Saikrishna Arcot M. Hudachek-Buswell

August 18, 2018

• Given a set of items, people can "easily" sort the items into ascending or descending order.

- Given a set of items, people can "easily" sort the items into ascending or descending order.
- However, a computer cannot simply look at more than two items and easily sort them (whereas people can). Because of this, computers need to use an algorithm to sort items.

- Given a set of items, people can "easily" sort the items into ascending or descending order.
- However, a computer cannot simply look at more than two items and easily sort them (whereas people can). Because of this, computers need to use an algorithm to sort items.
- There are multiple such algorithms for sorting items. Some are easier to implement, but take longer to run.

In-place Sorts

 An in-place sort is a sorting algorithm that doesn't copy over elements into another array/list. (Creating variables to store a fixed number of items is allowed.) In other words, regardless of the length of the array to be sorted, a fixed amount of (additional) space is used.

In-place Sorts

- An in-place sort is a sorting algorithm that doesn't copy over elements into another array/list. (Creating variables to store a fixed number of items is allowed.) In other words, regardless of the length of the array to be sorted, a fixed amount of (additional) space is used.
- An out-of-place sort is a sorting algorithm that does allocate a variable amount of additional space.

Stable Sorts

A stable sort is a sort in which the order of duplicate items is preserved. For example, in the array, if there is a 4 near the starting of the array (let's call this 4a) and another 4 near the ending of the array (let's call this 4b), then after the array is sorted, 4a is guaranteed to be before 4b.

Stable Sorts

- A stable sort is a sort in which the order of duplicate items is preserved. For example, in the array, if there is a 4 near the starting of the array (let's call this 4a) and another 4 near the ending of the array (let's call this 4b), then after the array is sorted, 4a is guaranteed to be before 4b.
- An unstable sort is a sort in which the order of duplicate items may change.

• Six sorting algorithms will be covered:

- Six sorting algorithms will be covered:
 - Bubble sort

- Six sorting algorithms will be covered:
 - Bubble sort
 - Insertion sort

- Six sorting algorithms will be covered:
 - Bubble sort
 - Insertion sort
 - Selection sort

- Six sorting algorithms will be covered:
 - Bubble sort
 - Insertion sort
 - Selection sort
 - Merge sort

- Six sorting algorithms will be covered:
 - Bubble sort
 - Insertion sort
 - Selection sort
 - Merge sort
 - Quick sort

- Six sorting algorithms will be covered:
 - Bubble sort
 - Insertion sort
 - Selection sort
 - Merge sort
 - Quick sort
 - Radix sort

- Six sorting algorithms will be covered:
 - Bubble sort
 - Insertion sort
 - Selection sort
 - Merge sort
 - Quick sort
 - Radix sort
- All of the above algorithms except for the last one are known as comparison sorts because they directly compare two items; radix sort does not directly compare two items.

- Six sorting algorithms will be covered:
 - Bubble sort
 - Insertion sort
 - Selection sort
 - Merge sort
 - Quick sort
 - Radix sort
- All of the above algorithms except for the last one are known as comparison sorts because they directly compare two items; radix sort does not directly compare two items.
- The first three sorting algorithms may also be referred to as $O(n^2)$ sorts because they run in $O(n^2)$ time for the average case.



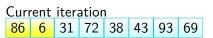
 In bubble sort, compare the first two items. If they are in the wrong order, then swap them; if not, then keep them as-is.
 Then, move on to the next two items.

- In bubble sort, compare the first two items. If they are in the wrong order, then swap them; if not, then keep them as-is.
 Then, move on to the next two items.
- Repeat the above step until you get to the end of the array.
 Once you reach the end of the array, you have performed an iteration of bubble sort.

- In bubble sort, compare the first two items. If they are in the wrong order, then swap them; if not, then keep them as-is.
 Then, move on to the next two items.
- Repeat the above step until you get to the end of the array.
 Once you reach the end of the array, you have performed an iteration of bubble sort.
- At this point, the largest item in the array is in the last spot.
 Run the previous two steps on the array again, but don't
 include the last spot. Now, the two largest items are at the
 end of the array (in the correct spots). Repeat until the array
 is sorted.

