IT1312

Data Structures & Algorithms

Topic 04: Sort

version 1.0

(Content adapted from – Data Structures & Algorithms using Python. Rance D. Necaise, Wiley, 1st Edition, 2011.)

Sorting Algorithms

- Sorting is the process of **arranging or ordering** a collection of items such that each item and its successor satisfy a prescribed relationship (e.g. in ascending order, chronological order etc.).
- The ordering of the items is based on the value of a **sort key**.

Sorting Algorithms

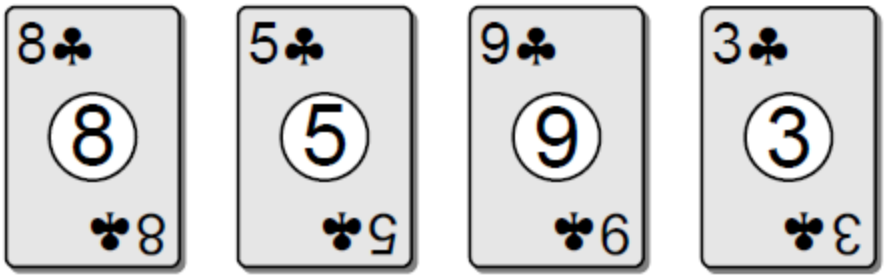
- Basic Sort
  - Bubble Sort
  - Selection Sort
  - Insertion Sort
- Advanced Sort (Topic 06)
  - Quick Sort
  - Merge Sort

Bubble Sort

- A simple solution to the sorting problem, which re-arranges the values by iterating over the list multiple times, causing larger values to bubble to the top or end of the list.

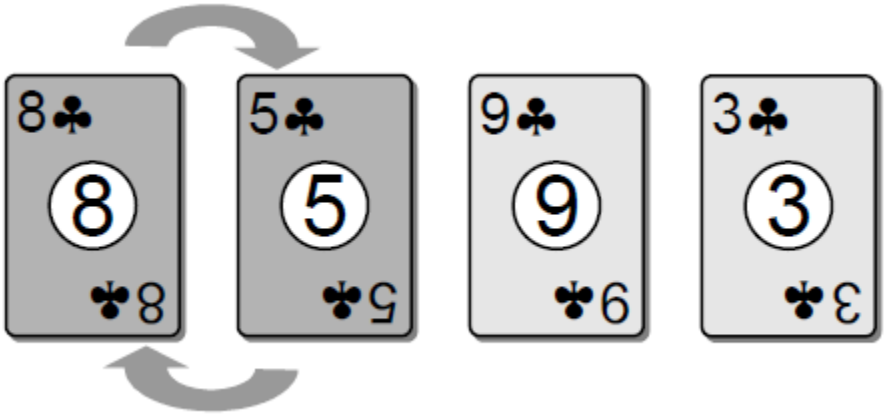
Bubble Sort

- Let's say we have four playing cards that we want to order from smallest to largest face value.
- The algorithm requires **multiple passes** over the cards.
- Each pass starting at the first card and ending one card earlier than on the previous iteration.



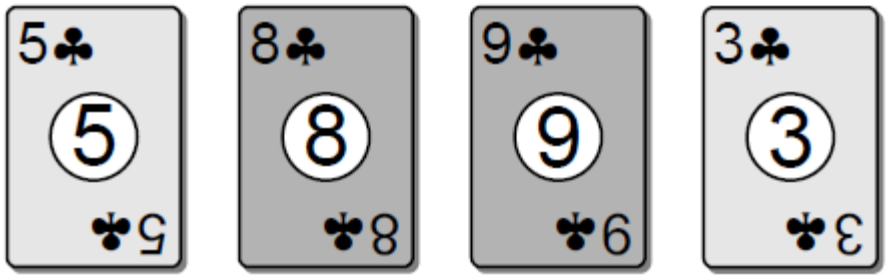
Bubble Sort

- During each pass, the cards in the first and second positions are compared.
- If the first is larger than the second, the two cards are swapped.



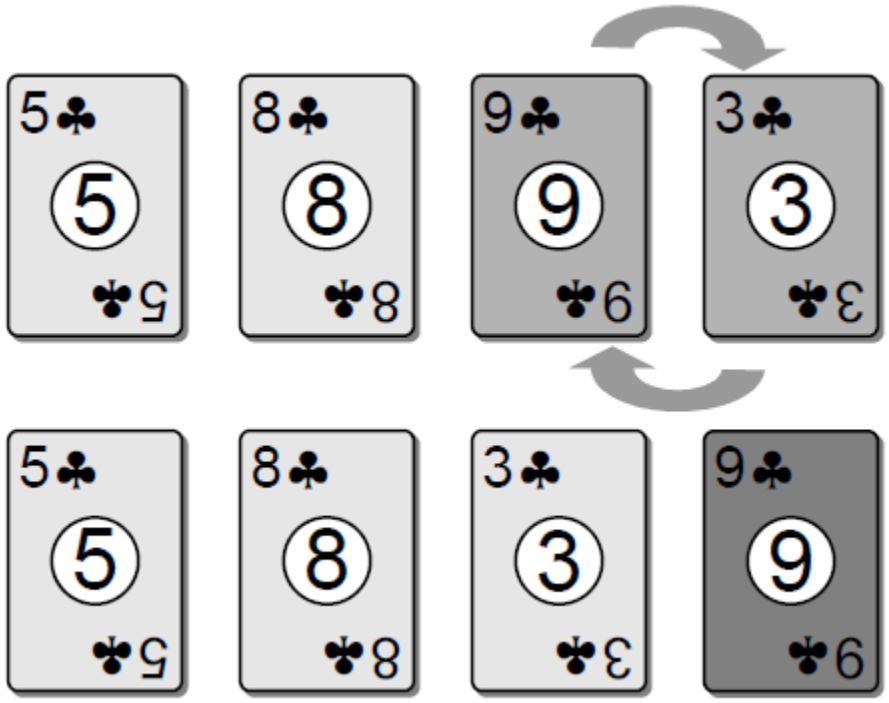
Bubble Sort

- Next, the cards in positions two and three are compared.
- If the first one is larger than the second, they are swapped.
- Otherwise, we leave them as they were.



