

# **Sorting Algorithms**

# Sorting Algorithms

There are different ways to sort a list. Some methods of sorting a list could be faster or slower than others.

The four Sorting algorithms we are going to study are:

- Selection sort/ Simple sort
- Insertion sort
- Bubble sort
- Quicksort (Higher Level only)

# Selection/Simple Sort

As selection sort basically looks for the smallest item (ascending order) and places it in the first position. It then finds the next smallest item and places it in the second position and so on.

The selection sort iterates from left to right through each list (right to left for descending order). For each list item, the algorithm searches the unsorted section for an element that is less than the current index..

If there is an element smaller than the current index, they are swapped and the algorithm moves onto the next list item.

This is repeated until the algorithm moves onto the last list item.

Select the first item in the list

[85, 24, 63, 45, 17, 31, 96, 50]

Find smallest in the unsorted section

[85, 24, 63, 45, 17, 31, 96, 50]

Swap them

[17, 24, 63, 45, 85, 31, 96, 50]

As we now know that the first item is sorted move onto the next item and repeat..

[17, 24, 63, 45, 85, 31, 96, 50] No Swap

[17, 24, 63, 45, 85, 31, 96, 50] Swap

[17, 24, 31, 45, 85, 63, 96, 50] No Swap

[17, 24, 31, 45, 85, 63, 96, 50] Swap

[17, 24, 31, 45, 50, 63, 96, 85] No Swap

[17, 24, 31, 45, 50, 63, 96, 85] Swap

[17, 24, 31, 45, 50, 63, 85, 96] Sorted

# Selection Sort

[https://en.wikipedia.org/wiki/Selection\\_sort#/media/File:Selection-Sort-Animation.gif](https://en.wikipedia.org/wiki/Selection_sort#/media/File:Selection-Sort-Animation.gif)

	8
	5
	2
	6
	9
	3
	1
	4
	0
	7

# Selection Sort - Python Code

<https://repl.it/@suzannelinnane/Selection-Sort>

```
myList = [85, 24, 63, 45, 17, 31, 96, 50]
```

```
for index in range(len(myList)):
```

```
    print(myList)
```

```
    # Find min value in unsorted section
```

```
    nextMinValue = min(myList[index+1:])
```

```
    # swap if there is a smaller value
```

```
    if nextMinValue < myList[index]:
```

```
        # Find location of min value to swap
```

```
        nextMinIndex = myList.index(nextMinValue)
```

```
    # Swap within unsorted section if smaller
```

```
    myList[nextMinIndex] = myList[index]
```

```
    myList[index] = nextMinValue
```

```
print(myList)
```

Output:

```
[85, 24, 63, 45, 17, 31, 96, 50]
[17, 24, 63, 45, 85, 31, 96, 50]
[17, 24, 63, 45, 85, 31, 96, 50]
[17, 24, 31, 45, 85, 63, 96, 50]
[17, 24, 31, 45, 85, 63, 96, 50]
[17, 24, 31, 45, 50, 63, 96, 85]
[17, 24, 31, 45, 50, 63, 96, 85]
[17, 24, 31, 45, 50, 63, 85, 96]
```

# Selection Sort - Python Code

Later on, we will be looking at the running times and efficiency of the algorithms.

When calculating the worst case running time for a selection sort, the `min()` function is really a loop.

It loops through the unsorted section to find the minimum value.

The following code is an alternative (and more complicated) to using the `min()` function.

<https://repl.it/@suzannelinnane/Selection-Sort-without-min>

Press **Esc** to exit full screen

# Sorting Algorithms



Activate Windows  
Go to Settings to activate Windows.



Slide 1 ▼



Q & A



Notes



Pointer



Captions ▼



Tips



EXIT





# Insertion Sort


An insertion sort is essentially how you might sort a deck of cards. You start with two cards and order these. Then for every new card, you decide where to place this new card within the sorted cards.

A computer cannot visualise the sorted cards like a human, it must iterate from right to left through the sorted cards to determine where the card is inserted.

Press **Esc** to exit full screen

# Insertion Sort

An insertion sort is essentially how you might sort a deck of cards. You start with two cards and order these. Then for every new card, you decide where to place this new card within the sorted cards.