- In bubble sort, compare the first two items. If they are in the wrong order, then swap them; if not, then keep them as-is.
 Then, move on to the next two items.
- Repeat the above step until you get to the end of the array.
 Once you reach the end of the array, you have performed an iteration of bubble sort.
- At this point, the largest item in the array is in the last spot.
 Run the previous two steps on the array again, but don't
 include the last spot. Now, the two largest items are at the
 end of the array (in the correct spots). Repeat until the array
 is sorted.
- If you do not make any swaps during an iteration, then this
 means that the array is sorted, and you can terminate early.

		ite					
86	6	31	72	38	43	93	69



Previous iterations | 86 | 6 | 31 | 72 | 38 | 43 | 93 | 69 |

Cur	rent	itera	atior	1			
86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69

			ratio				
86	6	31	72	38	43	93	69

				atior				
8	36	6	31	72	38	43	93	69
	6	86	31	72	38	43	93	69
	6	31	86	72	38	43	93	69

Previous iterations | 86 | 6 | 31 | 72 | 38 | 43 | 93 | 69 |

	rent						
86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	72	86	38	43	93	69

Previous iterations | 86 | 6 | 31 | 72 | 38 | 43 | 93 | 69

Cur	rent						
86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	72	86	38	43	93	69
6	31	72	38	86	43	93	69

 Previous iterations

 86
 6
 31
 72
 38
 43
 93
 69

Current iteration

Previous iterations | 86 | 6 | 31 | 72 | 38 | 43 | 93 | 69

Curi	rent	itera	atior	1			
86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	72	86	38	43	93	69
6	31	72	38	86	43	93	69
6	31	72	38	43	86	93	69
6	31	72	38	43	86	93	69

Previous iterations | 86 | 6 | 31 | 72 | 38 | 43 | 93 | 69

Cur	rent			_			
86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	72	86	38	43	93	69
6	31	72	38	86	43	93	69
6	31	72	38	43	86	93	69
6	31	72	38	43	86	93	69
6	31	72	38	43	86	69	93

Prev	vious	ite	ratio	ns			
86	6	31	72	38	43	93	69
6	31	72	38	43	86	69	93

6 31 72 38 43 86 69 93			itera					
	6	31	72	38	43	86	69	93

		ite					
86	6	31	72	38	43	93	69
6	31	72	38	43	86	69	93

	rent						
6	31	72	38	43	86	69	93
6	31	72	38	43	86	69	93

Previous iterations							
86	6	31	72	38	43	93	69
6	31	72	38	43	86	69	93

(Current iteration								
	6	31	72	38	43	86	69	93	
	6	31	72	38	43	86	69	93	
	6	31	72	38	43	86	69	93	

Previous iterations									
86	6	6 31 72 38 43 93 69							
6	31	72	38	43	86	69	93		

Current iteration								
6	31	72	38	43	86	69	93	
6	31	72	38	43	86	69	93	
6	31	72	38	43	86	69	93	
6	31	38	72	43	86	69	93	

Previous iterations							
86	6	31	72	38	43	93	69
6	31	72	38	43	86	69	93

	Current iteration								
6	31	72	38	43	86	69	93		
6	31	72	38	43	86	69	93		
6	31	72	38	43	86	69	93		
6	31	38	72	43	86	69	93		
6	31	38	43	72	86	69	93		

Previous iterations							
86	6	31	72	38	43	93	69
6	31	72	38	43	86	69	93

Current iteration 6 31 72 38 43 86 69 93								
6	31	72	38	43	86	69	93	
6	31	72	38	43	86	69	93	
6	31	38	72	43	86	69	93	
6	31	38	43	72	86	69	93	
6	31	38	43	72	86	69	93	

Previous iterations								
86	6	31	31 72 38 43 93 69					
6	31	72	38	43	86	69	93	

Cur	Current iteration 6 31 72 38 43 86 69 93								
6	31	72	38	43	86	69	93		
6	31	72	38	43	86	69	93		
6	31	72	38	43	86	69	93		
6	31	38	72	43	86	69	93		
6	31	38	43	72	86	69	93		
6	31	38	43	72	86	69	93		
6	31	38	43	72	69	86	93		

Notice how the 86 and 93 weren't compared; this is because 93 is guaranteed to be the largest item in the array, and is guaranteed to be in the correct spot.

	Previous iterations									
86	6	31	72	38	43	93	69			
6	31	72	38	43	86	69	93			
6	31	38	43	72	69	86	93			

3

	Previous iterations										
86	6	31	72	38	43	93	69				
6	31	72	38	43	86	69	93				
6	31	38	43	72	69	86	93				

	rent			•			
6	31	38	43	72	69	86	93
6	31	38	43	72	69	86	93

86	6	31	72	38	43	93	69
				43			
6	31	38	43	72	69	86	93

Cur	rent	itera	atior	1			
6	31	38	43	72	69	86	93
6	31	38	43	72	69	86	93
6	31	38	43	72	69	86	93

86	6	31	72	38	43	93	69
				43			
6	31	38	43	72	69	86	93

	Current iteration 6 31 38 43 72 69 86 93									
6	31	38	43	72	69	86	93			
6	31	38	43	72	69	86	93			
6	31	38	43	72	69	86	93			
6	31	38	43	72	69	86	93			

