UCONN
UNIVERSITY OF CONNECTICUT

Department of Computer Science and Engineering

Data Structures and Object-Oriented Design

(CSE – 2050)

Hasan Baig

Office: UConn (Stamford), 305C
email: hasan.baig@uconn.edu

Quick Recap

- Dynamic Programming
  - Bottom up iterative approach
- Studied LCS problem using recursion
- Search Algorithm
  - Binary search Algorithm
    - Works on sorted data

Quick Recap

- Binary Search Algorithm
  - Works on sorted data
- First implementation
  - Slicing causes O(n) time
- Improved implementation
  - Slicing removed

```
def BS(L, item):
    if len(L) == 0:
        return False
    mid_index = len(L) // 2
    if item == L[mid_index]:
        return True
    elif item < L[mid_index]:
        return BS(L[ : mid_index], item)
    else:
        return BS(L[mid_index + 1 : ], item)
```

```
def BS_improved(L, item, lower, upper):
    if lower > upper:
        return False
    else:
        mid_index = (lower + upper) // 2
        if item == L[mid_index]:
            return True
        elif item < L[mid_index]:
            return BS_improved(L, item, lower, mid_index - 1)
        else:
            return BS_improved(L, item, mid_index + 1, upper)
```

Binary Search Algorithm

Time Complexity

0th Iteration:
Data size = n

1st Iteration:
Data size = n/2 or $n/2^1$

2nd Iteration:
Data size = (n/2)/2 → n/4 or $n/2^2$

....

....

At K^{th} Iteration, the data size becomes 1:

$n/2^k = 1$

$n = 2^k$

$\log n = \log 2^k$

→ $O(\log n)$

How do you determine if the data is sorted?

Is item smaller than all the items on the right?

```
def is_sorted(L):
  for i in range(len(L)-1):
    for j in range(i+1,len(L)):
      if L[i] > L[j]:
        return False
   return True
```

O(?)

How do you determine if the data is sorted?

Is item smaller than all the items on the right?

```
def is_sorted(L):
  for i in range(len(L)-1):
    for j in range(i+1,len(L)):
      if L[i] > L[j]:
        return False
   return True
```

$O(n^2)$

How do you determine if the data is sorted?

Is item smaller than all the items on the right?

```
def is_sorted(L):
  for i in range(len(L)-1):
    for j in range(i+1,len(L)):
      if L[i] > L[j]:
        return False
   return True
```

$O(n^2)$

Is item smaller than its neighbor?

```
def is_sorted_better(L):
  for i in range(len(L)-1):
    if L[i] > L[i+1]:
      return False
  return True
```

$O(n)$

Sorting Algorithms

Algorithms to sort data

```
def bad_sort(L):
  for i in range(len(L) - 1):
    if L[i] > L[i+1]:                    #If two items are out of order
      L[i], L[i+1] = L[i+1], L[i]        #Switch them
```

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
  for i in range(len(L) - 1):
    if L[i] > L[i+1]:                    #If two items are out of order
      L[i], L[i+1] = L[i+1], L[i]        #Switch them
```

Example 1

```
data = [9, 5, 6, 7, 8]
bad_sort(data)
```

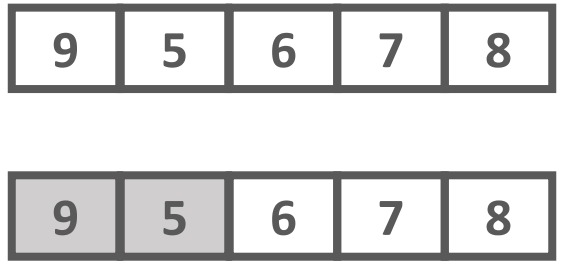
Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
  for i in range(len(L) - 1):
    if L[i] > L[i+1]:                    #If two items are out of order
      L[i], L[i+1] = L[i+1], L[i]        #Switch them
```

Example 1

```
data = [9, 5, 6, 7, 8]
bad_sort(data)
```



Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
  for i in range(len(L) - 1):
    if L[i] > L[i+1]:                    #If two items are out of order
      L[i], L[i+1] = L[i+1], L[i]        #Switch them
```

Example 1

