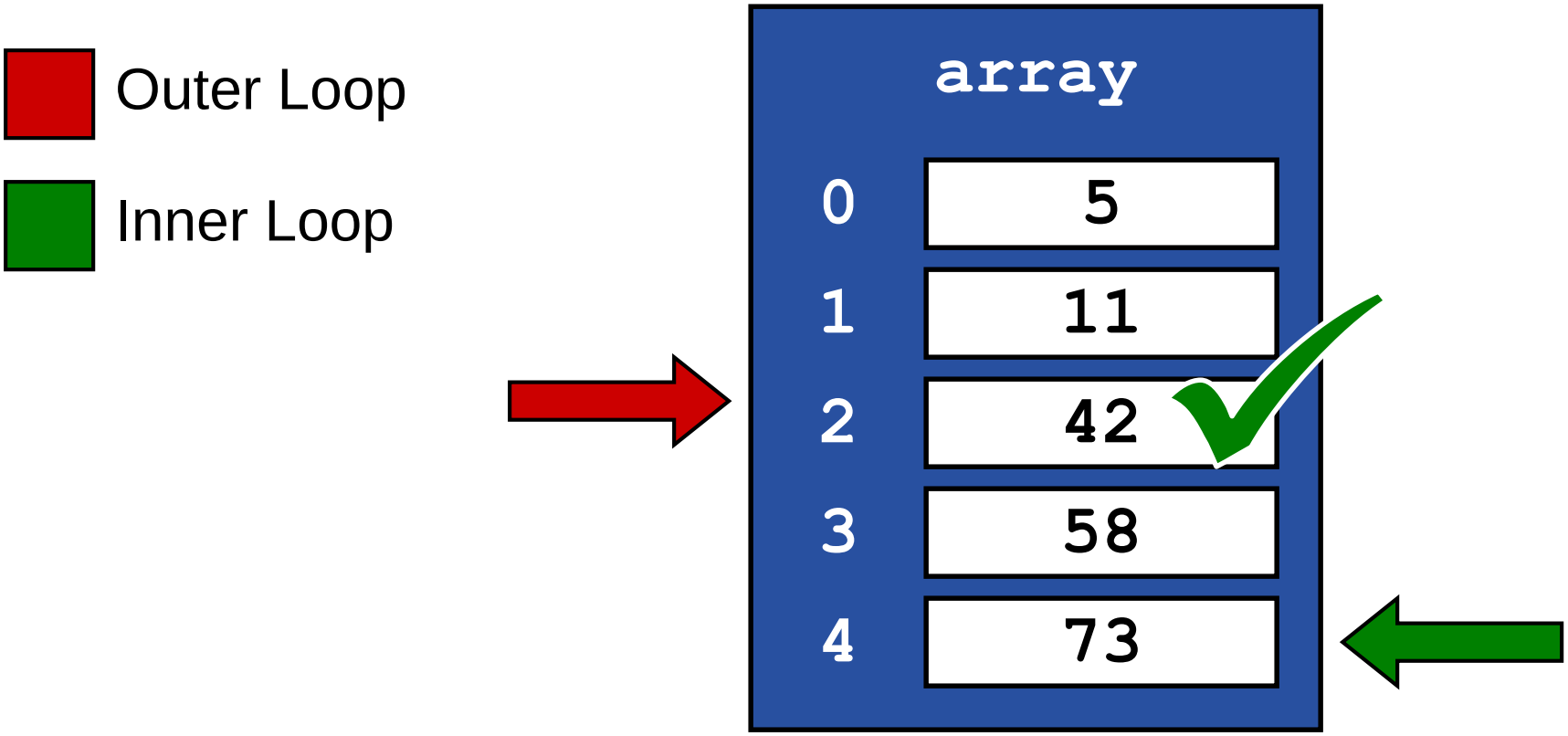
Selection Sort Example



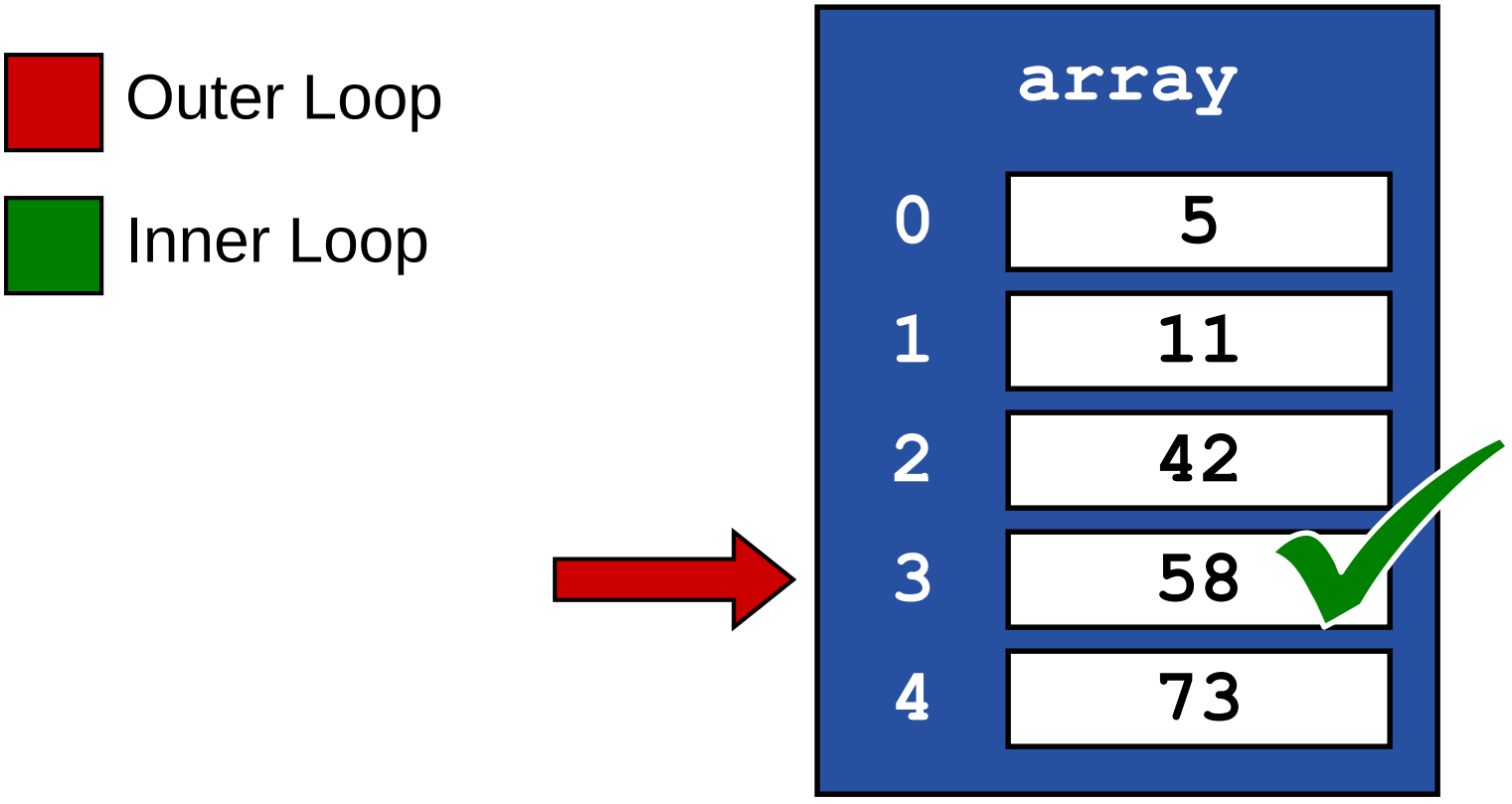
Selection Sort Example



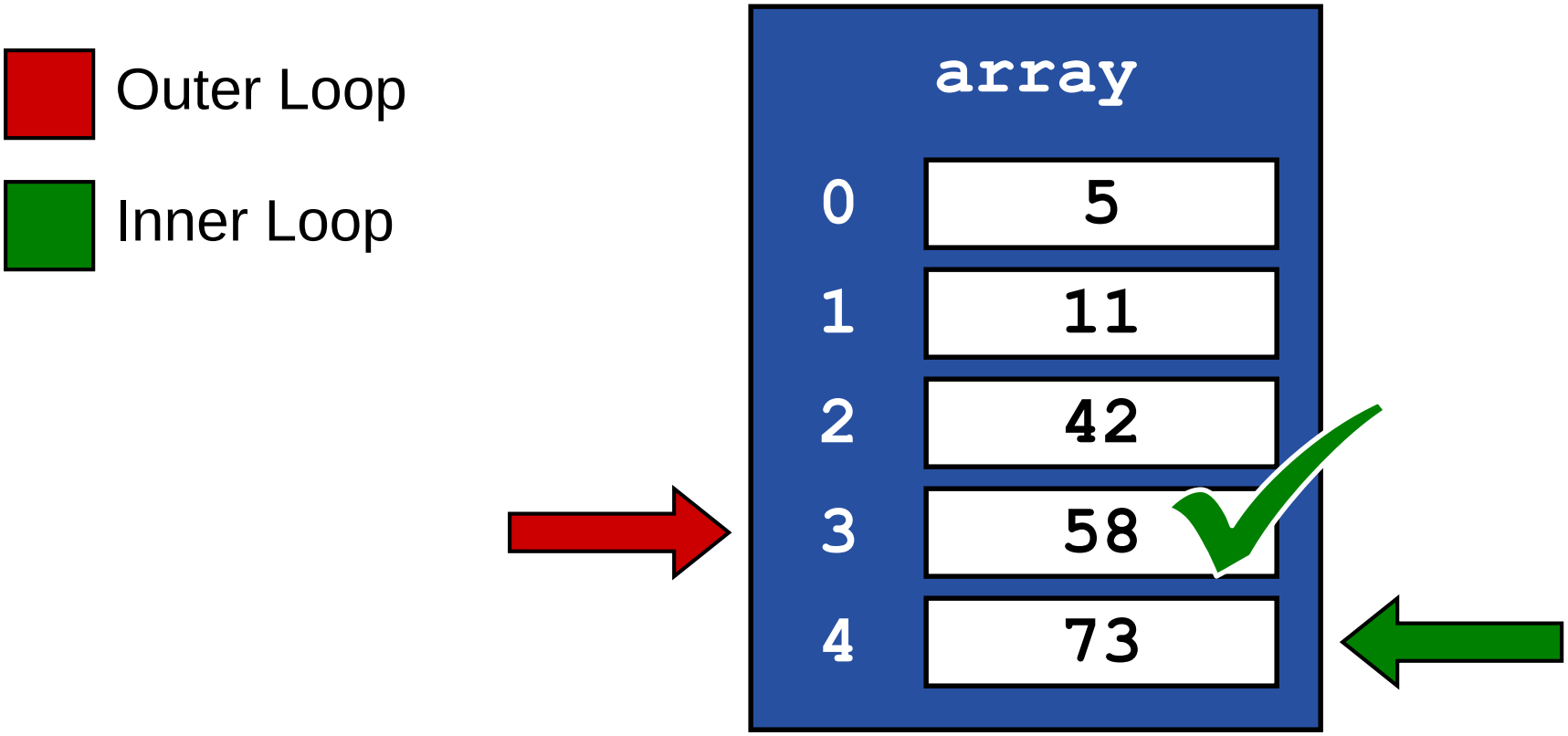
Selection Sort Example



Selection Sort Example



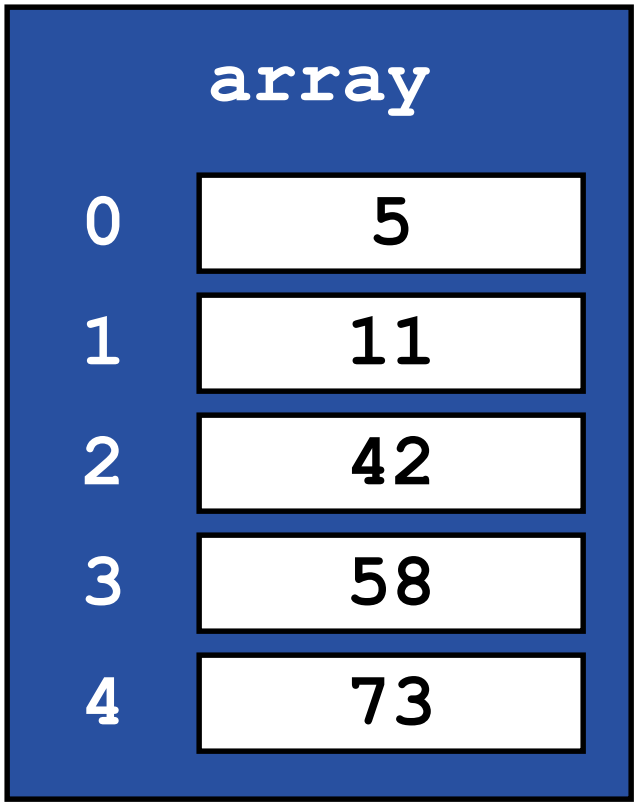
Selection Sort Example



Selection Sort Example

Outer Loop

Inner Loop



Insertion Sort

Chapter 9.3

Insertion Sort

- The *Insertion Sort* is sorting algorithm with several advantages over the Bubble and Selection
- Often, it is compared to sorting a deck of cards

Deck of Cards

- Let's say were asked to sort a row of cards… (and you start sorting from the left)
- You will find a card, move it, and shift the rest of the cards to the right
- So, you build a sorted list a bit at a time – on the left side