Bubble Sort

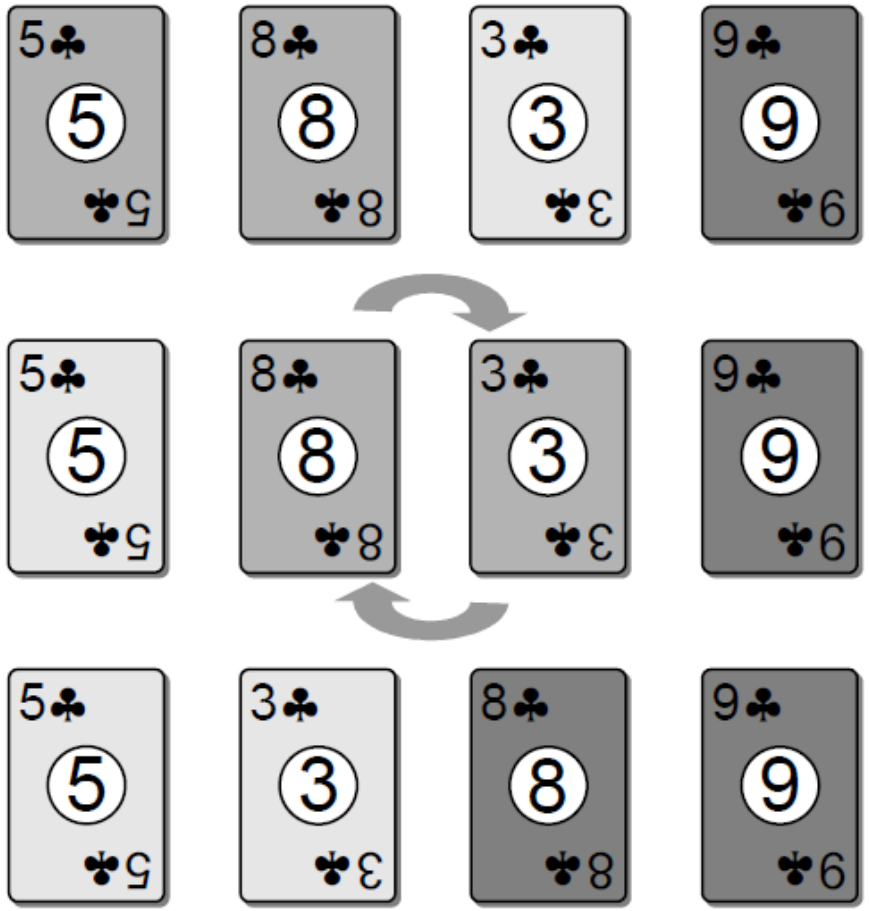
- This process continues for each successive pair of cards until the card with the largest face value is positioned at the end.



End of Pass 1

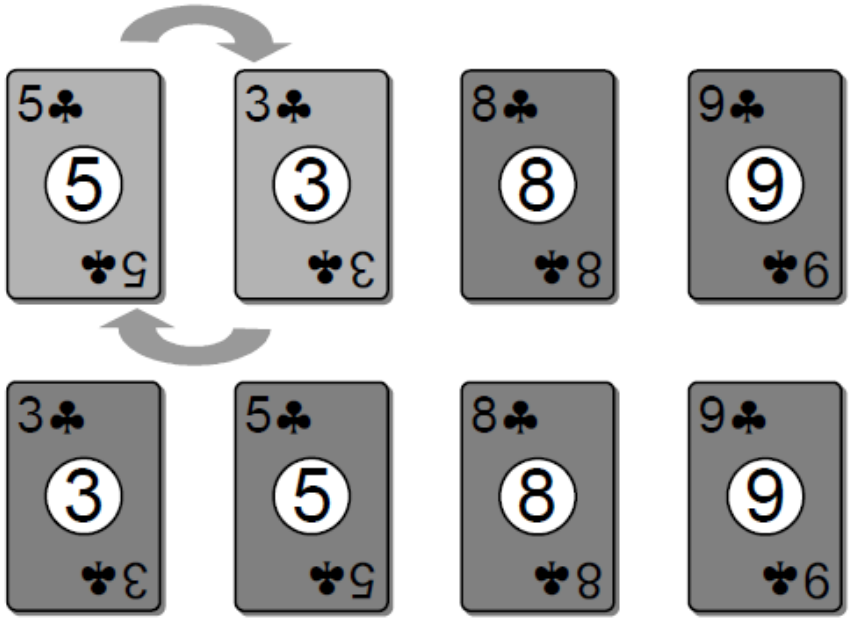
Bubble Sort

- In the **second pass**, the card with the second largest value is positioned in the next-to-last position.



Bubble Sort

- In the **third and final pass**, the first two cards will be positioned correctly.
- All the cards are now in their proper order.



Bubble Sort

```
# Sorts a sequence in ascending order using
# the bubble sort algorithm
def bubbleSort( theSeq ):
    n = len( theSeq )

    # Perform n-1 bubble operations on the sequence
    for i in range(n - 1, 0, -1):

        # Bubble the largest item to the end
        for j in range(i):
            if theSeq[j] > theSeq[j + 1]:
                # Swap the j and j+1 items
                tmp = theSeq[j]
                theSeq[j] = theSeq[j + 1]
                theSeq[j + 1] = tmp
```