```
data = [9, 5, 6, 7, 8]
bad_sort(data)
```

i=0

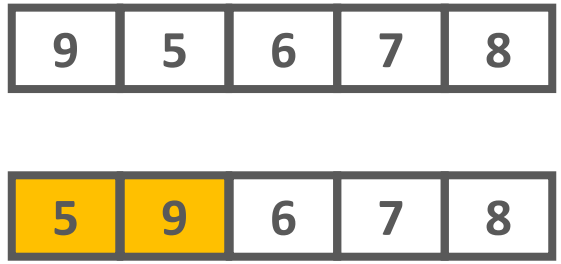
Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
  for i in range(len(L) - 1):
    if L[i] > L[i+1]:                    #If two items are out of order
      L[i], L[i+1] = L[i+1], L[i]        #Switch them
```

Example 1

```
data = [9, 5, 6, 7, 8]
bad_sort(data)
```



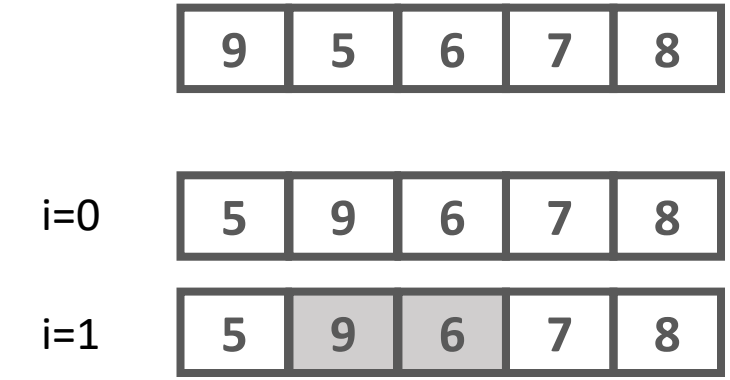
Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
  for i in range(len(L) - 1):
    if L[i] > L[i+1]:                    #If two items are out of order
      L[i], L[i+1] = L[i+1], L[i]        #Switch them
```

Example 1

```
data = [9, 5, 6, 7, 8]
bad_sort(data)
```



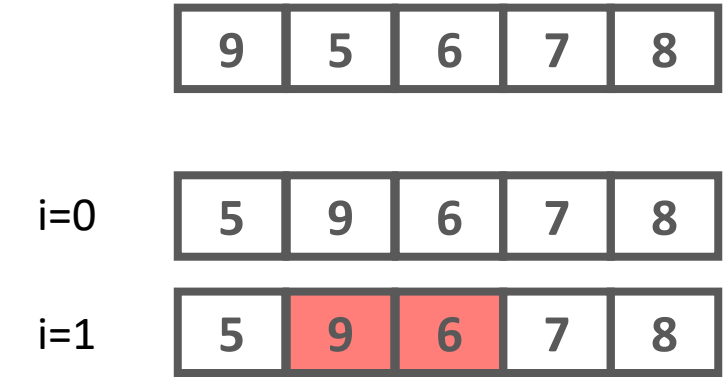
Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
  for i in range(len(L) - 1):
    if L[i] > L[i+1]:                    #If two items are out of order
      L[i], L[i+1] = L[i+1], L[i]        #Switch them
```

Example 1

```
data = [9, 5, 6, 7, 8]
bad_sort(data)
```



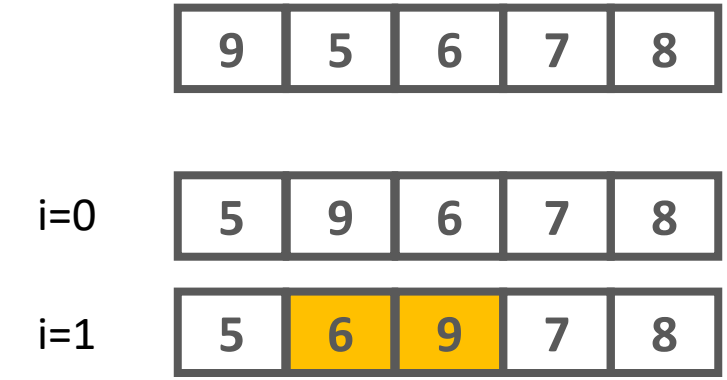
Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
  for i in range(len(L) - 1):
    if L[i] > L[i+1]:                    #If two items are out of order
      L[i], L[i+1] = L[i+1], L[i]        #Switch them
```

Example 1