How it Works

- Consists of two loops – inner and outer
- A sorted list is constructed above the outer
- Outer loop moves down the list
  - current array value is temporarily removed (copied) from the array
  - moves down

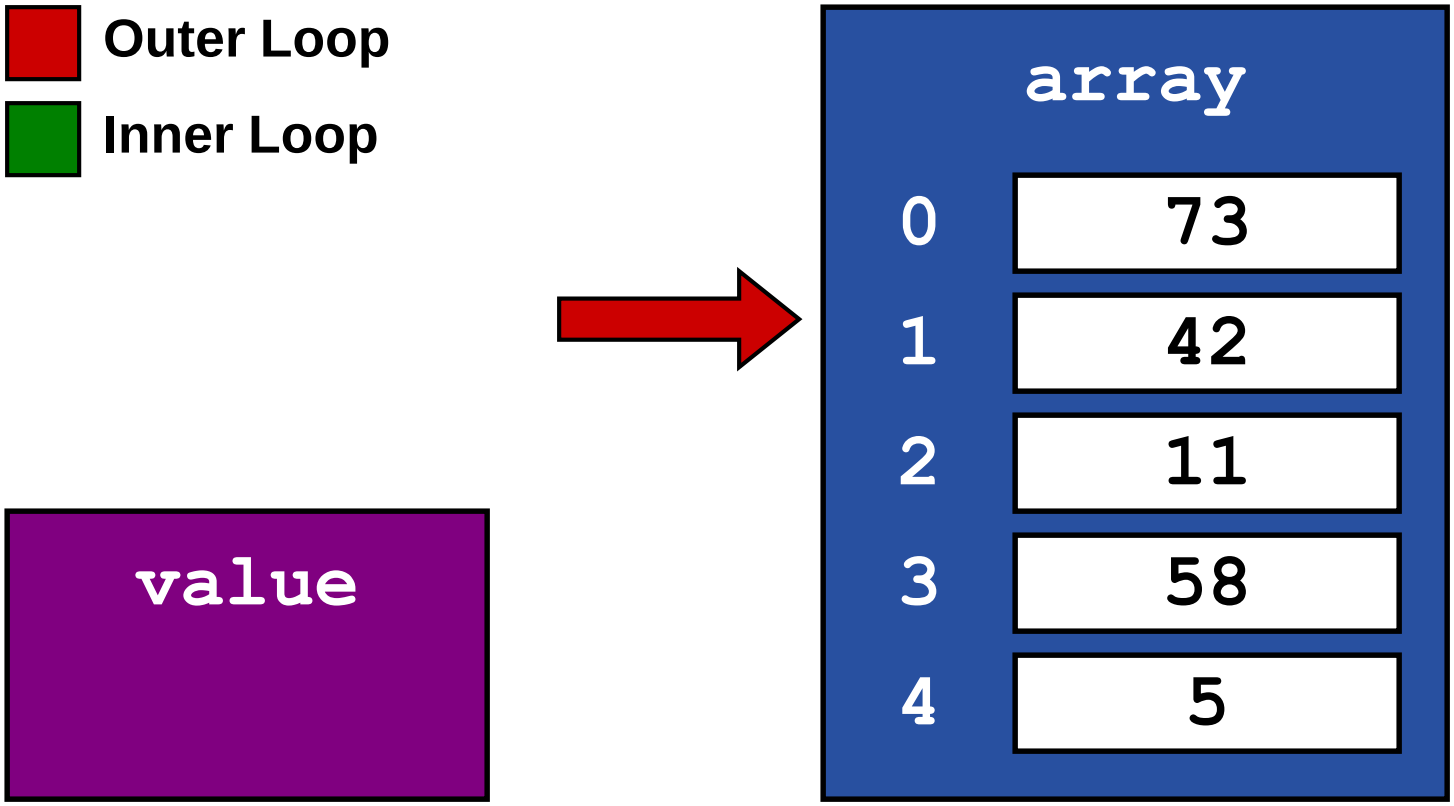
How it Works

- Inner loop
  - moves up from the current outer loop position
  - if the cell, being looked at, is larger than the saved value, it is moved down
  - so, the "cards" shift to make room for the saved cards proper position

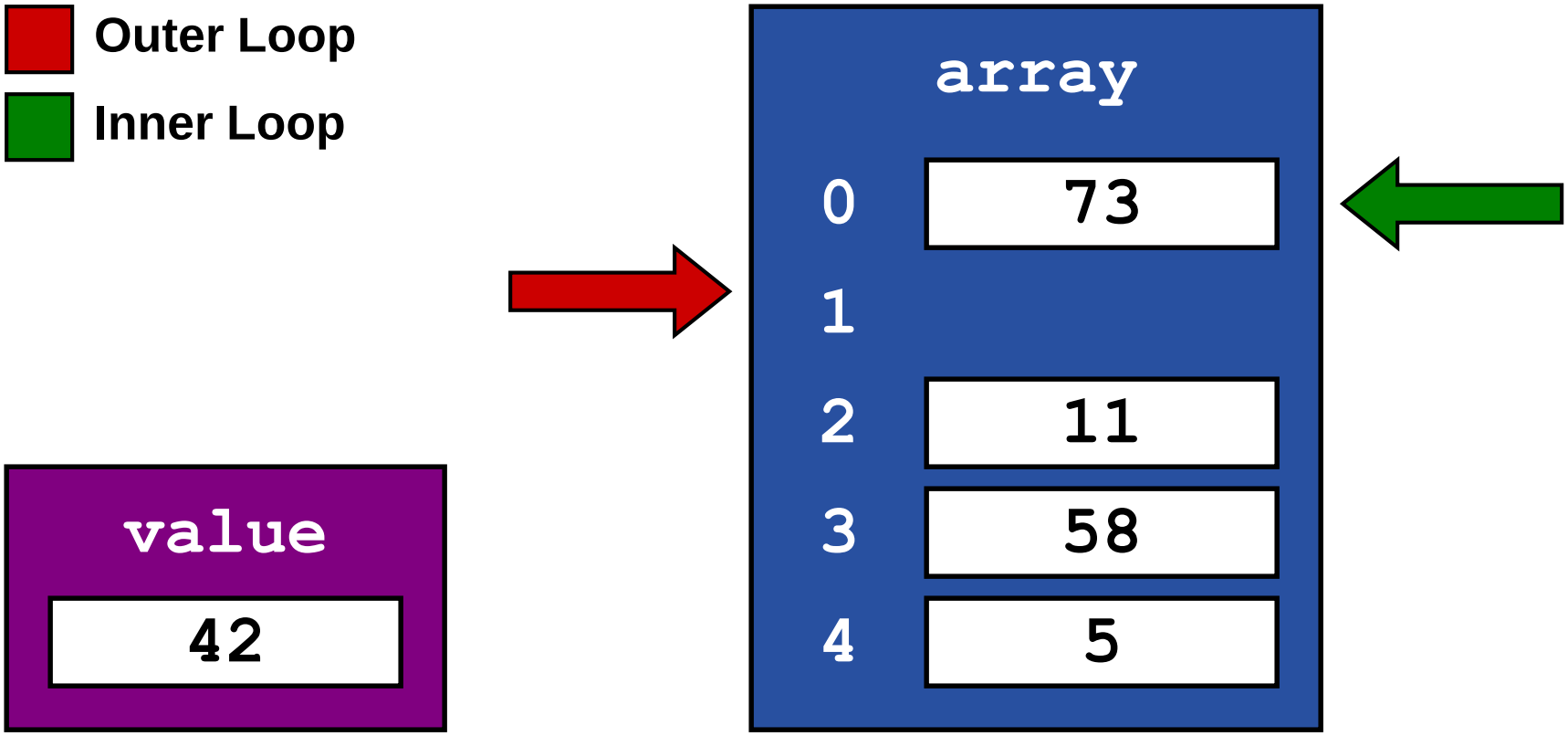
The Insertion Sort

```
For i = 1 to count-1
    Set value = array[i]

    Set j = i - 1;
    While j >= 0 && array[j] > value
        array[j + 1] = array[j]
        Set j = j - 1
    End While

    Set array[j + 1] = value
End For
```