Bubble Sort

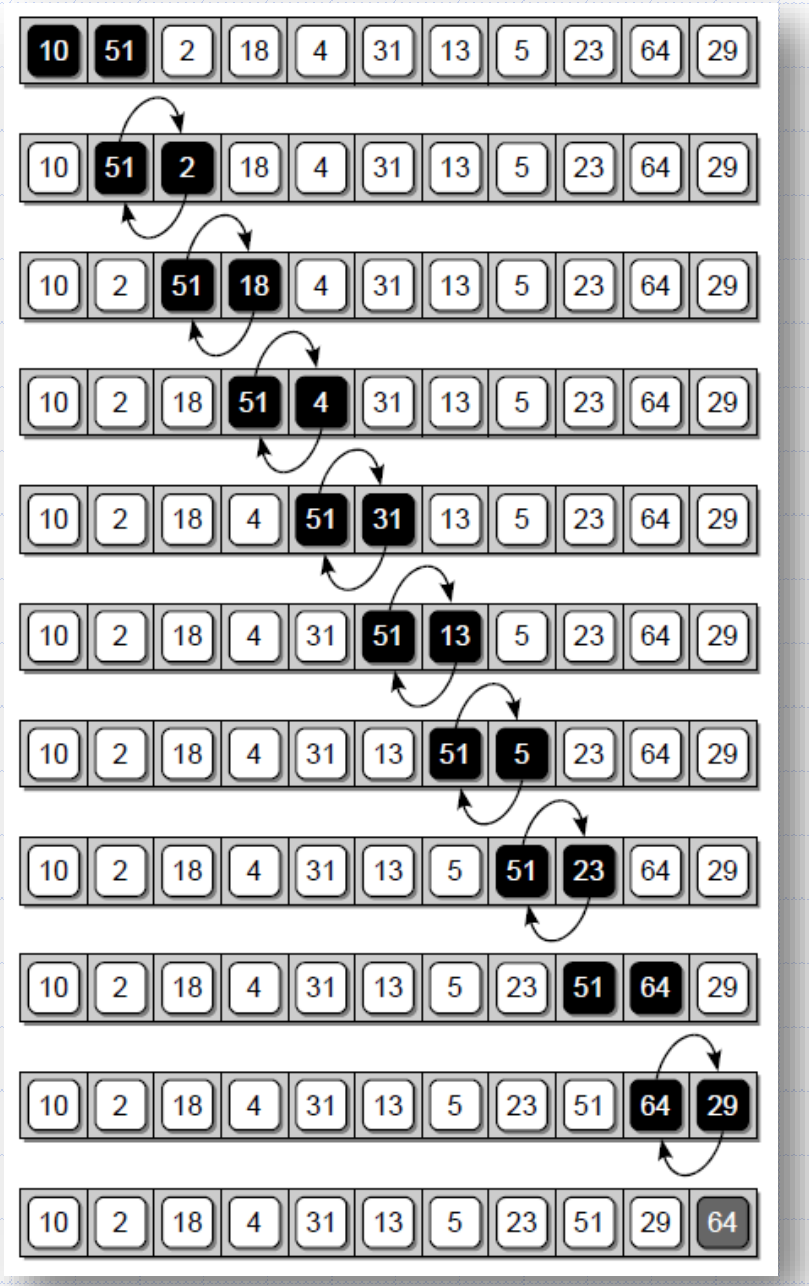
```
# Test codes
list_of_numbers = [10, 51, 2, 18, 4, 31, 13, 5, 23, 64, 29]

print(list_of_numbers)
bubbleSort(list_of_numbers)
print(list_of_numbers)
```

Output:

```
[10, 51, 2, 18, 4, 31, 13, 5, 23, 64, 29]
[2, 4, 5, 10, 13, 18, 23, 29, 31, 51, 64]

Process finished with exit code 0
```

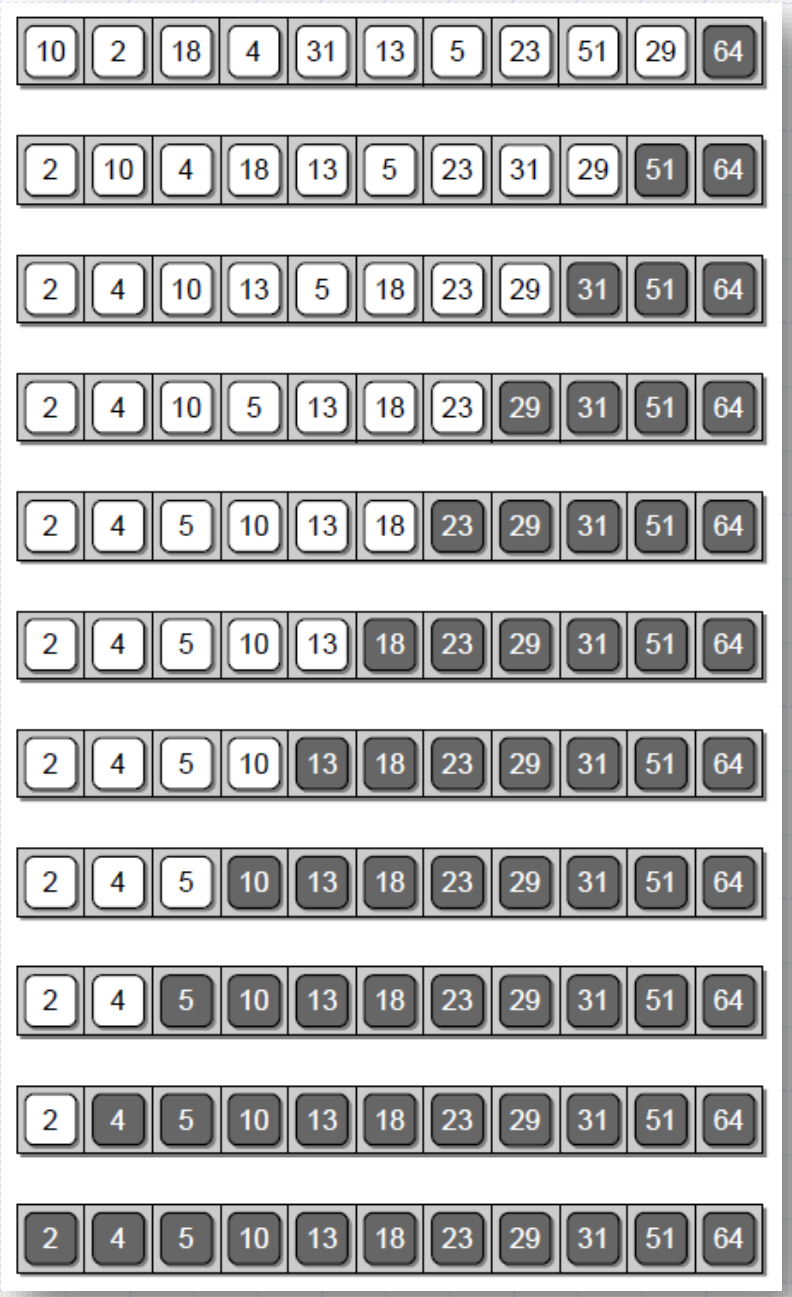
Bubble Sort

- First complete pass – places 64 in its correct position:
  - **Black boxes** – values being compared.
  - **Arrows** – indicate exchanges.