A computer cannot visualise the sorted cards like a human, it must iterate from right to left through the sorted cards to determine where the card is inserted. 

Activate Windows  
Go to Settings to activate Windows.



Slide 8



Q & A



Notes



Pointer



Captions



Tips



EXIT



# Insertion Sort

The algorithm takes the item outlined in blue and keeps iterating left until it finds its ordered position.

Computers cannot see the list as humans do so this algorithm keeps shifting the sorted items up the list (to the right) when they are larger than the item being inserted.

Select the second item in the list

[85, 24, 63, 45, 17, 31, 96, 50]

Decide if the first two items need to be swapped

[85, 24, 63, 45, 17, 31, 96, 50]

Insert the current item into the appropriate location in the sorted list

[24, 85, 63, 45, 17, 31, 96, 50]

Move onto the next item and repeat..

[24, 85, 63, 45, 17, 31, 96, 50] Inserts between 24 and 85

[24, 63, 85, 45, 17, 31, 96, 50] Inserts between 24 and 63

[24, 45, 63, 85, 17, 31, 96, 50] Inserts at first position

[17, 24, 45, 63, 85, 31, 96, 50] Inserts between 24 and 45

[17, 24, 31, 45, 63, 85, 96, 50] Do not insert

[17, 24, 31, 45, 63, 85, 96, 50] Insert between 45 and 63

[17, 24, 31, 45, 50, 63, 85, 96] Sorted

   = Current Item

   = Sorted

# Insertion Sort - Example

[https://en.wikipedia.org/wiki/Insertion\\_sort#/media/File:Insertion-sort-example-300px.gif](https://en.wikipedia.org/wiki/Insertion_sort#/media/File:Insertion-sort-example-300px.gif)

6 5 3 1 8 7 2 4

# Insertion Sort - Python Code

<https://repl.it/@suzannelinnane/Insertion-Sort>

```
myList = [85, 24, 63, 45, 17, 31, 96, 50]
for index in range(1, len(myList)):
    print(myList)

    # Gets current item to insert and position it is in
    itemInsert = myList[index]
    position = index

    # Loops through sorted section, right to left,
    # to find where the current item needs to be inserted
    # in the sorted section, shifting other items to the right
    while position > 0 and myList[position - 1] > itemInsert:
        myList[position] = myList[position - 1]
        position -= 1

    # Once finished shifting items, insert current value
    # into sorted position
    myList[position] = itemInsert

print(myList)
```

Output:

```
[85, 24, 63, 45, 17, 31, 96, 50]
[24, 85, 63, 45, 17, 31, 96, 50]
[24, 63, 85, 45, 17, 31, 96, 50]
[24, 45, 63, 85, 17, 31, 96, 50]
[17, 24, 45, 63, 85, 31, 96, 50]
[17, 24, 31, 45, 63, 85, 96, 50]
[17, 24, 31, 45, 63, 85, 96, 50]
[17, 24, 31, 45, 50, 63, 85, 96]
```



# Sorting Algorithms



File Edit View Insert Format Slide Arrange Tools Add-ons Help Last edit was...



Present



Share



+ [Icons] Background Layout Theme Transition

13

## Bubble Sort



14

## Bubble Sort

A bubble sort moves through the list like a bubble, comparing two items at a time. The bubble then moves into the next location. For each bubble step, the two numbers in that bubble are compared. If the second item is smaller than the first then the two items are swapped. The list is not sorted after one pass. It repeats until the whole list is sorted.



15

## Bubble Sort - Example

## Bubble Sort

Click to add text



Click to add speaker notes



Insertion Sort.webm

Activate Windows

Go to Settings to activate Windows.

Show all



ENG

19:37

20/04/2020

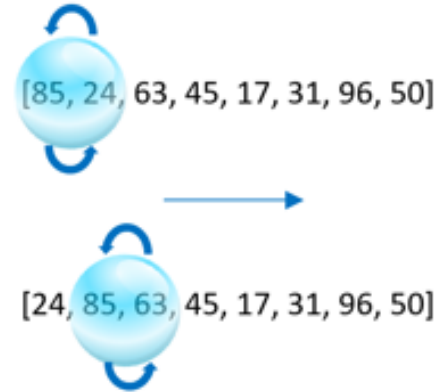
# Bubble Sort

A bubble sort iterates through the list like a bubble, examining two items at a time.

