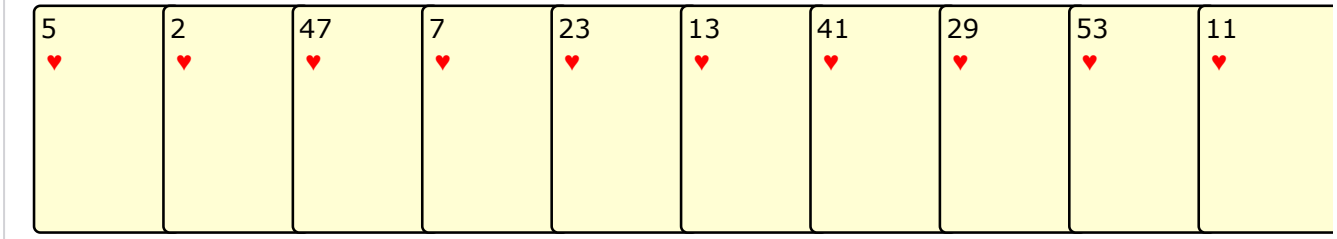
Insertion Sort

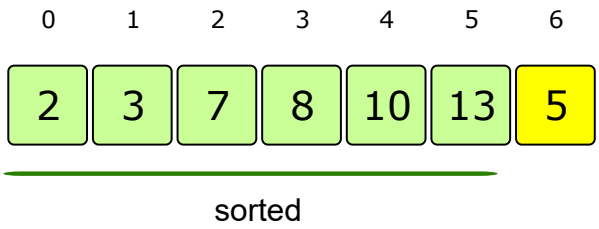
There are many different ways to sort. As selection sort runs, the subarray at the beginning of the array is sorted, but the subarray at the end is not. Selection sort scans the unsorted subarray for the next element to include in the sorted subarray.

Here's another way to think about sorting. Imagine that you are playing a card game. You're holding the cards in your hand, and these cards are sorted. The dealer hands you exactly one new card. You have to put it into the correct place so that the cards you're holding are still sorted. In selection sort, each element that you add to the sorted subarray is no smaller than the element already in the sorted subarray. But in our card example, the new card could be smaller than some of the cards you're already holding, and so you go down the line, comparing the new card against each card in your hand, until you find the place to put it. You insert the new card in the right place, and once again, your hand holds fully sorted cards. Then the dealer gives you another card, and you repeat the same procedure. Then another card, and another card, and so on, until the dealer stops giving you cards.

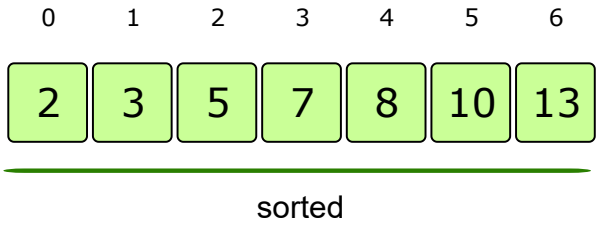
This is the idea behind insertion sort. Loop over positions in the array, starting with index 1. Each new position is like the new card handed to you by the dealer, and you need to insert it into the correct place in the sorted subarray to the left of that position. Here's a visualization that steps through that:



In terms of arrays, imagine that the subarray from index 0 through index 55 is already sorted, and we want to insert the element currently in index 6 into this sorted subarray, so that the subarray from index 0 through index 6 is sorted. Here's how we start:

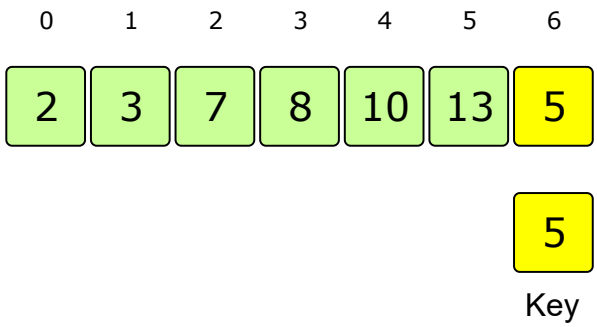


And here's what the subarray should look like when we're done:

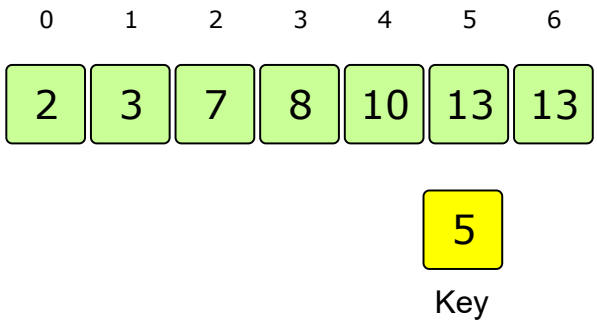


To insert the element in position 6 into the subarray to its left, we repeatedly compare it with elements to its left, going right to left. Let's call the element in position 6 the **key**. Each time we find that the key is less than an element to its

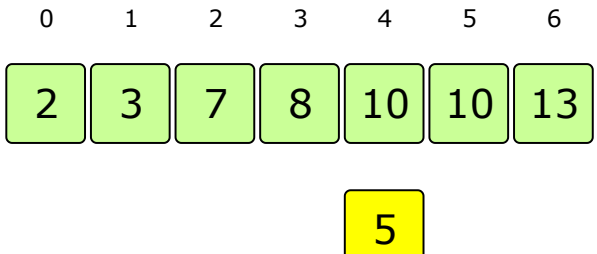
left, we slide that element one position to the right, since we know that the key will have to go to that element's left. We'll need to do two things to make this idea work: we need to have a **slide** operation that slides an element one position to the right, and we need to save the value of the key in a separate place (so that it doesn't get overridden by the element to its immediate left). In our example, let's pull the element at index 6 into a variable called **key**:



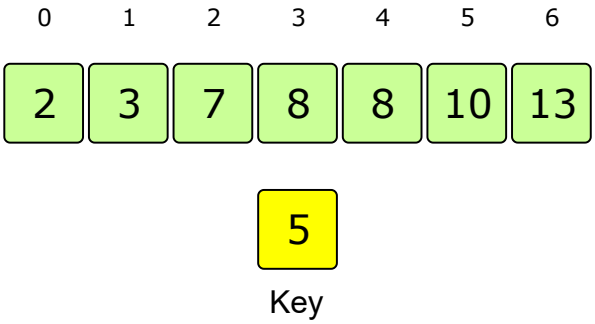
Now, we compare key with the element at position 5. We find that key (5) is less than the element at position 5 (13), and so we slide this element over to position 6:



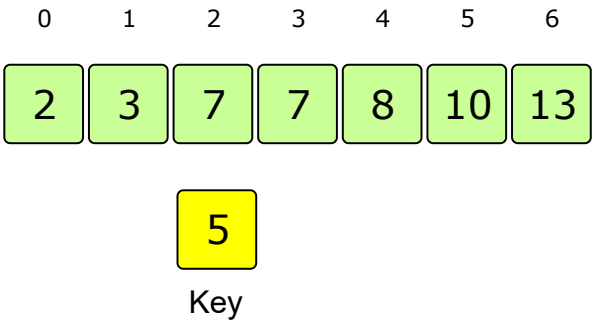
Notice that the slide operation just copies the element one position to the right. Next, we compare key with the element at position 4. We find that key (5) is less than the element at position 4 (10), and we slide this element over:



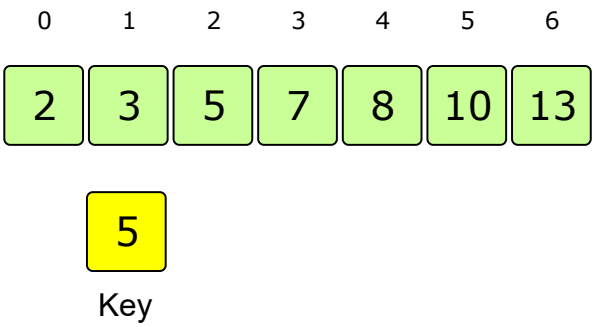
Next, we compare key with the element at position 3, and we slide this element over:



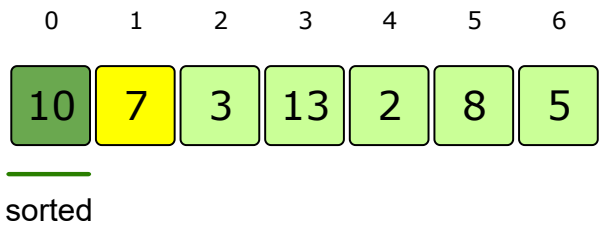
The same happens with the element at position 2:



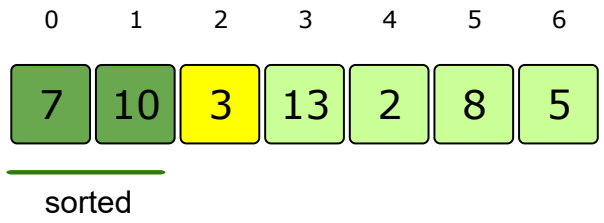
Now we come to the element at position 1, which has the value 3. This element is less than key, and so we do not slide it over. Instead, we drop key into the position immediately to the right of this element (that is, into position 2), whose element was most recently slid to the right. The result is that the subarray from index 0 through index 6 has become sorted:



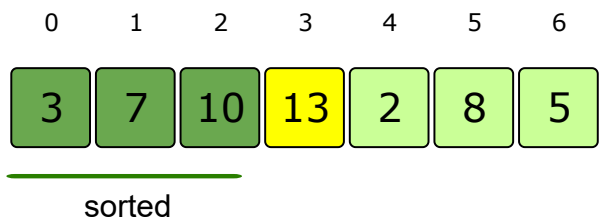
Insertion sort repeatedly inserts an element in the sorted subarray to its left. Initially, we can say that the subarray containing only index 0 is sorted, since it contains only one element, and how can a single element not be sorted with respect to itself? It must be sorted. Let's work through an example. Here's our initial array:



Because the subarray containing just index 0 is our initial sorted subarray, the first key is in index 1. (We'll show the sorted subarray in red, the key in yellow, and the part of the array that we have yet to deal with in blue.) We insert the key into the sorted subarray to its left:

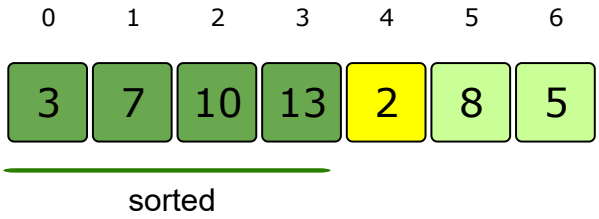


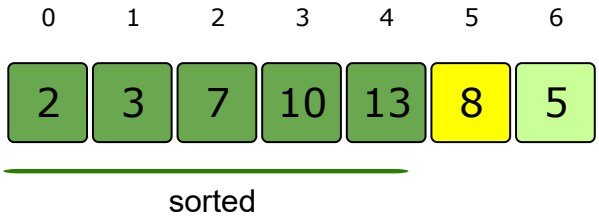
Now the sorted subarray runs from index 0 through index 1, and the new key is in index 2. We insert it into the sorted subarray to its left:

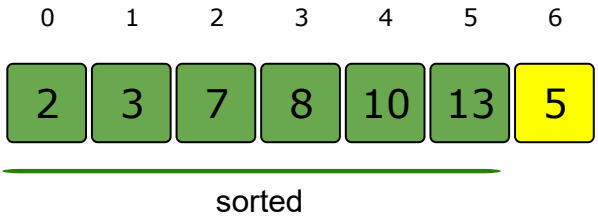


We keep going, considering each array element in turn as the key, and

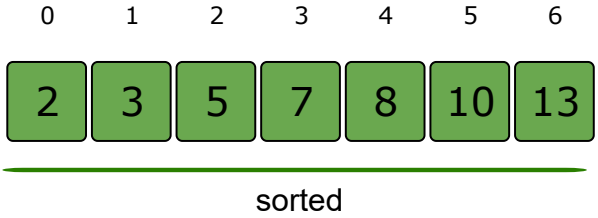
inserting it into the sorted subarray to its left:







Once we've inserted that rightmost element in the array, we have sorted the entire array:

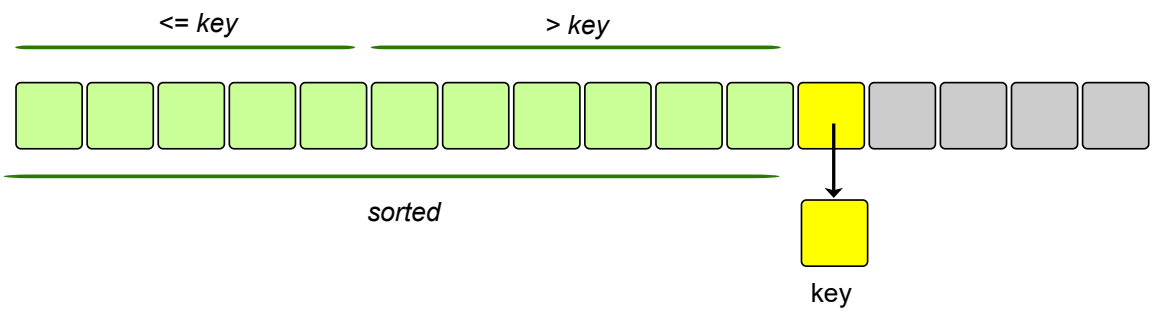


A couple of situations that came up in our example bear a little more scrutiny: when the key being inserted is less than all elements to its left (as when we inserted keys 2 and 3), and when it's greater than or equal to all elements to its left (as when we inserted key 13). In the former case, every element in the subarray to the left of the key slides one position to the right, and we have to

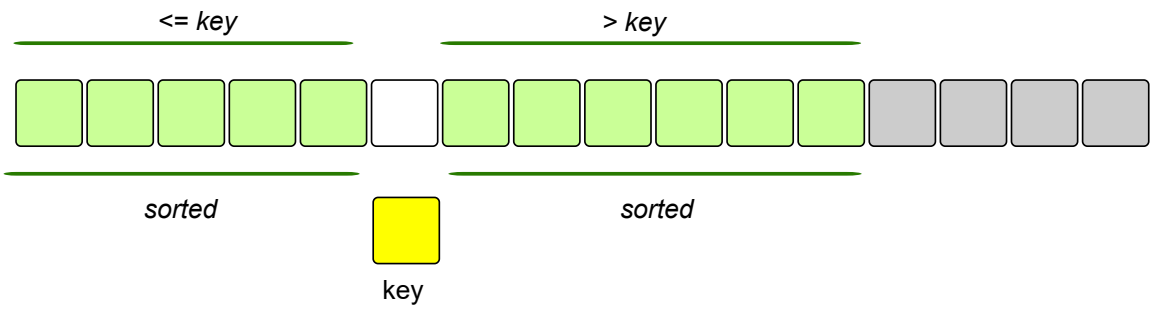
stop once we've run off the left end of the array. In the latter case, the first time we compare the key with an element to its left, we find that the key is already in its correct position relative to all elements to its left; no elements slide over and the key drops back into the position in which it started.

Inserting a value into Sorted Subarray

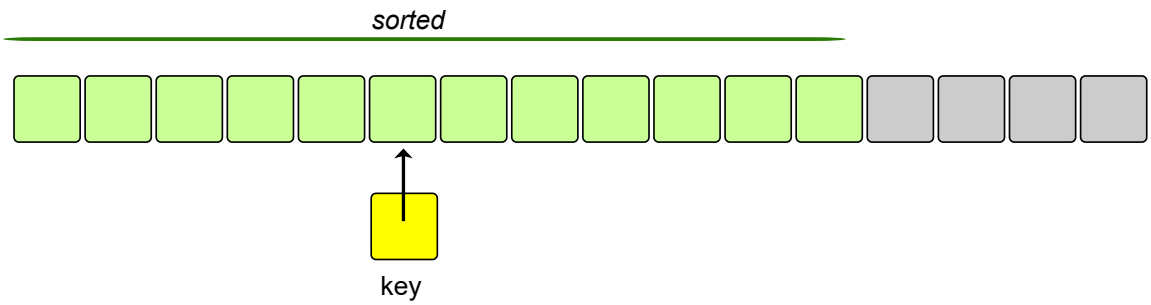
The main step in insertion sort is making space in an array to put the current value, which is stored in the variable key. As we saw above, we go through the subarray to the left of key's initial position, right to left, sliding each element that is greater than key one position to the right. Once we find an element that is less than key, or equal to key, we stop sliding and copy key into the vacated position just to the right of this element. (Of course, the position is not truly vacated, but its element was slid over to the right.) This diagram shows the idea:



1 of 3



2 of 3


sorted
key