Bubble Sort

- Result of applying the bubble sort algorithm:
  - **Grey boxes** – values that are in order after each outer-loop traversal.



Selection Sort

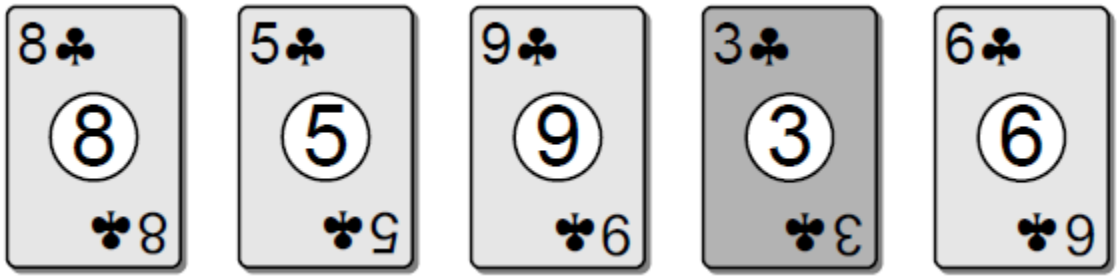
- Another sorting algorithm that improves on the Bubble Sort:
  - Like Bubble Sort, it makes multiple passes over the sequence.
  - However, **Selection Sort makes a single swap after each pass**.

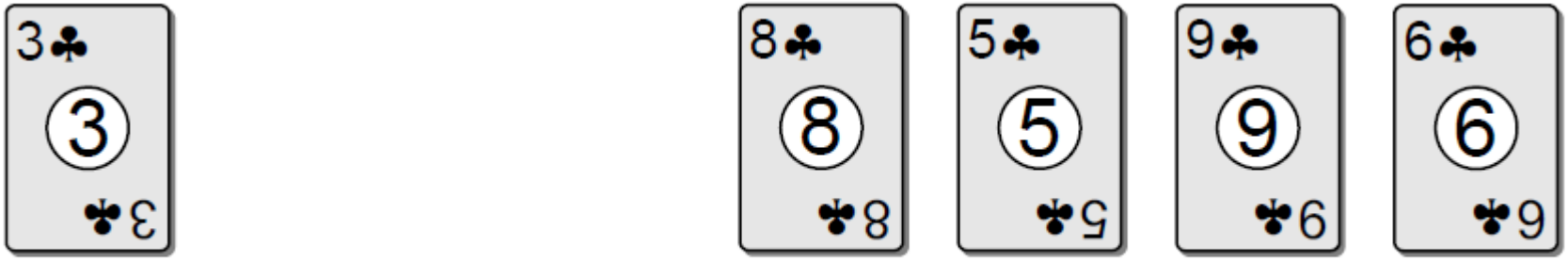
Selection Sort

- Let's say we have five playing cards on the table that are to be sorted in ascending order.

Selection Sort

- Scan through the cards and select the smallest from among those and place it in our hand.



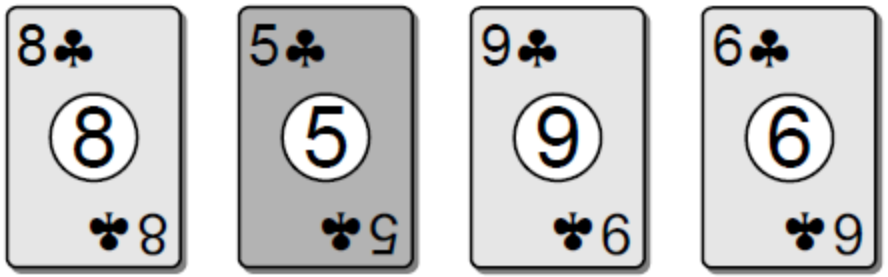


our hand

cards on the table

Selection Sort

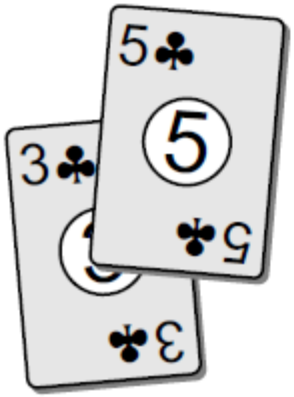
- Repeat the process and identify 5 as the next smallest face value.



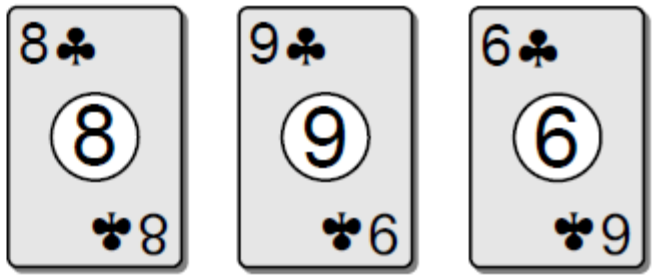
cards on the table

Selection Sort

- Pick up the 5 and **add it to the proper sorted position** in our hand.