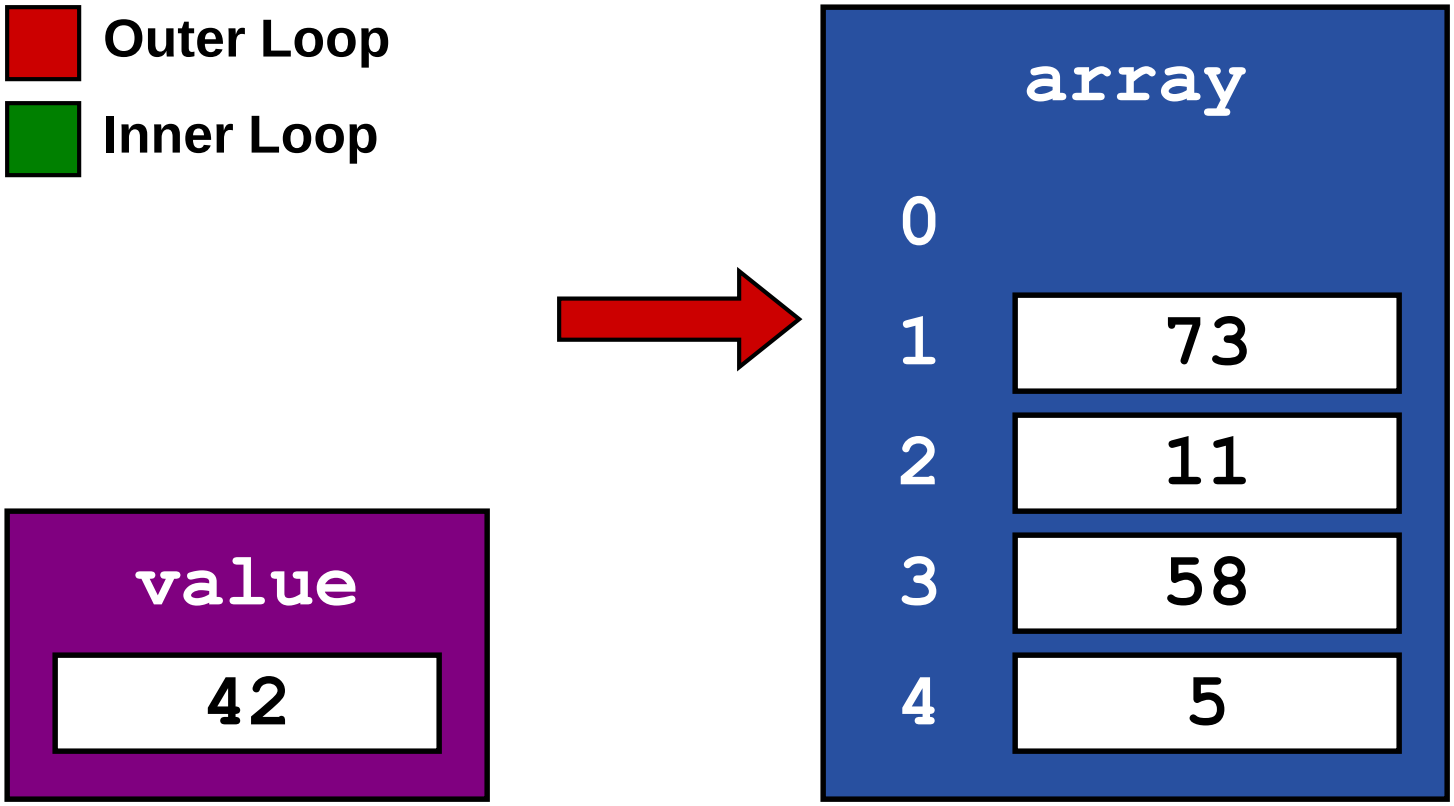
Insertion Sort Example



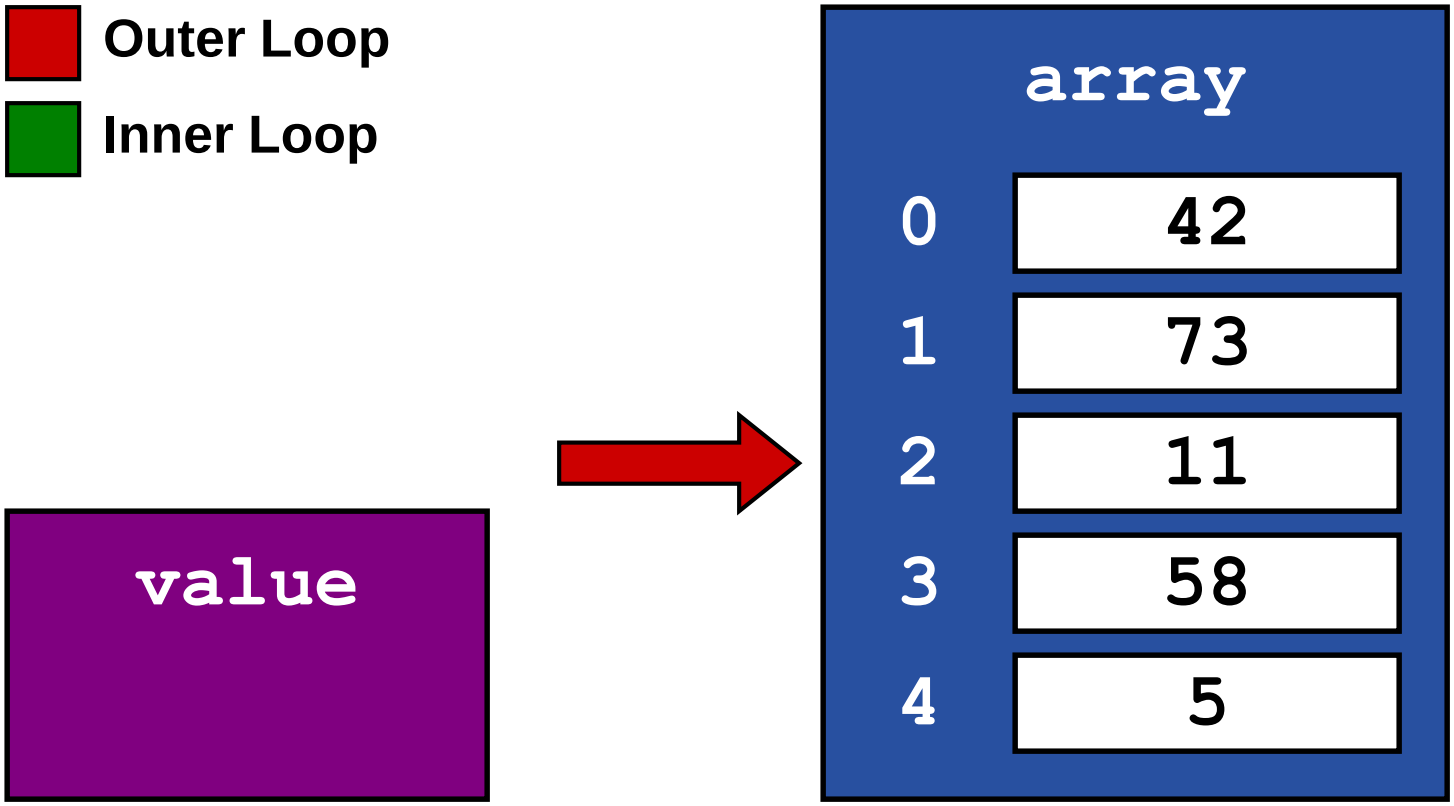
Insertion Sort Example



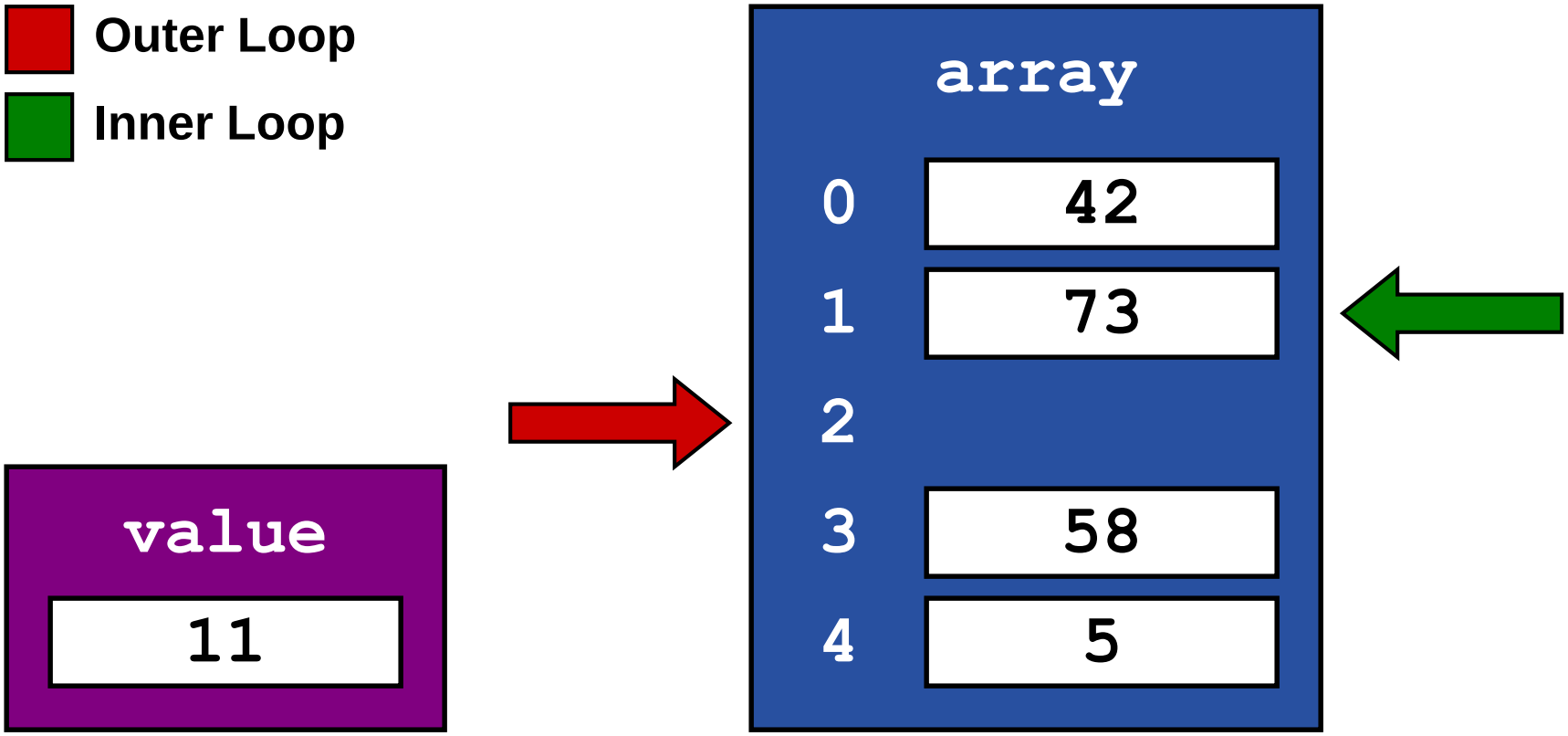
Insertion Sort Example



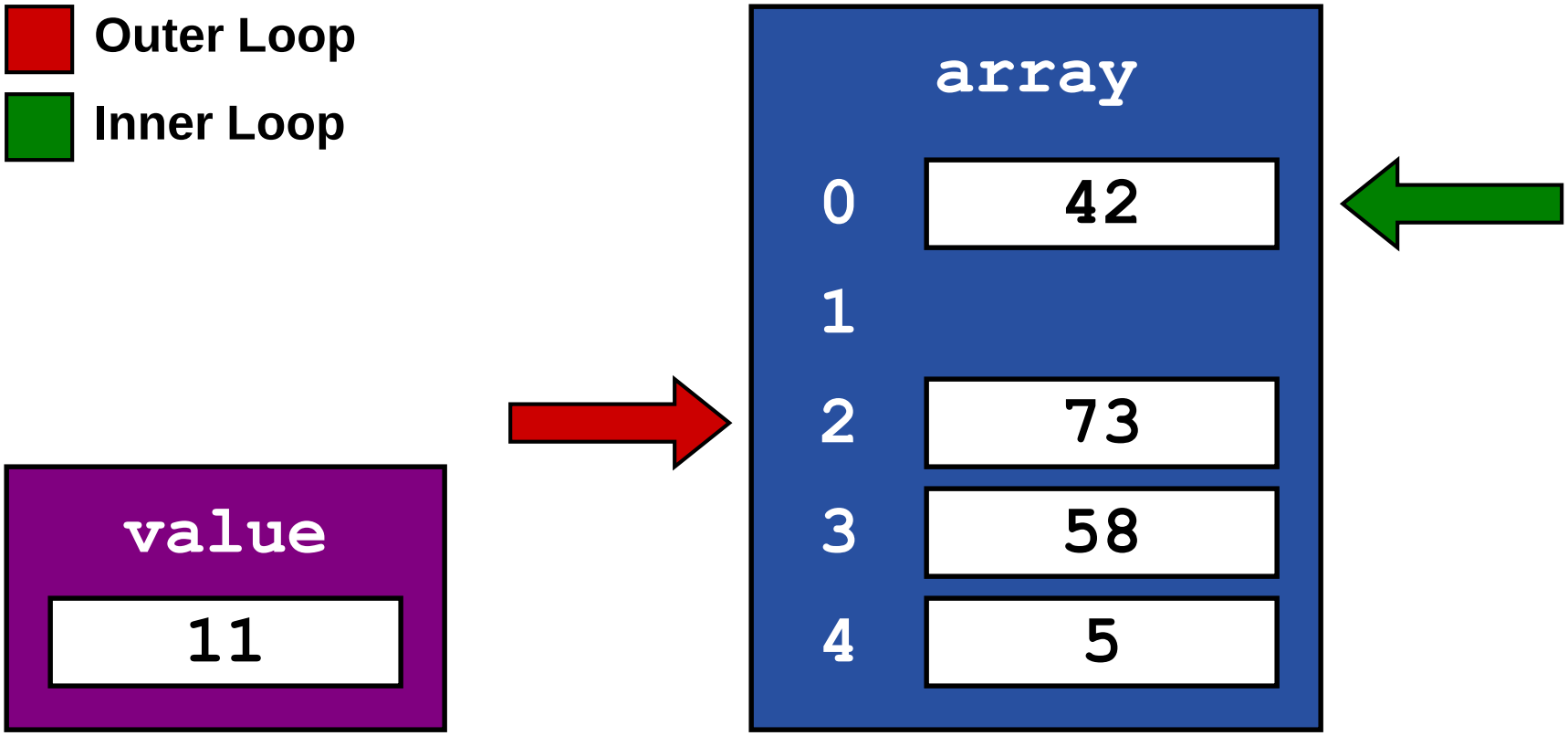
Insertion Sort Example



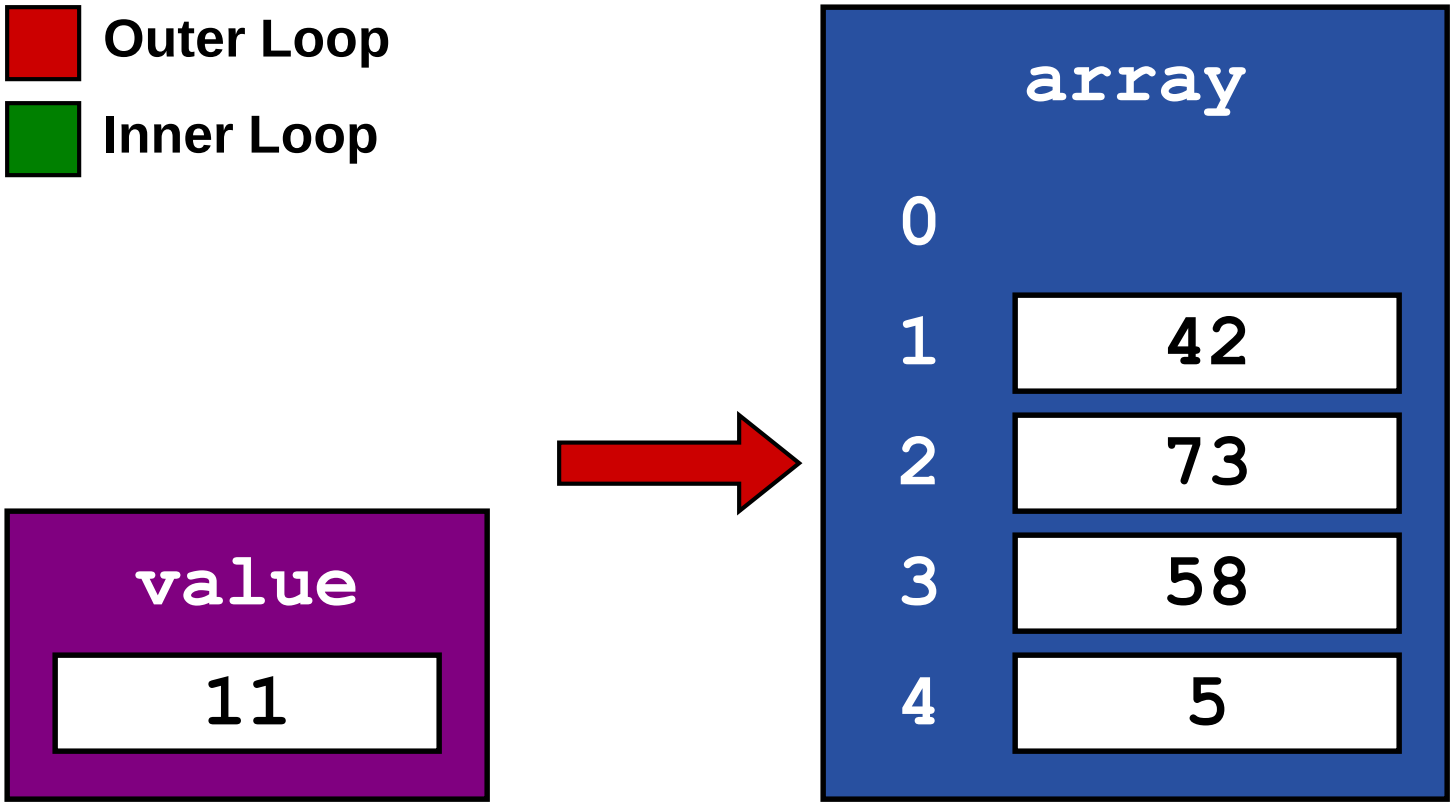
Insertion Sort Example



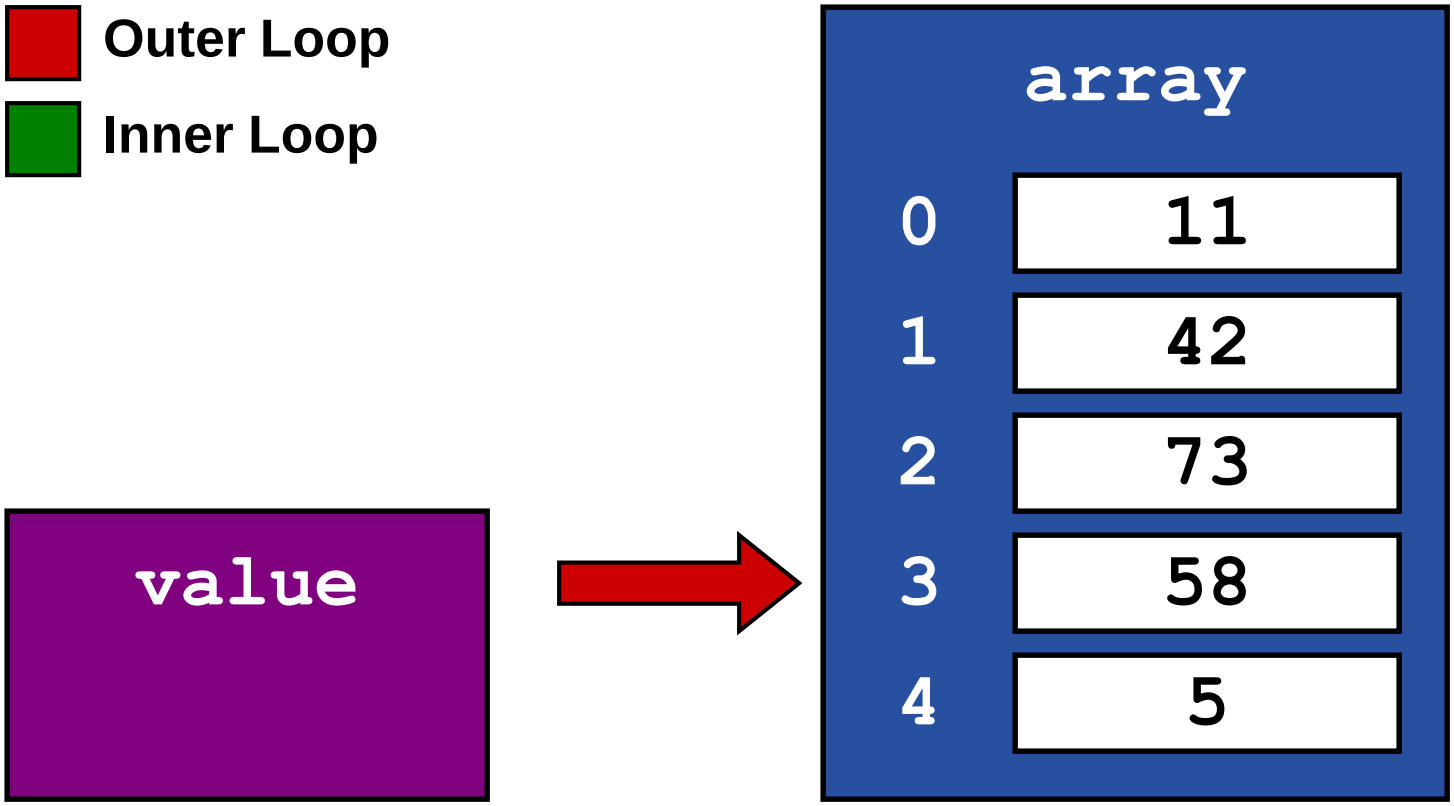
Insertion Sort Example