	Previous iterations										
86	6	31	72	38	43	93	69				
6	31	72	38	43	86	69	93				
6	31	38	43	72	69	86	93				

	Current iteration										
6	31	38	43	72	69	86	93				
6	31	38	43	72	69	86	93				
6	31	38	43	72	69	86	93				
6	31	38	43	72	69	86	93				
6	31	38	43	72	69	86	93				

	Previous iterations										
86	6	31	72	38	43	93	69				
6	31	72	38	43	86	69	93				
6	31	38	43	72	69	86	93				

Curi 6	31				69	86	93
6	31	38	43	72	69	86	93
6	31	38	43	72	69	86	93
6	31	38	43	72	69	86	93
6	31	38	43	72	69	86	93
6	31	38	43	69	72	86	93

Notice how the 72 and 86 weren't compared; this is because 86 and 93 are guaranteed to be the largest items in the array, and are guaranteed to be in the correct spots.

Prev	ious	itera	tions
1 1 5 7	ıvus	ILCIA	しいしいろ

86	6	31	72	38	43	93	69
6	31	72	38	43	86	69	93
6	31	38	43	72	69	86	93
6	31	38	43	69	72	86	93

Current iteration

6 31 38 43 69 72 86 93

Previous iterations

86	6	31	72	38	43	93	69
6	31	72	38	43	86	69	93
6	31	38	43	72	69	86	93
6	31	38	43	69	72	86	93

Current iteration

	31				72	86	93
6	31	38	43	69	72	86	93

Previous	iterations

_							
86	6	31	72	38	43	93	69
6	31	72	38	43	86	69	93
6	31	38	43	72	69	86	93
6	31	38	43	69	72	86	93

Current iteration

Cui	6 31 38 43 69 72 86 93									
6	31	38	43	69	72	86	93			
6	31	38	43	69	72	86	93			
6	31	38	43	69	72	86	93			

Previous iterations										
86	6	31	72	38	43	93	69			
6	31	72	38	43	86	69	93			
6	31	38	43	72	69	86	93			
6	31	38	43	69	72	86	93			

Current iteration									
6	31	38	43	69	72	86	93		
6	31	38	43	69	72	86	93		
6	31	38	43	69	72	86	93		
6	31	38	43	69	72	86	93		

		٠.	
ч	revious	itera	itions

86	6	31	72	38	43	93	69
6	31	72	38	43	86	69	93
6	31	38	43	72	69	86	93
6	31	38	43	69	72	86	93

Current iteration

	Lurrent iteration										
6	31	38	43	69	72	86	93				
6	31	38	43	69	72	86	93				
6	31	38	43	69	72	86	93				
6	31	38	43	69	72	86	93				
6	31	38	43	69	72	86	93				

Because no swaps were made in this iteration, the array must be sorted.

```
procedure BubbleSort(array)
    length \leftarrow length of array
    i \leftarrow 0
    swapped \leftarrow TRUE
    while i < length - 1 and swappedis TRUE do
        swapped \leftarrow FALSE
        for i \leftarrow 0, length - i - 1 do
           if array[i] > array[i+1] then
               swap array[j] and array[j+1]
                swapped \leftarrow TRUE
            end if
        end for
    end while
end procedure
```

• In the best case, if the array is already sorted in the correct order, only one iteration of bubble sort will be done because no swaps will be made on that iteration. This means that n-1 comparisons will be done and therefore the best case big-O of bubble sort is O(n).

- In the best case, if the array is already sorted in the correct order, only one iteration of bubble sort will be done because no swaps will be made on that iteration. This means that n-1 comparisons will be done and therefore the best case big-O of bubble sort is O(n).
- In the worst case, if the array is sorted, but in the reverse order, then all n-1 iterations of bubble sort will need to be done because at least one swap will be made on each iteration. This means that $\frac{n(n-1)}{2}$ comparisons will be done and therefore the worst case big-O of bubble sort is $O(n^2)$.

- In the best case, if the array is already sorted in the correct order, only one iteration of bubble sort will be done because no swaps will be made on that iteration. This means that n-1 comparisons will be done and therefore the best case big-O of bubble sort is O(n).
- In the worst case, if the array is sorted, but in the reverse order, then all n-1 iterations of bubble sort will need to be done because at least one swap will be made on each iteration. This means that $\frac{n(n-1)}{2}$ comparisons will be done and therefore the worst case big-O of bubble sort is $O(n^2)$.
- In the average case, bubble sort runs in $O(n^2)$ time, because the actual number of comparisons that would be done is somewhere between n-1 and $\frac{n(n-1)}{2}$.