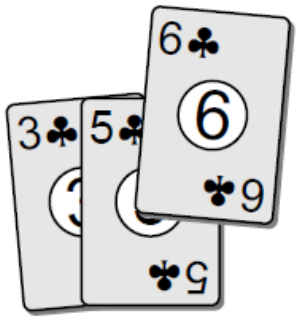
our hand



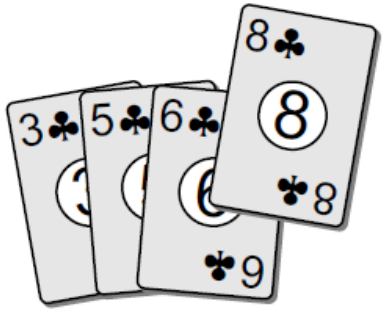
cards on the table

Selection Sort

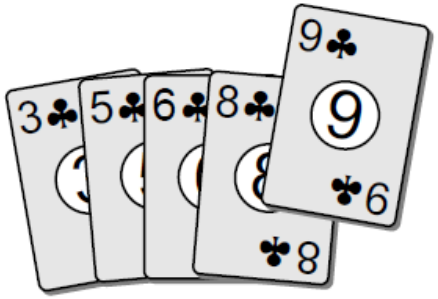
- This process is continued until all the cards have been picked and placed in our hand in the correct sorted order.



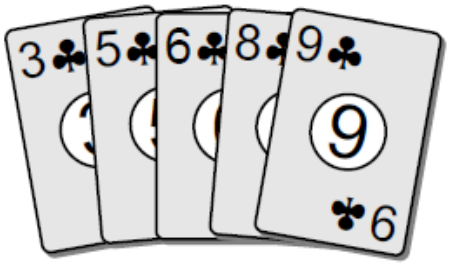
pick up the next smallest card (6)



pick up the next smallest card (8)



pickup the last card (9)



