**Insertion Sort**

* You can use the insertion sort strategy to sort items that reside in an array.

**Two Partitions**

* Insertion sort partitions the array into two regions:

1. **Sorted**

and

1. **Unsorted**

Diagram

Description automatically generated

* Initially, the entire array is the unsorted region.
* At each step, the insertion sort algorithm takes the first item of the unsorted region and places it into its correct position in the sorted region.
* The fact that the items in the sorted region are ***sorted among themselves*** is an assertion of the algorithm.
* Because at each step the size of the sorted region grows by 1 and the size of the unsorted region shrinks by 1, the entire array will be sorted when the algorithm terminates.

**Algorithm**

To sort an array of size n in ascending order:

1. Iterate from arr[1] to arr[n-1] over the array.
   1. Save the current elements value (key).
   2. While the current elements value (key) is smaller than its predecessor, swap the predecessor element’s (greater) value with the current element (up one position).
   3. Continue to compare the saved key to its predecessors until either you hit the beginning of the array or a predecessor has a value that is not smaller than the saved key.

* Why sstart the loop at arr[1]?
* The first step is trivial: Moving theArray[0] from the unsorted region to the sorted region really does not require moving data.
* Therefore, you can omit this first step by considering
  + the initial sorted region to be theArray[0]

and

* + the initial unsorted region to be theArray[1..n-1]

**Example Walkthrough**

* Figure 11-4 illustrates an insertion sort of an array of five integers.

1. Initially, the sorted region is theArray[0], which is 4, and the unsorted region is the rest of the array.
2. You take the first item in the unsorted region—the 3—and insert it into its proper position in the sorted region.
3. This insertion requires you to shift array entries to make room for the inserted item.
4. You then take the first item in the new unsorted region—the 2—and insert it into its proper position in the sorted region.
5. We repeat this until reaching the end of the array.



**Pseudocode**

insertionSort(array)

for each unsorted element i (arr[1] to arr[size-1])

save the current value of i as a key

j is an iteration variable for comparing key to predecessors

while j is greater than 0 (don’t want out of bounds) and arr[j – 1] > key (predecessor is greater than starting key value)

arr[j] (current index) = arr[j-1] (predecessor)

move j back one to compare to other previous predecessors

insert key into proper position at j

end insertionSort