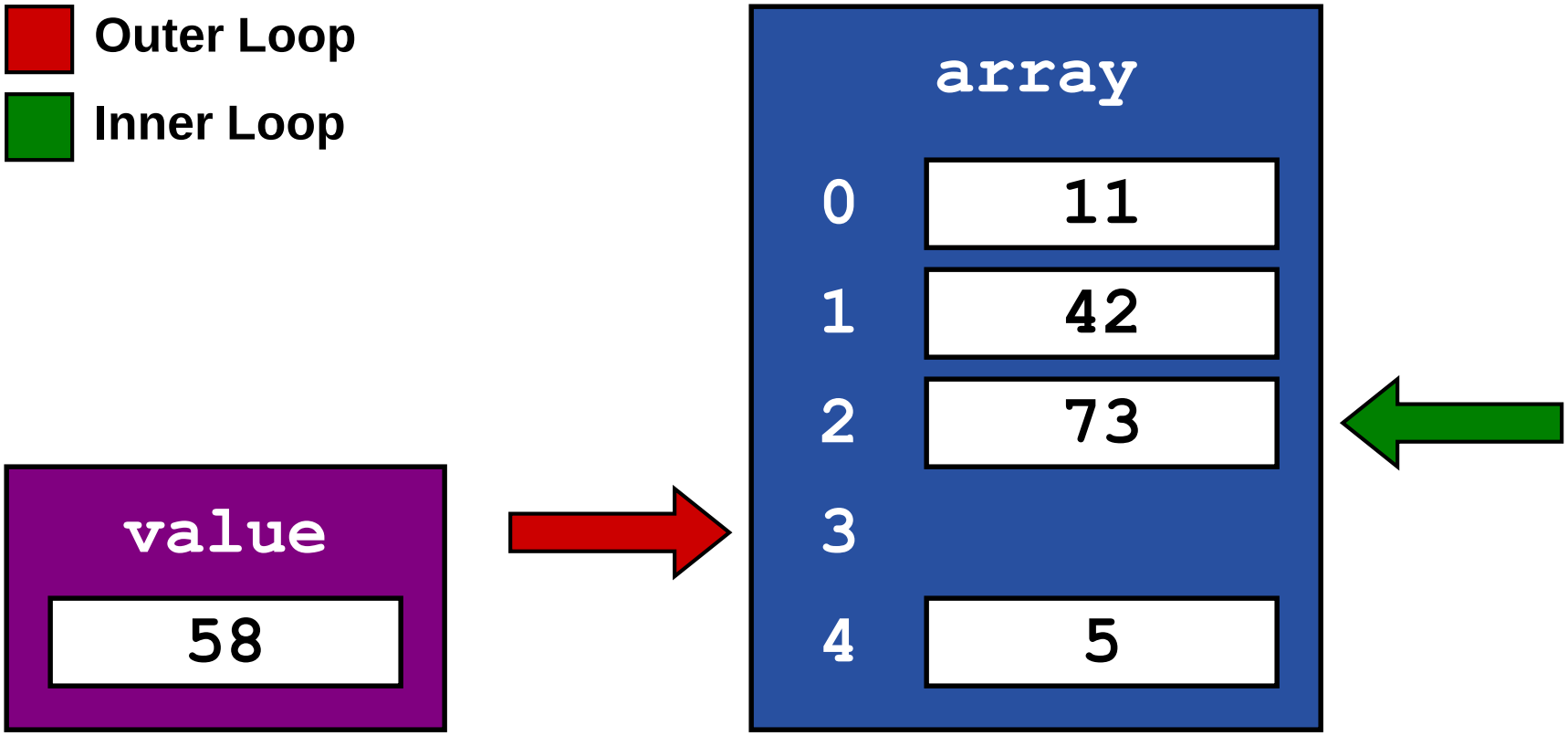
Insertion Sort Example



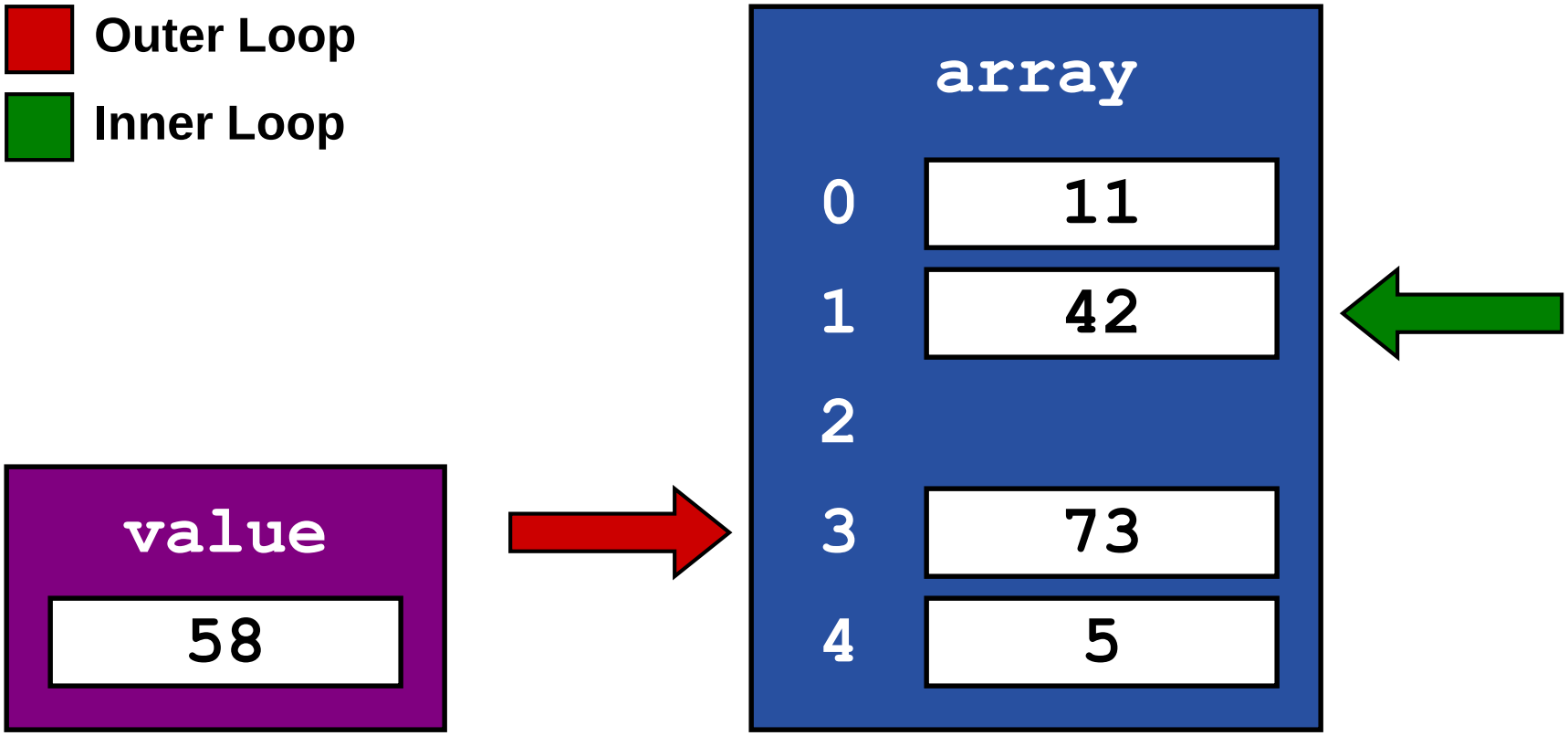
Insertion Sort Example



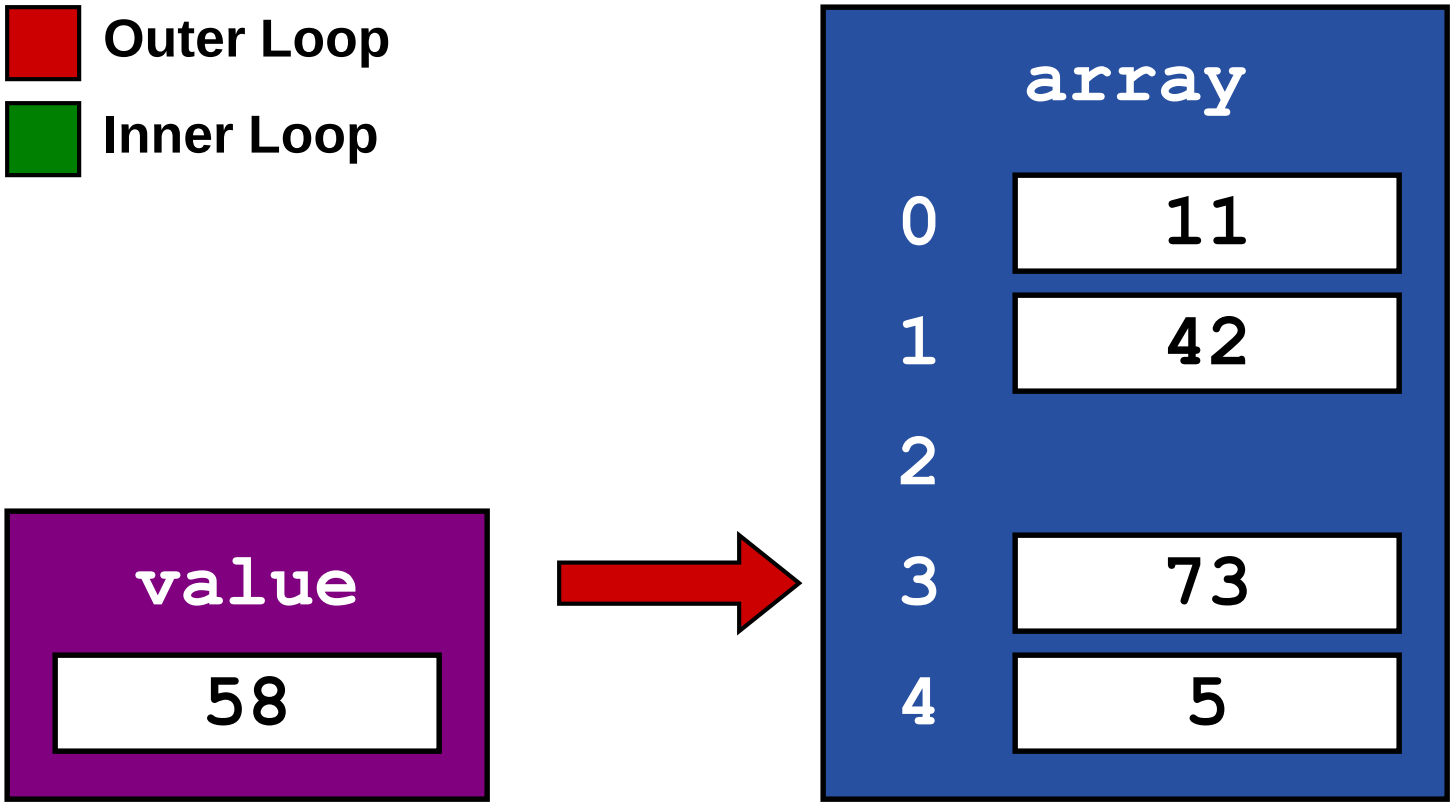
Insertion Sort Example



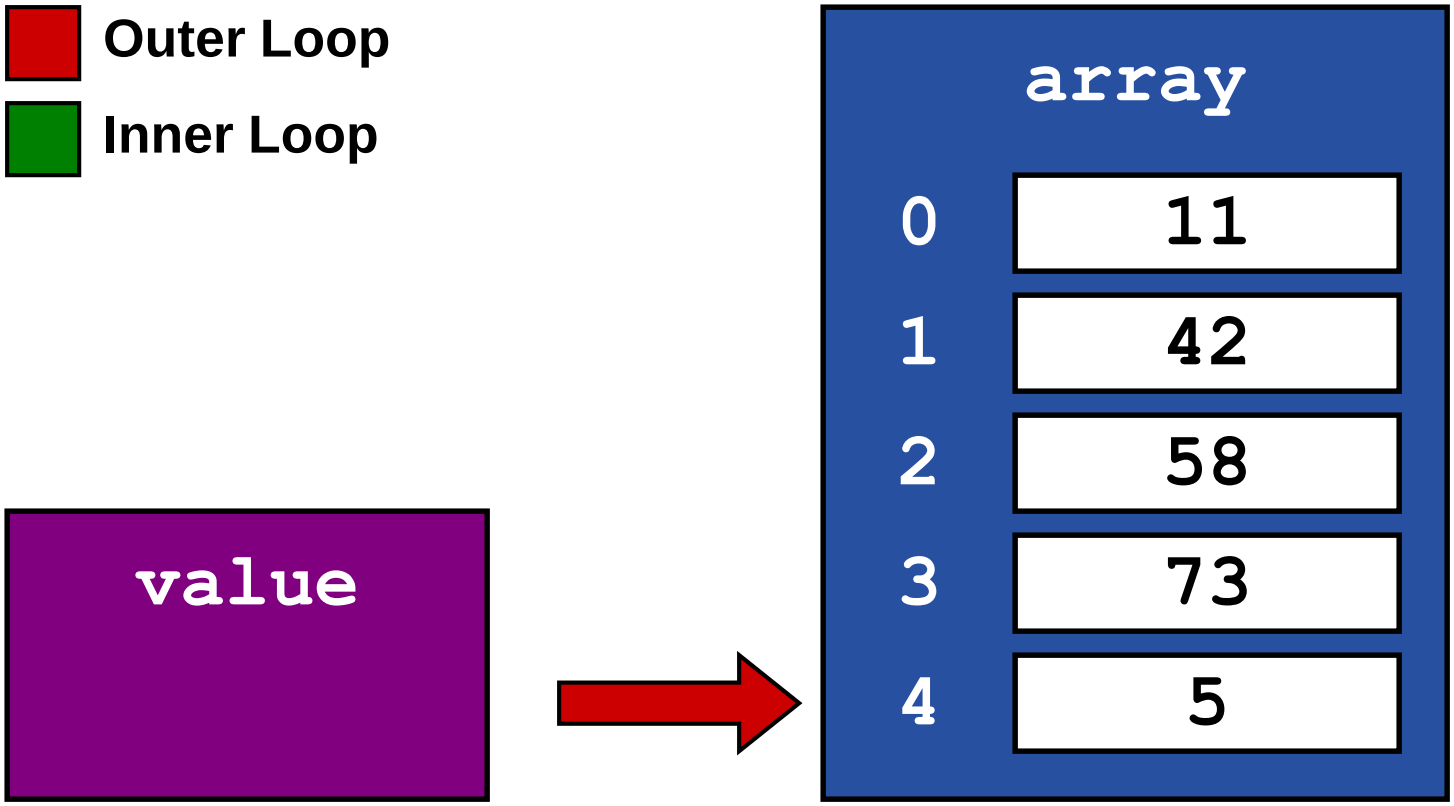
Insertion Sort Example



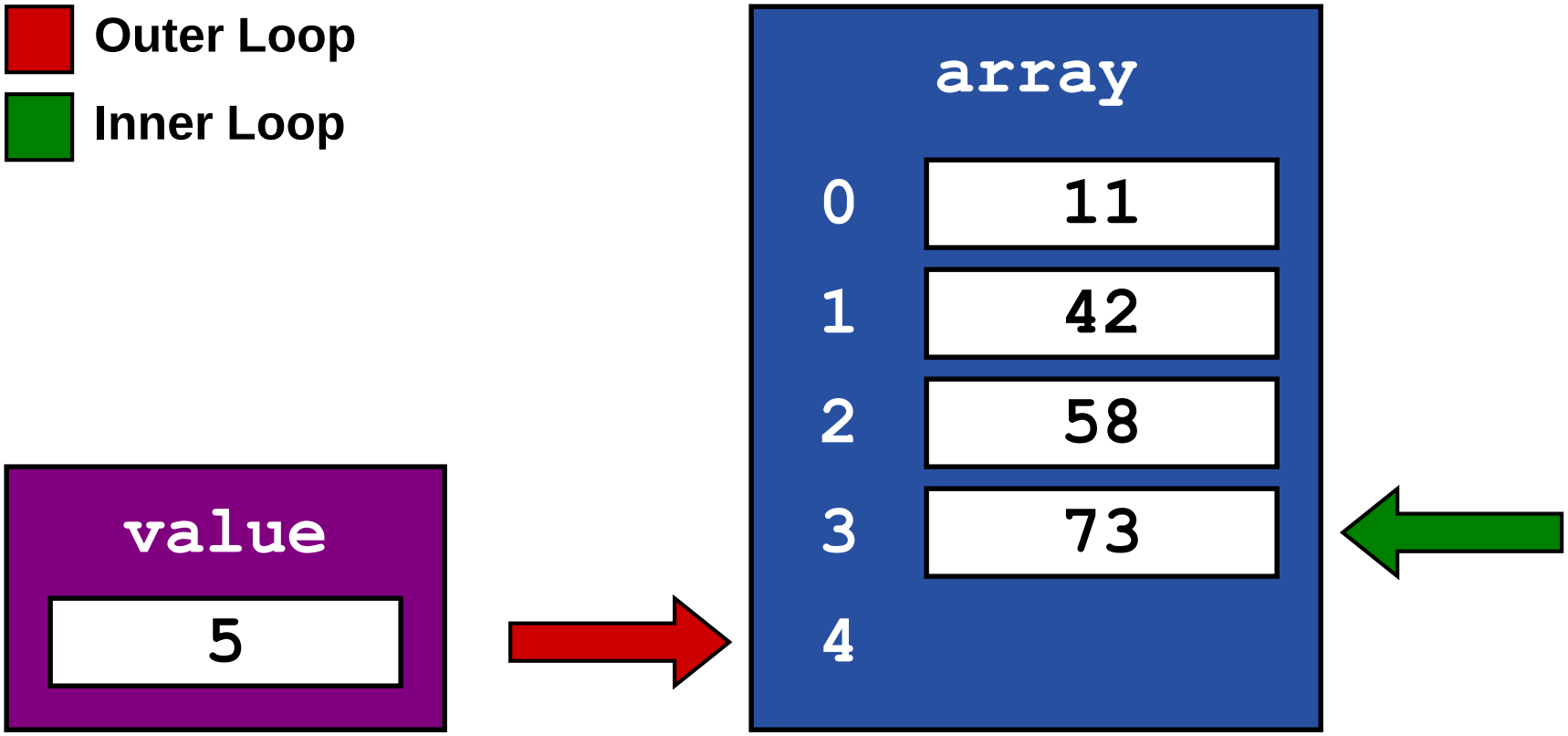
Insertion Sort Example



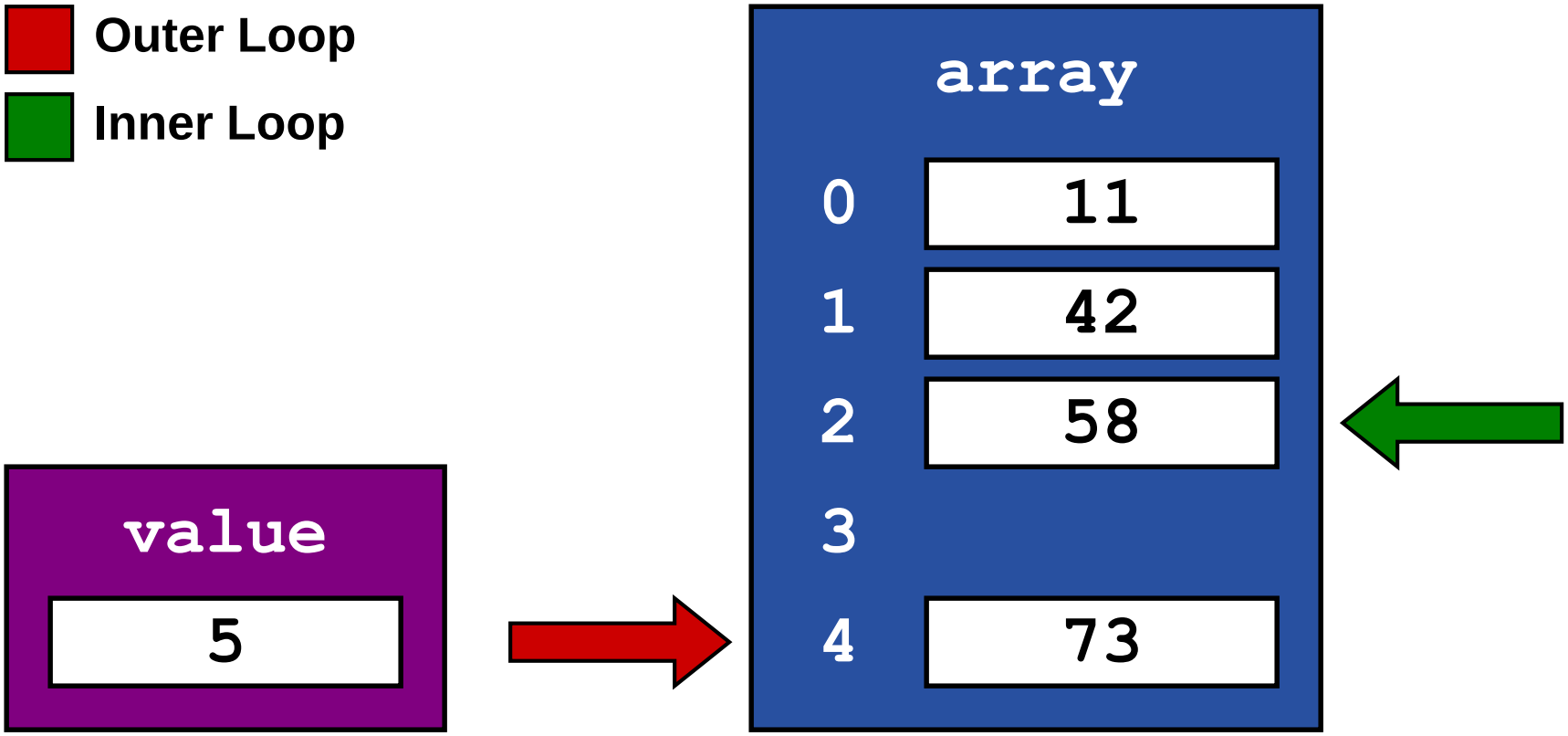
Insertion Sort Example



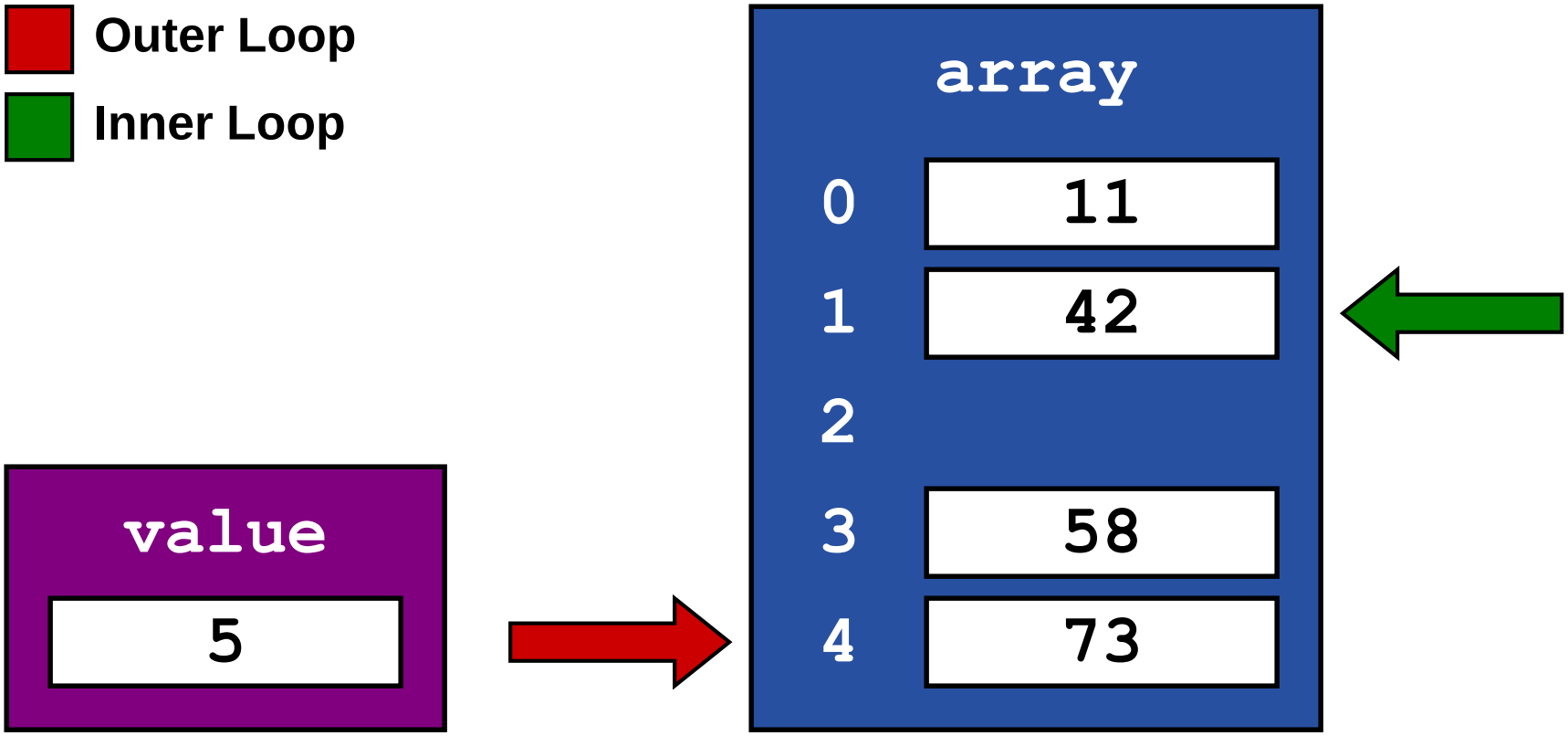