the resulting hand

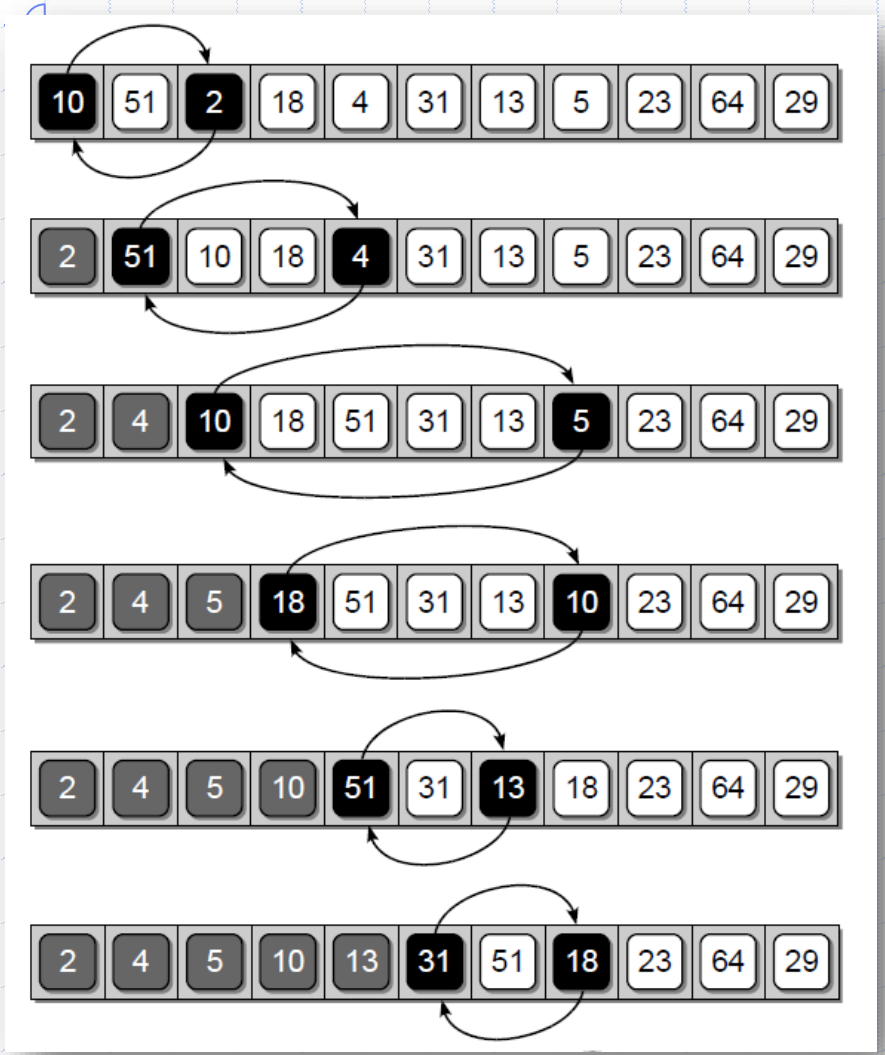
Selection Sort

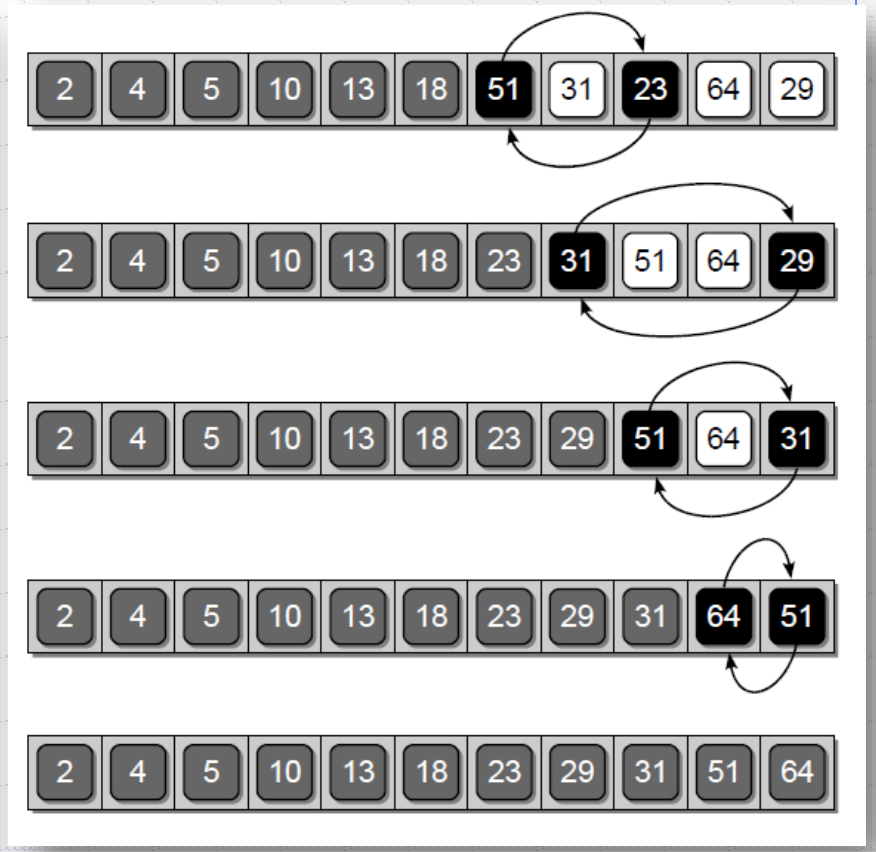
```
# Sort a sequence in ascending order using
# the selection sort algorithm
def selectionSort( theSeq ):
    n = len( theSeq )

    for i in range(n - 1):
        # Assume the ith element is the smallest.
        smallNdx = I

        # Determine if any other element contains a smaller value.
        for j in range(i+1, n):
            if theSeq[j] < theSeq[smallNdx]:
                smallNdx = j

        # Swap the ith value and smallNdx value only if the smallest
        # value is not already in its proper position.
        if smallNdx != i:
            tmp = theSeq[i]
            theSeq[i] = theSeq[smallNdx]
            theSeq[smallNdx] = tmp
```

Selection Sort

- Result of applying the selection sort algorithm:
  - **Grey boxes** – values that have been sorted.
  - **Black boxes** – values that are swapped during an iteration.

Selection Sort





Selection Sort – Activity

Exam Qn 2018/19 S1

- Trace each pass of the sorting process:

Pass	38	99	21	87	68	6	52	17	74	28
1	6	99	21	87	68	38	52	17	74	28
2	**6**	17	21	87	68	38	52	99	74	28
3	**6**	**17**	21	87	68	38	52	99	74	28
4	**6**	**17**	**21**	28	68	38	52	99	74	87
5	**6**	**17**	**21**	**28**	38	68	52	99	74	87
6	**6**	**17**	**21**	**28**	**38**	52	68	99	74	87
7	**6**	**17**	**21**	**28**	**38**	**52**	68	99	74	87
8	**6**	**17**	**21**	**28**	**38**	**52**	**68**	74	99	87
9	**6**	**17**	**21**	**28**	**38**	**52**	**68**	**74**	87	99
RESULT	**6**	**17**	**21**	**28**	**38**	**52**	**68**	**74**	**87**	**99**

Insertion Sort

- The insertion sort maintains a collection of sorted items and a collection of items to be sorted in the sequence.
- It starts by assuming the first item is in its proper position.
- Next, an iteration is performed over the remaining items so each value can be inserted into its proper position within the sorted portion of the sequence.

Insertion Sort

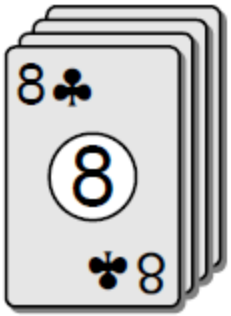
- Let's consider five cards stacked in a deck face up.

the deck

Insertion Sort

- We pick up the top card from the deck and place it in our hand.
- Since this is the first card, there is no decision to made as to its position.

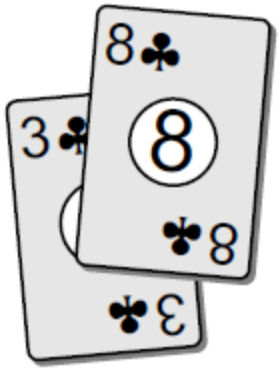
our hand



the deck

Insertion Sort

- Next, we again pick the top card from the stack and compare it to the card/s already in our hand and insert it into its proper sorted position.

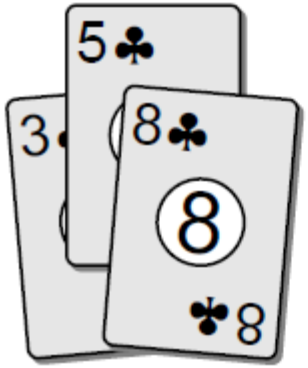


our hand

the deck

Insertion Sort

- The process is then repeated. This time, we pick up the 5 and find its position within our hand and insert it in the proper position.

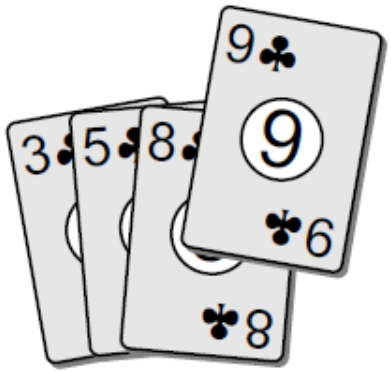


our hand

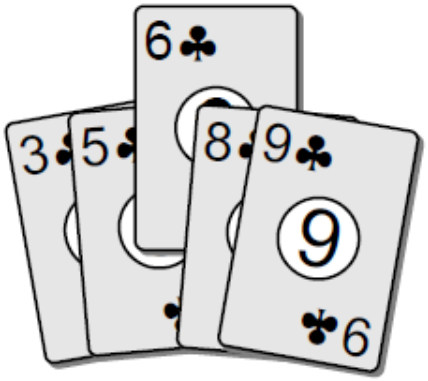
the deck

Insertion Sort

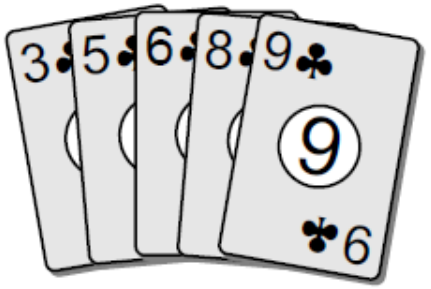
- This process continues, one card at a time, until all of the cards have been removed from the table and placed into our hand in their proper sorted