- In the best case, if the array is already sorted in the correct order, only one iteration of bubble sort will be done because no swaps will be made on that iteration. This means that n-1 comparisons will be done and therefore the best case big-O of bubble sort is O(n).
- In the worst case, if the array is sorted, but in the reverse order, then all n-1 iterations of bubble sort will need to be done because at least one swap will be made on each iteration. This means that $\frac{n(n-1)}{2}$ comparisons will be done and therefore the worst case big-O of bubble sort is $O(n^2)$.
- In the average case, bubble sort runs in $O(n^2)$ time, because the actual number of comparisons that would be done is somewhere between n-1 and $\frac{n(n-1)}{2}$.
- Bubble sort runs in-place and is a stable sort.

In insertion sort, assume the first item is sorted. Then, take
the second item, and "slide" it to the left so that it is correctly
placed in the sorted portion of the array. The first two items
are now considered sorted, and one iteration has been done.

- In insertion sort, assume the first item is sorted. Then, take
 the second item, and "slide" it to the left so that it is correctly
 placed in the sorted portion of the array. The first two items
 are now considered sorted, and one iteration has been done.
- Repeat with the third item and so on until the entire array is sorted.

- In insertion sort, assume the first item is sorted. Then, take
 the second item, and "slide" it to the left so that it is correctly
 placed in the sorted portion of the array. The first two items
 are now considered sorted, and one iteration has been done.
- Repeat with the third item and so on until the entire array is sorted.
- Unlike bubble sort, a fixed number of iterations are done for insertion sort.

Previous iterations										
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			

Current iteration											
6	31	72	38	43	93	69					
						ent iteration 6 31 72 38 43 93					

Previous iterations										
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			

Current iteration										
86	6	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			

Previous iterations										
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			

	Current iteration											
86	6	31	72	38	43	93	69					
6	86	31	72	38	43	93	69					

6 86 31 72 38 43 93 69

Previous iterations										
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			

Current iteration										
6	86	31	72	38	43	93	69			

Prev	Previous iterations										
86	6	31	72	38	43	93	69				
86	6	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				

C	urı	rent	itera	atior	1			
	6	86	31	72	38	43	93	69
	6	31	86	72	38	43	93	69

	Previous iterations										
86	6	31	72	38	43	93	69				
86	6	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				

Current iteration										
6	86	31	72	38	43	93	69			
6	31	86	72	38	43	93	69			
6	31	86	72	38	43	93	69			

	Previous iterations									
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	31	86	72	38	43	93	69			

 Current iteration

 6
 31
 86
 72
 38
 43
 93
 69

Prev	Previous iterations									
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	31	86	72	38	43	93	69			

			atior				
6	31	86	72	38	43	93	69
6	31	72	86	38	43	93	69

Prev	vious	s ite	ratio	ns			
86	6	31	72	38	43	93	69
86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69

Cur	rent	itera	atior	1			
6	31	86	72	38	43	93	69
6	31	72	86	38	43	93	69
6	31	72	86	38	43	93	69

Previous iterations									
86	6	31	72	38	43	93	69		
86	6	31	72	38	43	93	69		
6	86	31	72	38	43	93	69		
6	31	86	72	38	43	93	69		
6	31	72	86	38	43	93	69		

(rent						
	6	31	72	86	38	43	93	69

Prev	/ious	ite	ratio	ns			
86	6	31	72	38	43	93	69
86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	72	86	38	43	93	69

			ation 86		43	93	69
6	31	72	38	86	43	93	69

C...... !++...+!+...

Pre	evious	s ite	ratio	ns			
86	6	31	72	38	43	93	69
86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	72	86	38	43	93	69

C		~		atior				
	6	31	72	86	38	43	93	69
	6	31	72	38	86	43	93	69
	6	31	38	72	86	43	93	69

Prev		ite					
86	6	31	72	38	43	93	69
86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	72	86	38	43	93	69

31	72	86	38	43	93	69
31	72	38	86	43	93	69
31	38	72	86	43	93	69
31	38	72	86	43	93	69
	31 31 31	31 7231 7231 38	31 72 86 31 72 38 31 38 72	31 72 38 86 31 38 72 86	31 72 86 38 43 31 72 38 86 43 31 38 72 86 43	31 72 86 38 43 93 31 72 38 86 43 93 31 38 72 86 43 93

Previous iterations										
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	31	86	72	38	43	93	69			
6	31	72	86	38	43	93	69			
6	31	38	72	86	43	93	69			

		itera					
6	31	38	72	86	43	93	69

Prev	vious	s ite	ratio	ns			
86	6	31	72	38	43	93	69
86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	72	86	38	43	93	69
6	31	38	72	86	43	93	69

Current iteration											
6	31	38	72	86	43	93	69				
6	31	38	72	43	86	93	69				

Prev	vious	ite	ratio	ns			
86	6	31	72	38	43	93	69
86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	72	86	38	43	93	69
6	31	38	72	86	43	93	69