Insertion Sort Example



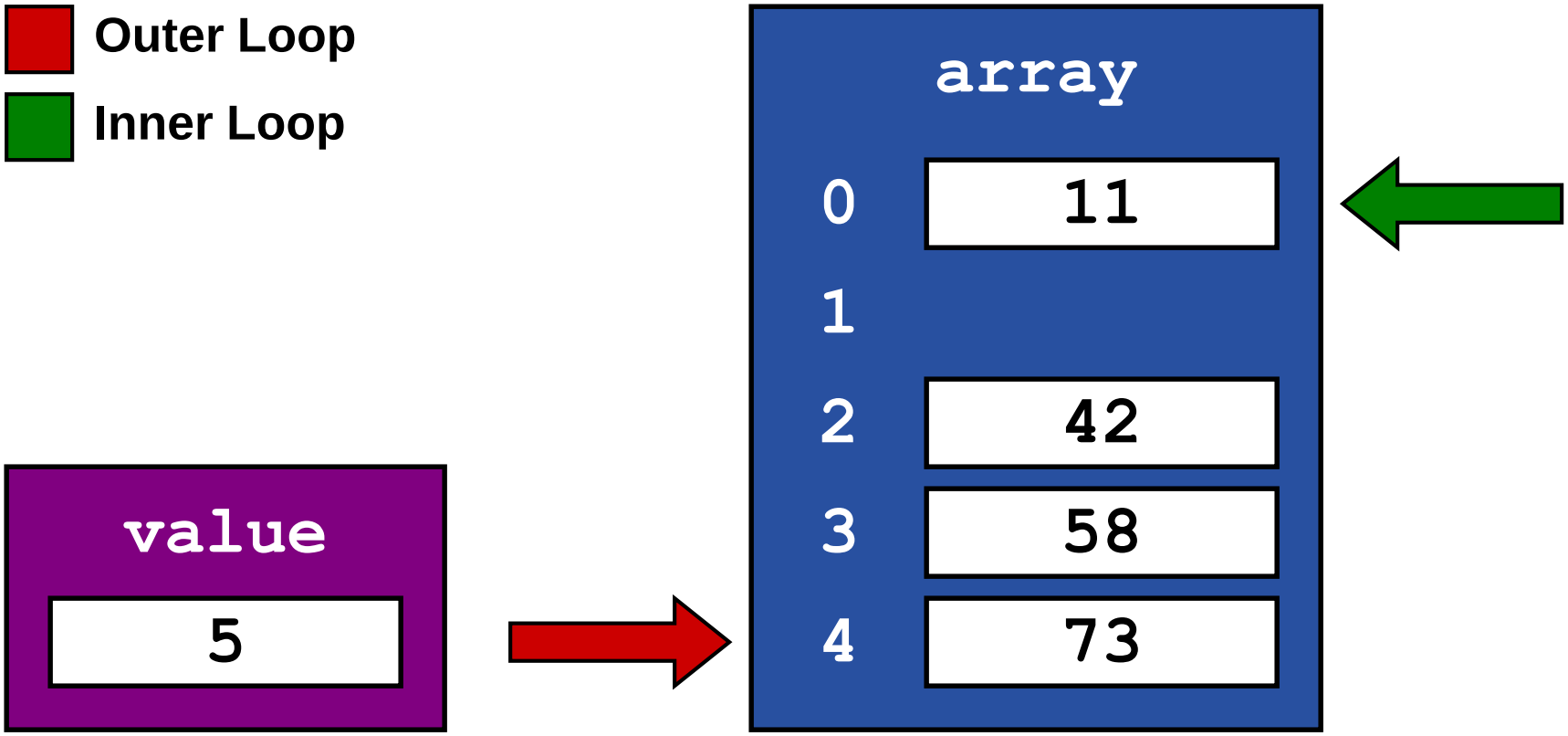
Insertion Sort Example



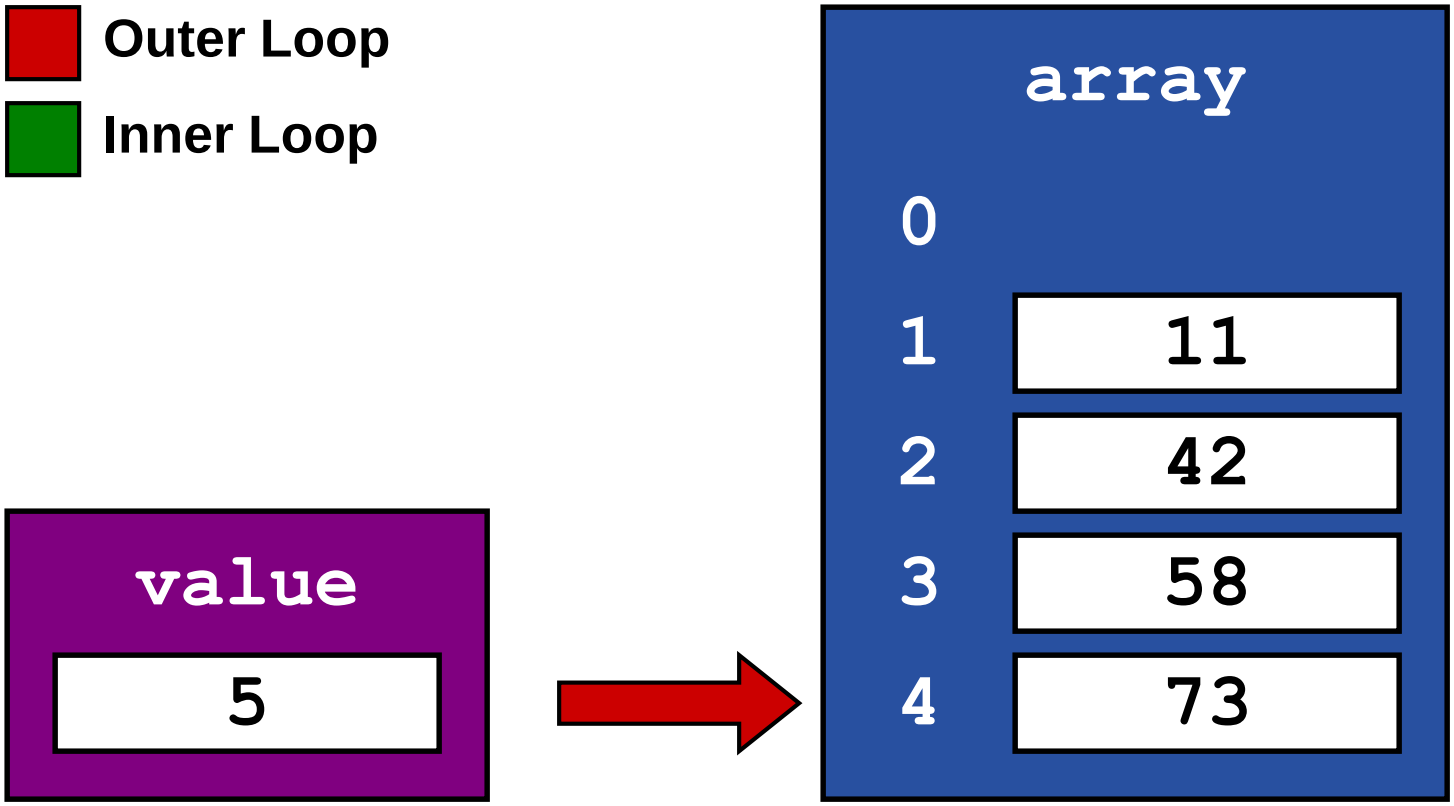
Insertion Sort Example



Insertion Sort Example



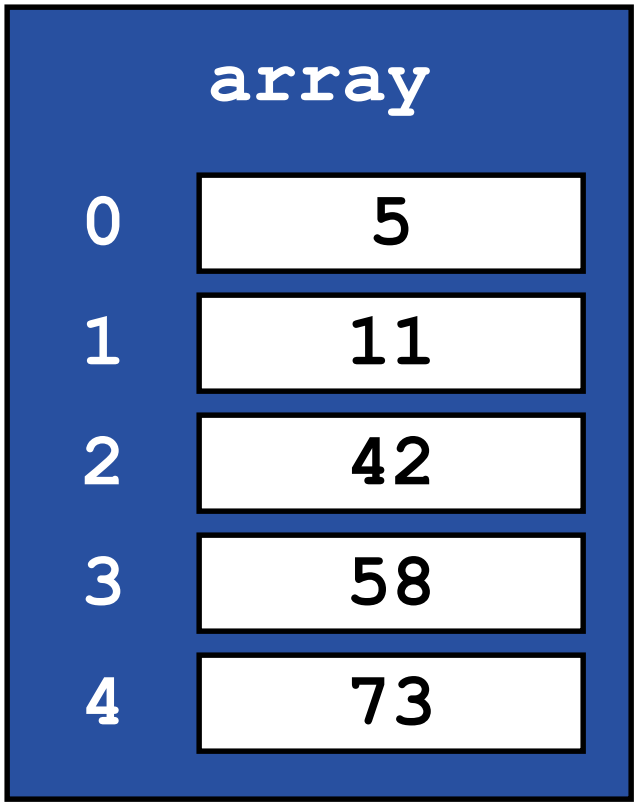
Insertion Sort Example



Insertion Sort Example

Outer Loop

Inner Loop



Binary Search

Chapter 9.4

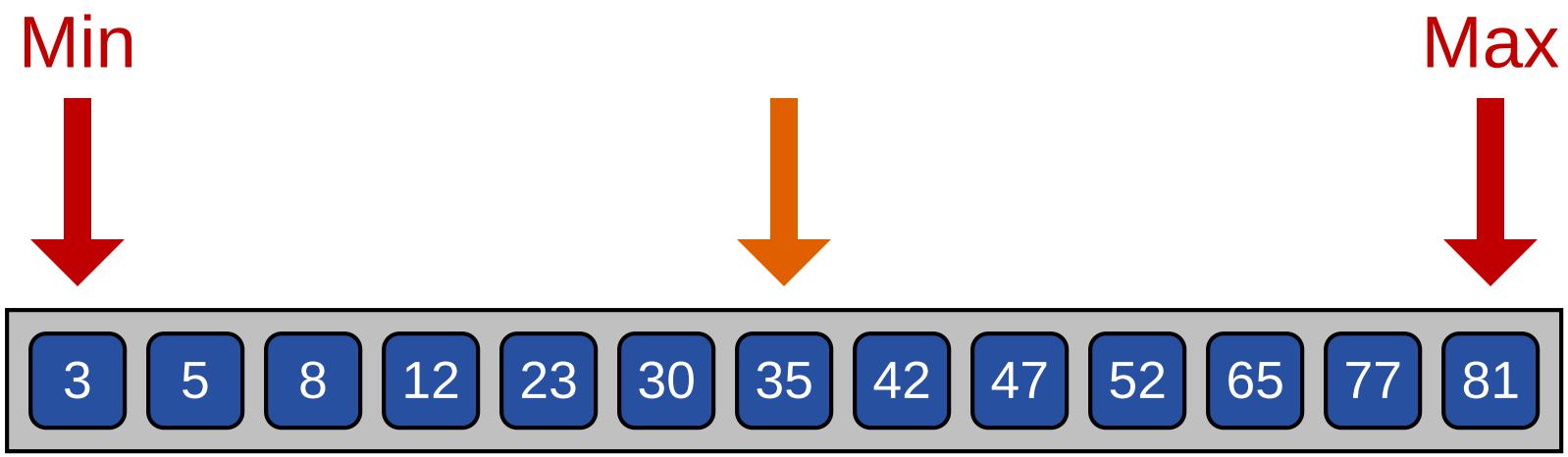
Binary Searching