```
data = [9, 5, 6, 7, 8]
bad_sort(data)
```



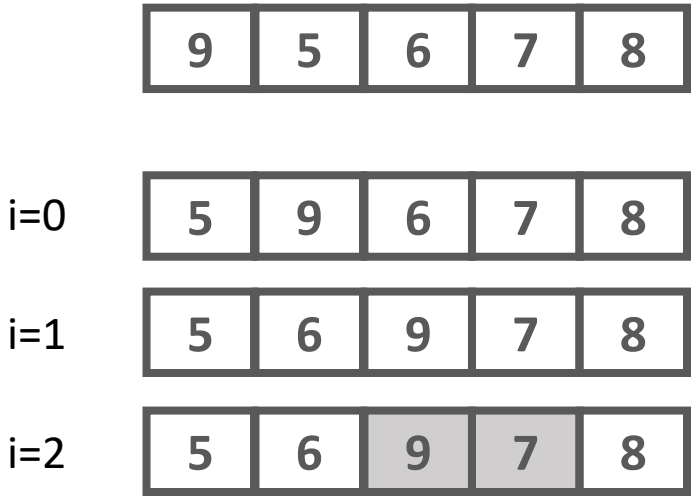
Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
  for i in range(len(L) - 1):
    if L[i] > L[i+1]:                    #If two items are out of order
      L[i], L[i+1] = L[i+1], L[i]        #Switch them
```

Example 1

```
data = [9, 5, 6, 7, 8]
bad_sort(data)
```



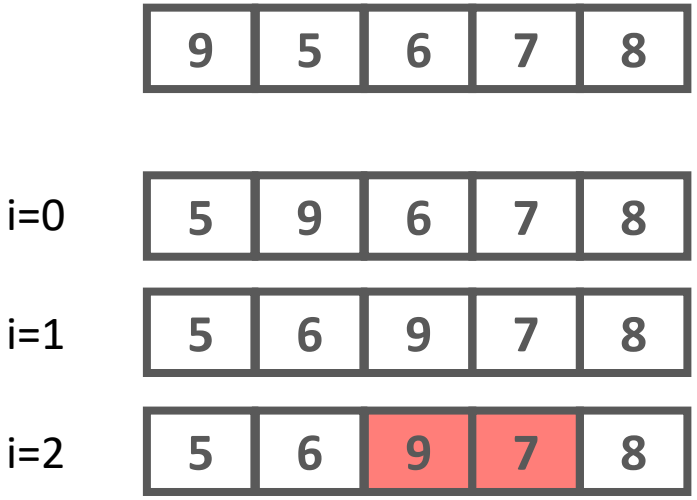
Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
  for i in range(len(L) - 1):
    if L[i] > L[i+1]:                    #If two items are out of order
      L[i], L[i+1] = L[i+1], L[i]        #Switch them
```

Example 1

```
data = [9, 5, 6, 7, 8]
bad_sort(data)
```



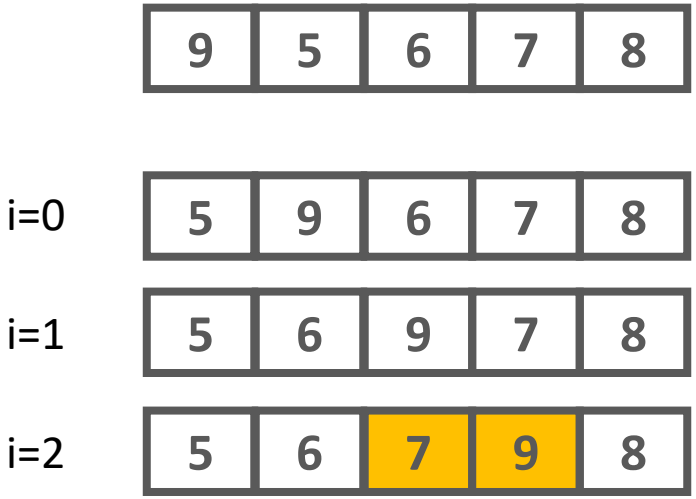
Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
  for i in range(len(L) - 1):
    if L[i] > L[i+1]:                    #If two items are out of order
      L[i], L[i+1] = L[i+1], L[i]        #Switch them
```