6 31		72	86	43	0.3	60
6 21				.0	93	09
0 31	38	72	43	86	93	69
6 31	38	43	72	86	93	69

Prev	/ious	ite	ratio	ns			
86	6	31	72	38	43	93	69
86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	72	86	38	43	93	69
6	31	38	72	86	43	93	69

	rent						
6	31	38	72	86	43	93	69
6	31	38	72	43	86	93	69
6	31	38	43	72	86	93	69
6	31	38	43	72	86	93	69

Prev	Previous iterations										
86	6	31	72	38	43	93	69				
86	6	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				
6	31	86	72	38	43	93	69				
6	31	72	86	38	43	93	69				
6	31	38	72	86	43	93	69				
6	21	38	13	72	26	03	60				

 Current iteration
 6
 31
 38
 43
 72
 86
 93
 69

\neg		•	
_	revious	iterat	IONG
	ICVIOUS	ILCIAL	.10113

	, ious		utio				
86	6	31	72	38	43	93	69
86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
					43		
6	31	72	86	38	43	93	69
6	31	38	72	86	43	93	69
6	31	38	43	72	86	93	69

	CIIL						
6	31	38	43	72	86	93	69
6	31	38	43	72	86	93	69

Prev 86	6			38	43	93	69
86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	72	86	38	43	93	69
6	31	38	72	86	43	93	69
6	31	38	43	72	86	93	69
6	31	38	43	72	86	93	69

	rent						
6	31	38	43	72	86	93	69

Pre۱	Previous iterations											
86	6	31	72	38	43	93	69					
86	6	31	72	38	43	93	69					
6	86	31	72	38	43	93	69					
6	31	86	72	38	43	93	69					
6	31	72	86	38	43	93	69					
6	31	38	72	86	43	93	69					
6	31	38	43	72	86	93	69					
6	31	38	43	72	86	93	69					

Cur	rent	itera	atior	1			
6	31	38	43	72	86	93	69
6	31	38	43	72	86	69	93

Prev	Previous iterations										
86	6	31	72	38	43	93	69				
86	6	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				
6	31	86	72	38	43	93	69				
6	31	72	86	38	43	93	69				
6	31	38	72	86	43	93	69				
6	31	38	43	72	86	93	69				
6	31	38	43	72	86	93	69				

	Current iteration								
6	31	38	43	72	86	93	69		
6	31	38	43	72	86	69	93		
6	31	38	43	72	69	86	93		

Prev	/ious	ite	ratio	ns			
86	6	31	72	38	43	93	69
86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	72	86	38	43	93	69
6	31	38	72	86	43	93	69
6	31	38	43	72	86	93	69
6	31	38	43	72	86	93	69

Cur	Current iteration 6 31 38 43 72 86 93 69									
6	31	38	43	72	86	93	69			
6	31	38	43	72	86	69	93			
6	31	38	43	72	69	86	93			
6	31	38	43	69	72	86	93			

Previous iterations										
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	31	86	72	38	43	93	69			
6	31	72	86	38	43	93	69			
6	31	38	72	86	43	93	69			
6	31	38	43	72	86	93	69			
6	31	38	43	72	86	93	69			

Current iteration									
6	31	38	43	72	86	93	69		
6	31	38	43	72	86	69	93		
6	31	38	43	72	69	86	93		
6	31	38	43	69	72	86	93		
6	31	38	43	69	72	86	93		

```
procedure InsertionSort(array)
    length \leftarrow length of array
    for i \leftarrow 1, length - 1 do
       i \leftarrow i
       while j > 0 and array[j-1] > array[j] do
           swap array[i-1] and array[i]
           j \leftarrow j - 1
        end while
    end for
end procedure
```

• In the best case, if the array is already sorted in the correct order, then only one comparison will be done for each iteration of insertion sort (because the item will be already in the right slot). This means that n-1 comparisons will be done and therefore the best case big-O of insertion sort is O(n).

- In the best case, if the array is already sorted in the correct order, then only one comparison will be done for each iteration of insertion sort (because the item will be already in the right slot). This means that n-1 comparisons will be done and therefore the best case big-O of insertion sort is O(n).
- In the worst case, if the array is sorted, but in the reverse order, then each iteration of insertion sort will do the maximum number of comparisons possible (because each item has to slide all the way to the left). Specifically, this means that $\frac{n(n-1)}{2}$ comparisons will be done and therefore the worst case big-O of insertion sort is $O(n^2)$.

