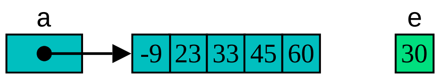
English name: ____________________

The Insertion Sort Algorithm

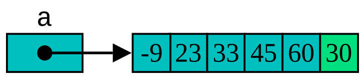
In this lesson, we will work on understanding the ***insertion sort*** algorithm. This document will describe the *insertion sort* algorithm that will sort an array in ascending order, and will sort the array starting from left, moving through to the right.

Part 1: Insertion (Inner Loop)

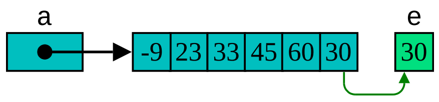
First, we will understand the general idea of inserting a new element into an array that has already been sorted in ascending order. The diagram below shows a sorted list, and to the right one additional element which we wish to insert into the array in the correct position so that the array will remain sorted.



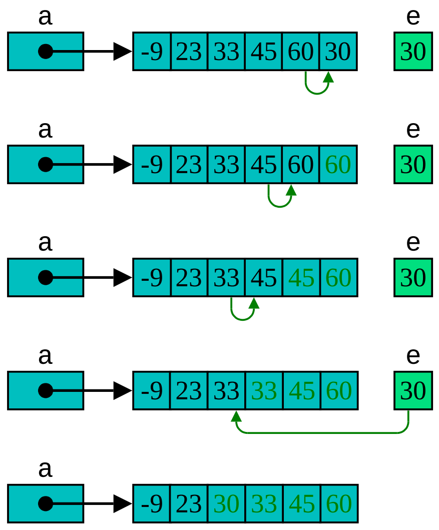
For this algorithm, rather than have the element to insert separate, we will start with the element to be inserted at the end of the array. (The reason we want this will become apparent when we look at the later steps of the algorithm.) The diagram below shows our array sorted in ascending order with an additional element at the end that has not yet been inserted into the correct position.



Each time we wish to insert an element into the correct position, the insertion sort algorithm starts by copying the element to insert into a temporary variable. For this document, we have decided to label the variable e (for element). We can then use the last element of the array as an empty slot in the array to allow us to shift elements to the right.

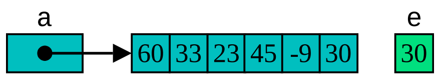


In order to determine the correct place to insert element e, we start at the penultimate (second to last) element, and compare it to the element stored in e. If the element in the array is greater than element e, it must be shifted. This process repeats, until: (1) it encounters an element that is smaller than or equal to element e, or (2) it arrives at the start of the list. For those who completed the PBL IntArray assignment, you have written similar code to shift the elements of an array to insert an element at a given index. The diagram below represents the process.

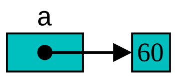


Part 2: Sorting (Outer Loop)

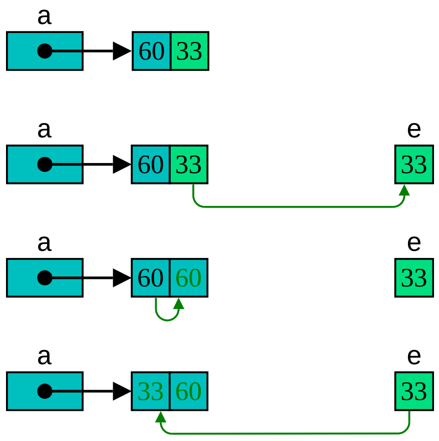
We will now extend the algorithm so that it may sort an unsorted array. For our discussion, we will be sorting this array:



We start by realizing that an array that contains a single element is necessarily sorted. We can see that even if it were a different element in the following array, it would still be sorted:



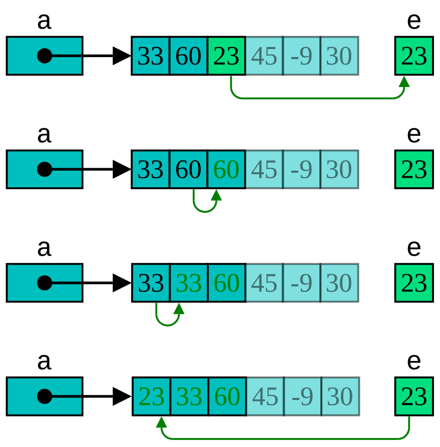
So if we start with an array of two elements, we can consider this as a sorted array of one element (colored cyan, or darker in grayscale), plus an element (colored lime green, or lighter in grayscale) that we must insert into the sorted array. We can use our algorithm from part 1 to insert the second element into the appropriate place in the array such that the resulting two-element array is sorted.



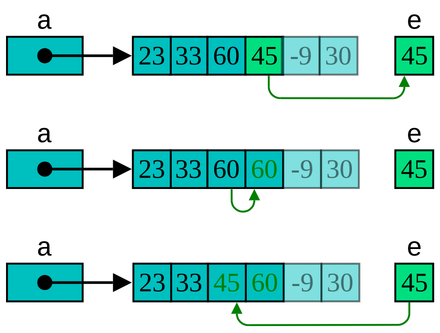
You may have noticed that this “two-element array” is actually the first two elements of the array that we said we are going to sort. Thus, we have modified our original, unsorted array such that the first two elements have been properly sorted.

We continue to perform the algorithm on the first three elements of our array, assuming the first two elements have been sorted. Then afterwards, on the first four elements of the array, assuming the first three elements have been sorted. Below are diagrammatic representations of our algorithm working on these three- and four- element arrays. The elements that are not being sorted are included in the diagram, but have been grayed-out.

Three-element array



Four-element array

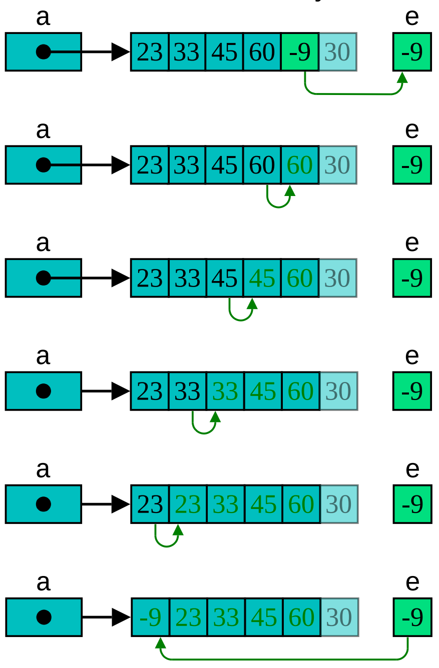


English name: ____________________

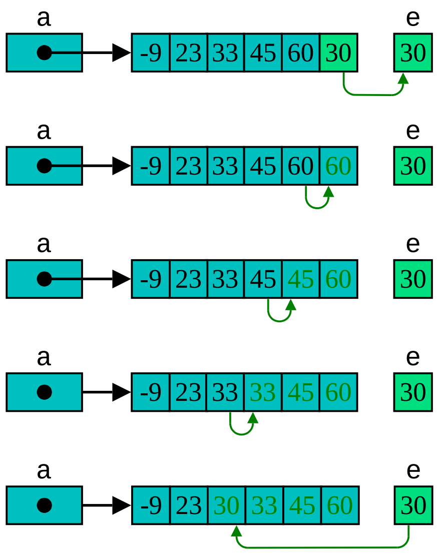
The Insertion Sort Algorithm

For the sake of completeness, a diagrammatic representation of the remaining steps of the algorithm to finish the sorting the array is given below.

Five-element array



Six-element array


a
e
-9 23 33 45 60 30
30
-9 23 33 45 60 60
-9 23 33 45 45 60
-9 23 33 33 45 60
-9 23 30 33 45 60