- A *binary search* is an fast and efficient way to search an array
- Algorithm works like the classic "secret number game"
- Requires that the array is sorted before the search

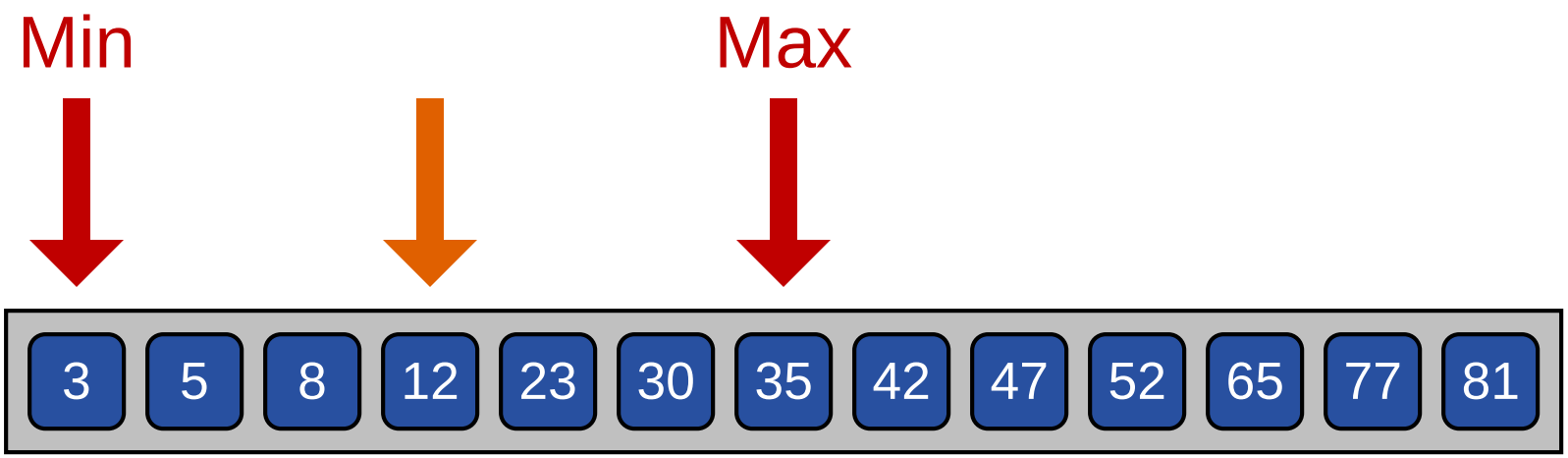
How it Works

- Starts knowing the max & min values
  - in the case of arrays, this is the min and max index
  - in the number game, it is the min and max value
- Algorithm continues
  - it looks at the midpoint between the first and last