- In the best case, if the array is already sorted in the correct order, then only one comparison will be done for each iteration of insertion sort (because the item will be already in the right slot). This means that n-1 comparisons will be done and therefore the best case big-O of insertion sort is O(n).
- In the worst case, if the array is sorted, but in the reverse order, then each iteration of insertion sort will do the maximum number of comparisons possible (because each item has to slide all the way to the left). Specifically, this means that $\frac{n(n-1)}{2}$ comparisons will be done and therefore the worst case big-O of insertion sort is $O(n^2)$.
- In the average case, insertion sort runs in $O(n^2)$ time.

- In the best case, if the array is already sorted in the correct order, then only one comparison will be done for each iteration of insertion sort (because the item will be already in the right slot). This means that n-1 comparisons will be done and therefore the best case big-O of insertion sort is O(n).
- In the worst case, if the array is sorted, but in the reverse order, then each iteration of insertion sort will do the maximum number of comparisons possible (because each item has to slide all the way to the left). Specifically, this means that $\frac{n(n-1)}{2}$ comparisons will be done and therefore the worst case big-O of insertion sort is $O(n^2)$.
- In the average case, insertion sort runs in $O(n^2)$ time.
- Insertion sort runs in-place and is a stable sort.

 In selection sort, search the entire array for the smallest item (start by assuming the first item you see is the smallest item).
 Swap that item with the first item.

- In selection sort, search the entire array for the smallest item (start by assuming the first item you see is the smallest item).
 Swap that item with the first item.
- Then, search the entire array (excluding the first item) for the next smallest item. Swap that item with the second item.

- In selection sort, search the entire array for the smallest item (start by assuming the first item you see is the smallest item).
 Swap that item with the first item.
- Then, search the entire array (excluding the first item) for the next smallest item. Swap that item with the second item.
- Repeat until the entire array is sorted.

Previous iterations | 86 | 6 | 31 | 72 | 38 | 43 | 93 | 69 |

Current iteration 86 6 31 72 38 43 93 69

Previous iterations | 86 | 6 | 31 | 72 | 38 | 43 | 93 | 69

			atior				
86	6	31	72	38	43	93	69
86	6	31	72	38	43	93	69

Current iteration								
86	6	31	72	38	43	93	69	
86	6	31	72	38	43	93	69	
86	6	31	72	38	43	93	69	

Current iteration									
86	6	31	72	38	43	93	69		
86	6	31	72	38	43	93	69		
86	6	31	72	38	43	93	69		
86	6	31	72	38	43	93	69		

Curi	Current iteration									
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			

Curi	Current iteration									
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			

Current iteration										
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			
86	6	31	72	38	43	93	69			

Current iteration									
86	6	31	72	38	43	93	69		
86	6	31	72	38	43	93	69		
86	6	31	72	38	43	93	69		
86	6	31	72	38	43	93	69		
86	6	31	72	38	43	93	69		
86	6	31	72	38	43	93	69		
86	6	31	72	38	43	93	69		
86	6	31	72	38	43	93	69		

Curi	Current iteration										
86	6	31	72	38	43	93	69				
86	6	31	72	38	43	93	69				
86	6	31	72	38	43	93	69				
86	6	31	72	38	43	93	69				
86	6	31	72	38	43	93	69				
86	6	31	72	38	43	93	69				
86	6	31	72	38	43	93	69				
86	6	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				

Previous iterations								
86	6	31	72	38	43	93	69	
6	86	31	72	38	43	93	69	

Current iteration										
6	86	31	72	38	43	93	69			

Previous iterations										
86	6	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			

Current iteration										
6	86	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			

Previous iterations											
86	6	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				

Cur	Current iteration										
6	86	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				

Prev	Previous iterations											
86	6	31	72	38	43	93	69					
6	86	31	72	38	43	93	69					

Current iteration											
6	86	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				

Prev	Previous iterations											
86	6	31	72	38	43	93	69					
6	86	31	72	38	43	93	69					

Cur	Current iteration									
6	86	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			

Previous iterations											
86	6	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				

Cur	rent	itera	atior	1			
6	86	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	86	31	72	38	43	93	69

Previous iterations											
86	6	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				

Current iteration										
6	86	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			

Previous iterations										
86	6	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			

Current iteration										
6	86	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	31	86	72	38	43	93	69			

Previous iterations

86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69

6 3	86	72	38	43	93	69
-----	----	----	----	----	----	----

Previous iterations

	rious		acio				
86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69

Cui	CIIL	ILCI					
6	31	86	72	38	43	93	69
6	31	86	72	38	43	93	69

\neg			• •		
Ρ	rav.	\cap	s ite	rati	\cap nc

86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69

	CIIL						
6	31	86	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	86	72	38	43	93	69

			• .		
Ρ	rav/	IOIIC	ITA	ratio	nc

86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69

	CIIL						
6	31	86	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	86	72	38	43	93	69