The bubble then moves onto the next two items. For each bubble step, the two numbers in that bubble are compared. If the second item is smaller than the first, then the two items are swapped.

The list is not sorted after one pass.

It repeats until the whole list is sorted.



# Bubble Sort - Example

[https://en.wikipedia.org/wiki/Bubble\\_sort#/media/File:Bubble-sort-example-300px.gif](https://en.wikipedia.org/wiki/Bubble_sort#/media/File:Bubble-sort-example-300px.gif)

6 5 3 1 8 7 2 4



# Bubble sort

This shows the first pass of a bubble sort.

You may be lucky where the list is sorted after one pass. But for the worst case, the smallest value may be the last item in the list and several more passes are needed for it to bubble down the list.

Select the first two items in the list (bubble)

[85, 24] [63, 45, 17, 31, 96, 50]

If the second of the two items is smaller

then the first item in the bubble, swap them.

[24, 85] [63, 45, 17, 31, 96, 50]

move onto the next two items and repeat...

[24, 85, 63] [45, 17, 31, 96, 50] Swap

[24, 63, 85, 45] [17, 31, 96, 50] Swap

[24, 63, 45, 85, 17] [31, 96, 50] Swap

[24, 63, 45, 17, 85, 31] [96, 50] Swap

[24, 63, 45, 17, 31, 85, 96, 50] No Swap

[24, 63, 45, 17, 31, 85, 96, 50] Swap

[24, 63, 45, 17, 31, 85, 50, 96] First Pass

Repeat all steps (by the number of items in the list -1)

 = Current Bubble

 = Unsorted

# Bubble Sort

Therefore we need two loops.

The first outer loop iterates one less than the number of items in the list, which is the number of passes we need in the worst case scenario.

For each outer loop iteration, we have an inner loop (nested loop) which conducts the single bubble sort pass.

# Bubble Sort - Python Code

<https://repl.it/@suzannelinnane/Bubble-Sort>

```
myList = [85, 24, 63, 45, 17, 31, 96, 50]
```

```
for outerIndex in range(len(myList)-1):  
    print(myList)
```

```
    # Single pass
```

```
    for index in range(1, len(myList)):  
        if myList[index] < myList[index-1]:
```

```
            # If the second in the bubble is
```

```
            # less than the first, swap
```

```
            tempValue = myList[index]
```

```
            myList[index] = myList[index-1]
```

```
            myList[index-1] = tempValue
```

```
print(myList)
```

Output:

```
[85, 24, 63, 45, 17, 31, 96, 50]  
[24, 63, 45, 17, 31, 85, 50, 96]  
[24, 45, 17, 31, 63, 50, 85, 96]  
[24, 17, 31, 45, 50, 63, 85, 96]  
[17, 24, 31, 45, 50, 63, 85, 96]  
[17, 24, 31, 45, 50, 63, 85, 96]  
[17, 24, 31, 45, 50, 63, 85, 96]  
[17, 24, 31, 45, 50, 63, 85, 96]
```

# Bubble Sort

## Advantages:

- Easy to implement
- Performs well on a small list
- Elements are swapped in place without using additional temporary storage, so the space requirement is at a minimum.

## Disadvantages:

- Does not deal well with a list containing a huge number of items.

# Selection/Simple Sort

## Advantages:

- Performs well on a small list.
- In-place sorting algorithm, no additional temporary storage is required beyond what is needed to hold the original list.

## Disadvantages:

- Not efficient when dealing with a large list of items.
- Performance is easily influenced by the initial ordering of the items before the sorting process.

# Insertion Sort

## Advantages:

- Easy to implement
- Performs well when dealing with a small list.
- In-place sorting algorithm so the space requirement is minimal.

## Disadvantages:

- Not efficient when dealing with a large list of items.