  - if the value > target, the max is set to the midpoint
  - if the value < target, the min is set to the midpoint
  - *this eliminates half of the numbers each iteration*

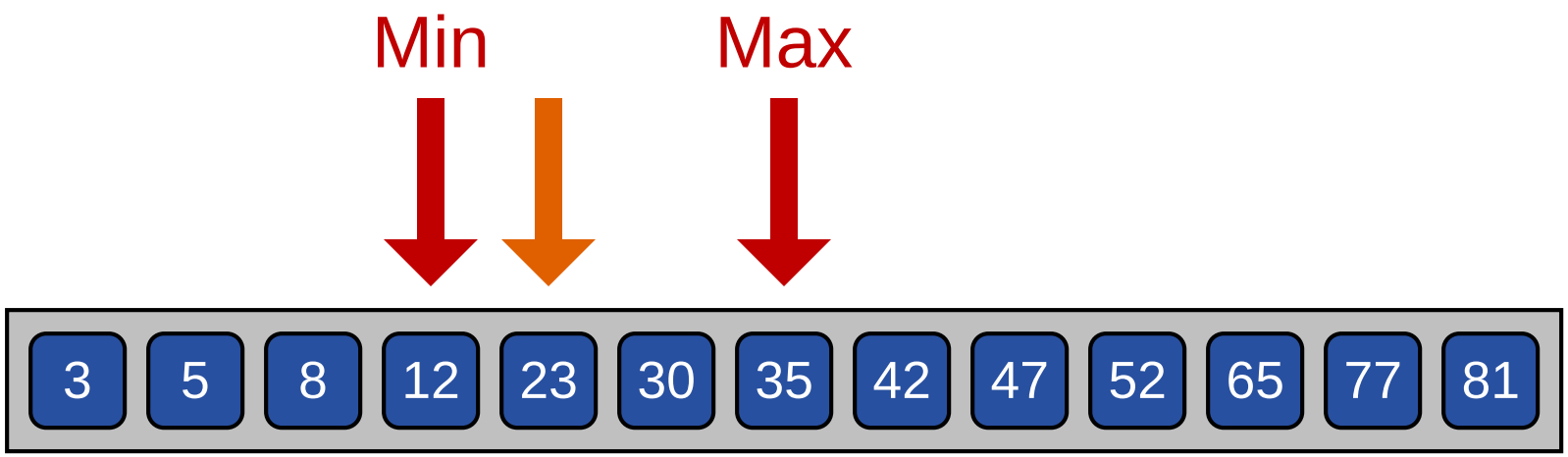
Binary Example: Find 30



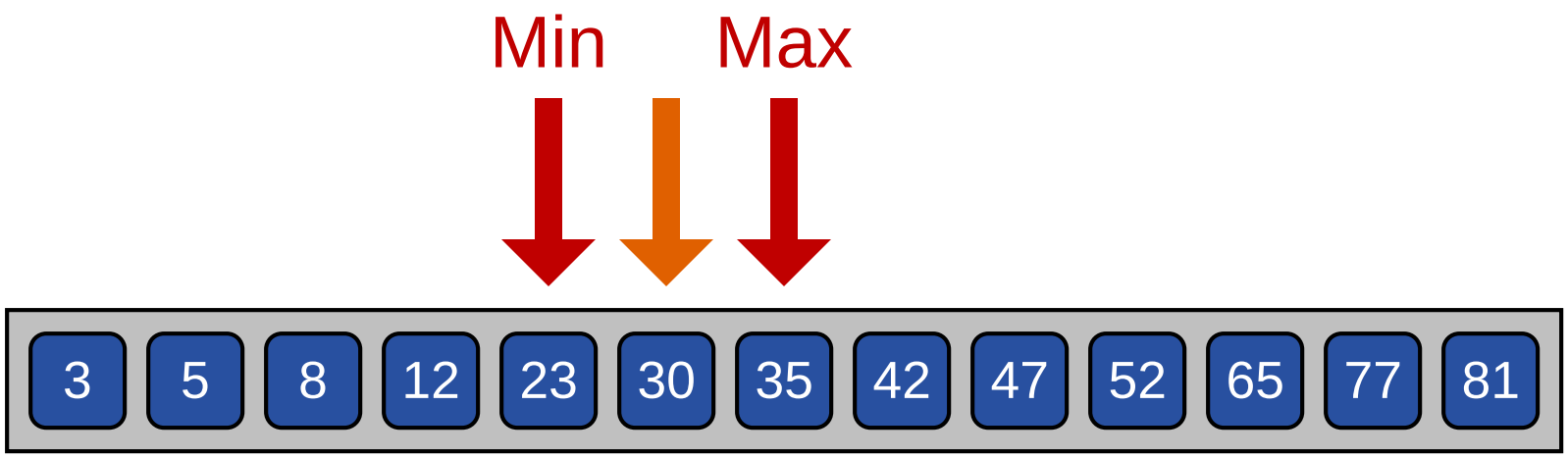
Binary Example: Find 30



Binary Example: Find 30



Binary Example: Find 30



Benefits

- The binary search is incredibly efficient and absolutely necessary for large arrays
- Any item can be found only $\log_2(n)$ searches!
- However, since array must be sorted, sorting algorithms are equally vital

Maximum # of Searches

Array Size	Sequential Search	Binary Search
10	10	4
100	100	7
1,000	1,000	10
10,000	10,000	14
100,000	100,000	17
1,000,000	1,000,000	20
10,000,000	10,000,000	24
100,000,000	100,000,000	27
1,000,000,000	1,000,000,000	30