			• .		
Ρ	rav/	IOIIC	ITA	ratio	nc

86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69

Cur	6 31 86 72 38 43 93 69									
6	31	86	72	38	43	93	69			
	31									
6	31	86	72	38	43	93	69			
	31									
6	31	86	72	38	43	93	69			

Previous iterations

86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69

Current iteration									
6	31	86	72	38	43	93	69		
6	31	86	72	38	43	93	69		
6	31	86	72	38	43	93	69		
6	31	86	72	38	43	93	69		
6	31	86	72	38	43	93	69		
6	31	86	72	38	43	93	69		

Previous iterations

86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69

Cui	CIIL	ILCI	atioi				
6	31	86	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	86	72	38	43	93	69
	31						
6	31	86	72	38	43	93	69
6	31	38	72	86	43	93	69

Previous iterations											
86	6	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				
6	31	86	72	38	43	93	69				
6	31	38	72	86	43	93	69				

69

Previous iterations											
86	6	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				
6	31	86	72	38	43	93	69				
6	31	38	72	86	43	93	69				

Current iteration										
6	31	38	72	86	43	93	69			
6	31	38	72	86	43	93	69			

Previous iterations											
86	6	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				
6	31	86	72	38	43	93	69				
6	31	38	72	86	43	93	69				

Current iteration											
86 43 93 69											
86 43 93 69											
86 <mark>43</mark> 93 69											

	Previous iterations											
8	86	6	31	72	38	43	93	69				
	6	86	31	72	38	43	93	69				
	6	31	86	72	38	43	93	69				
	6	31	38	72	86	43	93	69				

	Current iteration											
6	31	38	72	86	43	93	69					
6	31	38	72	86	43	93	69					
6	31	38	72	86	43	93	69					
6	31	38	72	86	43	93	69					

Previous iterations											
86	6	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				
6	31	86	72	38	43	93	69				
6	31	38	72	86	43	93	69				

	Current iteration											
6	31	38	72	86	43	93	69					
6	31	38	72	86	43	93	69					
6	31	38	72	86	43	93	69					
6	31	38	72	86	43	93	69					
6	31	38	72	86	43	93	69					

Previous iterations											
86	6	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				
6	31	86	72	38	43	93	69				
6	31	38	72	86	43	93	69				

Cur	Current iteration 6 31 38 72 86 43 93 69										
6	31	38	72	86	43	93	69				
6	31	38	72	86	43	93	69				
6	31	38	72	86	43	93	69				
6	31	38	72	86	43	93	69				
6	31	38	72	86	43	93	69				
6	31	38	43	86	72	93	69				

Previous iterations											
86	6	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				
6	31	86	72	38	43	93	69				
6	31	38	72	86	43	93	69				
6	31	38	43	86	72	93	69				

	Current iteration 6 31 38 43 86 72 93 69												
6	31	38	43	86	72	93	69						

Previous iterations											
86	6	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				
6	31	86	72	38	43	93	69				
6	31	38	72	86	43	93	69				
6	31	38	43	86	72	93	69				

	Current iteration												
6	31	38	43	86	72	93	69						
6	31	38	43	86	72	93	69						
	O1	50	10	00	1-	33	UJ.						

Previous iterations											
86	6	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				
6	31	86	72	38	43	93	69				
6	31	38	72	86	43	93	69				
6	31	38	43	86	72	93	69				

6 31 38 43 86 72 9 6 31 38 43 86 72 9	3 69
6 31 38 43 86 72 9	- 00
	3 69
6 31 38 43 86 72 9	69

Previous iterations										
86	6	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	31	86	72	38	43	93	69			
6	31	38	72	86	43	93	69			
6	31	38	43	86	72	93	69			

(rent						
	6	31	38	43	86	72	93	69
	6	31	38	43	86	72	93	69
	6	31	38	43	86	72	93	69
	6	31	38	43	86	72	93	69

Prev	vious	s ite	ratio	ns			
86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	38	72	86	43	93	69
6	31	38	43	86	72	93	69

Curi	rent						
6	31	38	43	86	72	93	69
6	31	38	43	86	72	93	69
6	31	38	43	86	72	93	69
6	31	38	43	86	72	93	69
6	31	38	43	86	72	93	69

Previous iterations											
86	6	31	72	38	43	93	69				
6	86	31	72	38	43	93	69				
6	31	86	72	38	43	93	69				
6	31	38	72	86	43	93	69				
6	31	38	43	86	72	93	69				

Cur	rent						
6	31	38	43	86	72	93	69
6	31	38	43	86	72	93	69
6	31	38	43	86	72	93	69
6	31	38	43	86	72	93	69
6	31	38	43	86	72	93	69
6	31	38	43	69	72	93	86