pick up the next card on top (9)



pick up the last card (6)



the resulting hand

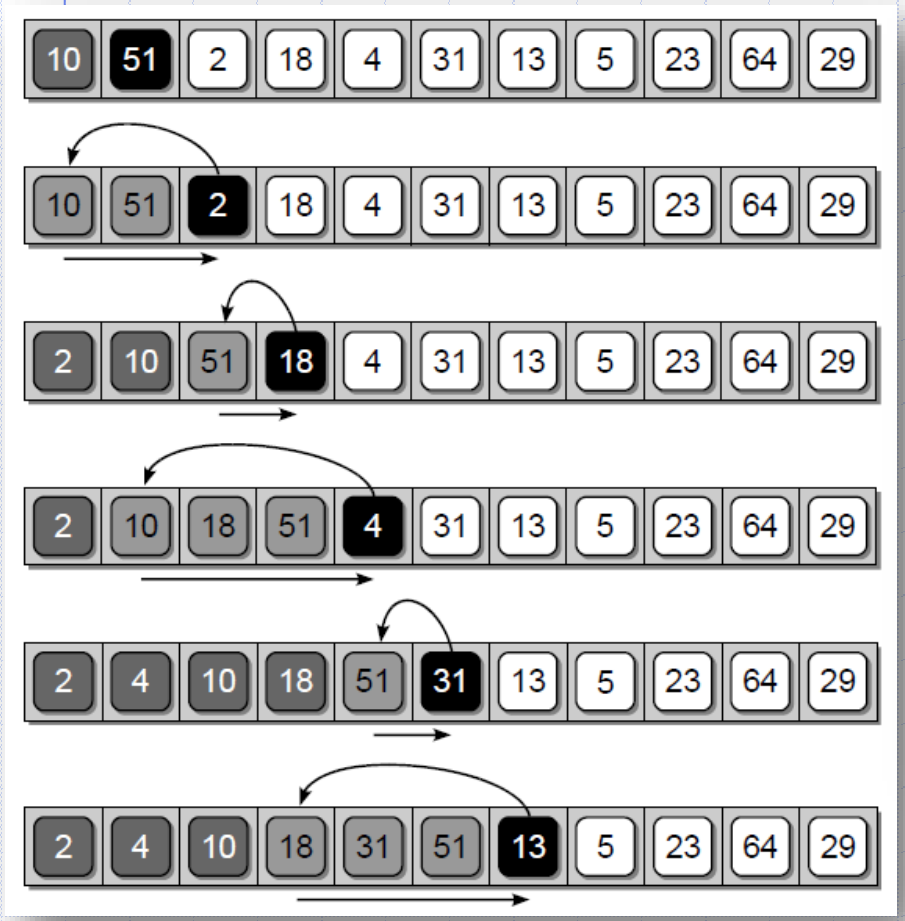
Insertion Sort

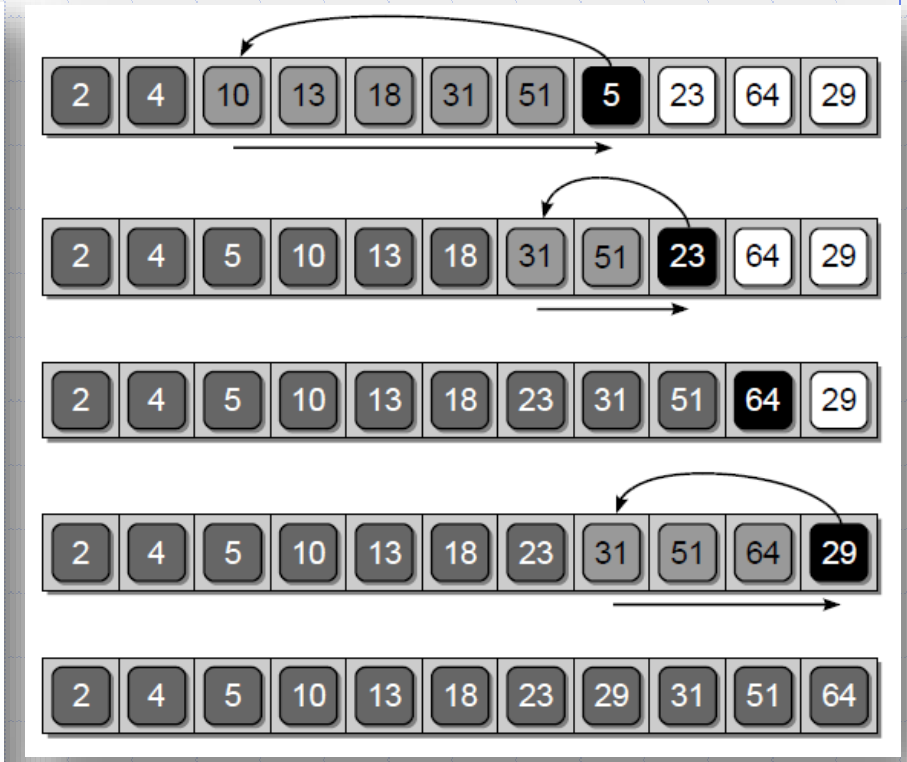
```
# Sorts a sequence in ascending order using
# the insertion sort algorithm
def insertionSort( theSeq ):
    n = len( theSeq )

    # Starts with the first item as the only sorted entry.
    for i in range(1, n):
        # Save the value to be positioned
        value = theSeq[i]

        # Find the position where value fits in the
        # ordered part of the list.
        pos = i
        while pos > 0 and value < theSeq[pos - 1]:
            # Shift the items to the right during the search
            theSeq[pos] = theSeq[pos-1]
            pos -= 1

        # Put the saved value into the open slot.
        theSeq[pos] = value
```

Insertion Sort

- Result of applying the insertion sort algorithm:
  - **Grey boxes** – values that have been sorted.
  - **Black boxes** – the next value to be positioned.
  - **Lighter Grey boxes with black text** – the sorted values that have to be shifted to the right to open a spot for the next value.