Prev	Previous iterations												
86	6	31	72	38	43	93	69						
6	86	31	72	38	43	93	69						
6	31	86	72	38	43	93	69						
6	31	38	72	86	43	93	69						
6	31	38	43	86	72	93	69						
6	31	38	43	69	72	93	86						

Curi	current iteration												
6	31	38	43	69	72	93	86						

Prev	Previous iterations												
86	6	31	72	38	43	93	69						
6	86	31	72	38	43	93	69						
6	31	86	72	38	43	93	69						
6	31	38	72	86	43	93	69						
6	31	38	43	86	72	93	69						
6	31	38	43	69	72	93	86						

Current iteration											
6	31	38	43	69	72	93	86				
6	31	38	43	69	72	93	86				
	01		.0	03		30					

Prev	Previous iterations												
86	6	31	72	38	43	93	69						
6	86	31	72	38	43	93	69						
6	31	86	72	38	43	93	69						
6	31	38	72	86	43	93	69						
6	31	38	43	86	72	93	69						
6	31	38	43	69	72	93	86						

(Current iteration								
	6	31	38	43	69	72	93	86	
	6	31	38	43	69	72	93	86	
	6	31	38	43	69	72	93	86	

Previous iterations										
86	6	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	31	86	72	38	43	93	69			
6	31	38	72	86	43	93	69			
6	31	38	43	86	72	93	69			
6	31	38	43	69	72	93	86			

	Current iteration									
6	31	38	43	69	72	93	86			
6	31	38	43	69	72	93	86			
6	31	38	43	69	72	93	86			
6	31	38	43	69	72	93	86			

Prev	ious	iter	atio	ns

1 Tevious iterations										
86	6	31	72	38	43	93	69			
6	86	31	72	38	43	93	69			
6	31	86	72	38	43	93	69			
6	31	38	72	86	43	93	69			
6	31	38	43	86	72	93	69			
6	31	38	43	69	72	93	86			
6	31	38	43	69	72	93	86			

6	31	38	43	69	72	93	86	

_	
Draviance	iterations
r revious	Heranons

86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	38	72	86	43	93	69
6	31	38	43	86	72	93	69
6	31	38	43	69	72	93	86
6	31	38	43	69	72	93	86

Current iteration										
6	31	38	43	69	72	93	86			
6	31	38	43	69	72	93	86			

_	
Draviance	iterations
r revious	Heranons

86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	38	72	86	43	93	69
6	31	38	43	86	72	93	69
6	31	38	43	69	72	93	86
6	31	38	43	69	72	93	86

6 31 38 43 69 72 93 86									
6	31	38	43	69	72	93	86		
6	31	38	43	69	72	93	86		
6	31	38	43	69	72	93	86		

		• •	
Ρ	revious	ITARS	o†i∩no
	ICVIOUS		

86	6	31	72	38	43	93	69
6	86	31	72	38	43	93	69
6	31	86	72	38	43	93	69
6	31	38	72	86	43	93	69
6	31	38	43	86	72	93	69
6	31	38	43	69	72	93	86
6	31	38	43	69	72	93	86

Current recrution								
	6	31	38	43	69	72	93	86
	6	31	38	43	69	72	93	86
	6	31	38	43	69	72	93	86
	6	31	38	43	69	72	86	93

Pre۱	/ious	ite	ratio	ns
				_

i revious iterations								
86	6	31	72	38	43	93	69	
6	86	31	72	38	43	93	69	
6	31	86	72	38	43	93	69	
6	31	38	72	86	43	93	69	
6	31	38	43	86	72	93	69	
6	31	38	43	69	72	93	86	
6	31	38	43	69	72	93	86	

Current iteration							
6	31	38	43	69	72	93	86
6	31	38	43	69	72	93	86
6	31	38	43	69	72	93	86
6	31	38	43	69	72	86	93
6	31	38	43	69	72	86	93

```
procedure SelectionSort(array)
    length \leftarrow length of array
    for i \leftarrow 0, length do
        minIndex \leftarrow i
        for i \leftarrow i + 1, length do
            if array[j] < array[minIndex] then
                minIndex \leftarrow i
            end if
        end for
        swap array[minIndex] and array[i]
    end for
end procedure
```

Selection Sort Performance

• Selection sort does the same number of comparisons in all cases $(\frac{n(n-1)}{2})$, since there is no early termination of any kind. The big-O of selection sort is $O(n^2)$.

Selection Sort Performance

- Selection sort does the same number of comparisons in all cases $(\frac{n(n-1)}{2})$, since there is no early termination of any kind. The big-O of selection sort is $O(n^2)$.
- Selection sort runs in-place, but is **not** a stable sort.