Insertion Sort





Insertion Sort – Activity

Exam Qn 2018/19 S1

- Trace each pass of the sorting process:

Pass	38	99	21	87	68	6	52	17	74	28
1	**38**	**99**	21	87	68	6	52	17	74	28
2	**21**	**38**	**99**	87	68	6	52	17	74	28
3	**21**	**38**	**87**	**99**	68	6	52	17	74	28
4	**21**	**38**	**68**	**87**	**99**	6	52	17	74	28
5	**6**	**21**	**38**	**68**	**87**	**99**	52	17	74	28
6	**6**	**21**	**38**	**52**	**68**	**87**	**99**	17	74	28
7	**6**	**17**	**21**	**38**	**52**	**68**	**87**	**99**	74	28
8	**6**	**17**	**21**	**38**	**52**	**68**	**74**	**87**	**99**	28
9	**6**	**17**	**21**	**28**	**38**	**52**	**68**	**74**	**87**	**99**
RESULT	**6**	**17**	**21**	**28**	**38**	**52**	**68**	**74**	**87**	**99**

https://docs.python.org/3/howto/sorting.html

Python HOW-TO – Sorting

- Python lists have a built-in `list.sort()` method that modifies the list in-place.
- There is also a `sorted()` built-in function that builds a new sorted list from an iterable.

```
>>> sorted([5, 2, 3, 1, 4])
[1, 2, 3, 4, 5]
```

Returns a new list

```
>>> a = [5, 2, 3, 1, 4]
>>> a.sort()
>>> a
[1, 2, 3, 4, 5]
```

Modifies the list in-place

```
>>> sorted({1: 'D', 2: 'B', 3: 'B', 4: 'E', 5: 'A'})
[1, 2, 3, 4, 5]
```

sorted() function accepts any iterable, including lists

https://docs.python.org/3/howto/sorting.html

Python HOW-TO – Sorting

- Both `list.sort()` and `sorted()` have a ***key*** parameter to specify a function to be called on each list element prior to making comparisons.
- The value of the ***key*** parameter should be a function that takes a single argument and returns a key to use for sorting purposes.

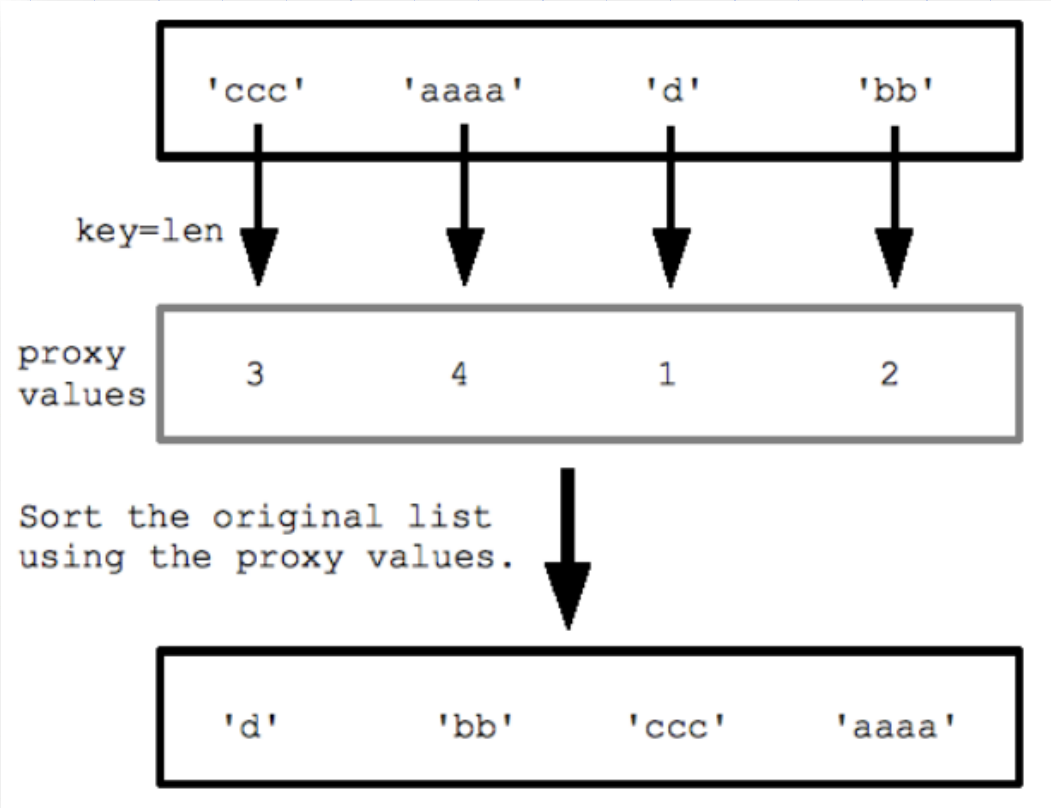
```
strs = ['ccc', 'aaaa', 'd', 'bb']
print sorted(strs, key=len)  ## ['d', 'bb', 'ccc', 'aaaa']
```

E.g. For a list of strings, specifying `key=len` (the built in `len()` function) sorts the strings by length, from shortest to longest.

Reference: https://developers.google.com/edu/python/sorting

Python HOW-TO – Sorting

```
strs = ['ccc', 'aaaa', 'd', 'bb']
print sorted(strs, key=len)  ## ['d', 'bb', 'ccc', 'aaaa']
```



Reference: https://developers.google.com/edu/python/sorting

Python HOW-TO – Sorting

- You can also pass in your own `MyFn` as the key function, like this:

```
## Say we have a list of strings we want to sort
## by the last letter of the string.
strs = ['xc', 'zb', 'yd' ,'wa']

## Write a little function that takes a string,
## and returns its last letter.
## This will be the key function (takes in 1 value,
## returns 1 value).
 def MyFn(s):
   return s[-1]

## Now pass key=MyFn to sorted() to sort by the last letter:
print sorted(strs, key=MyFn)  ## ['wa', 'zb', 'xc', 'yd']
```