Example 1

```
data = [9, 5, 6, 7, 8]
bad_sort(data)
```



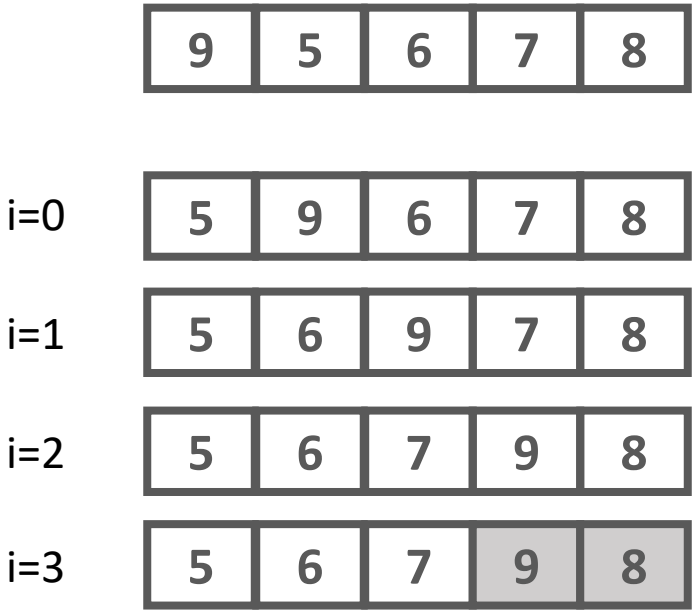
Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
  for i in range(len(L) - 1):
    if L[i] > L[i+1]:                    #If two items are out of order
      L[i], L[i+1] = L[i+1], L[i]        #Switch them
```

Example 1

```
data = [9, 5, 6, 7, 8]
bad_sort(data)
```



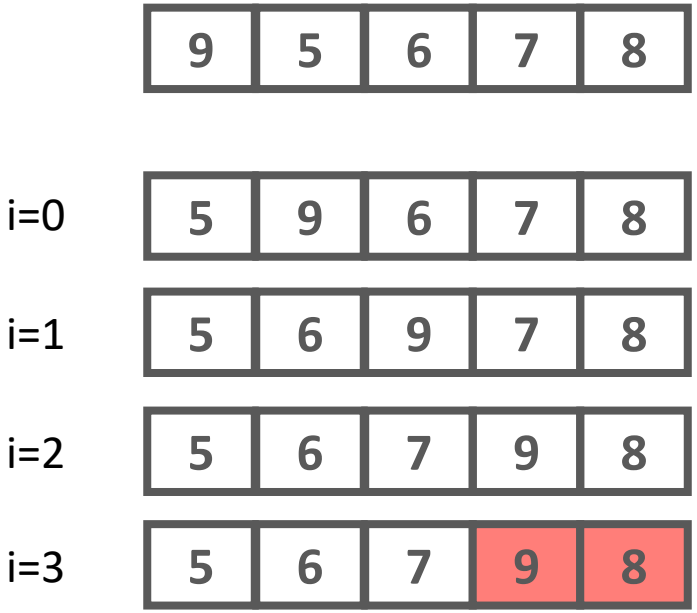
Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
  for i in range(len(L) - 1):
    if L[i] > L[i+1]:                    #If two items are out of order
      L[i], L[i+1] = L[i+1], L[i]        #Switch them
```

Example 1

```
data = [9, 5, 6, 7, 8]
bad_sort(data)
```



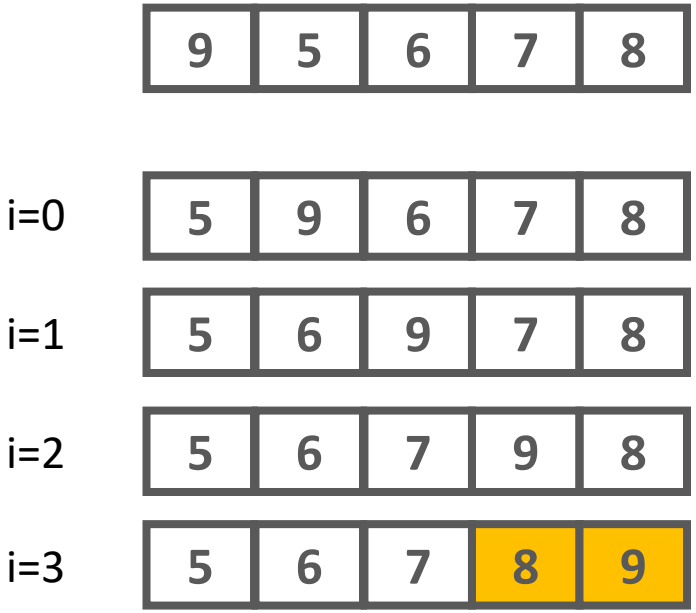
Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
  for i in range(len(L) - 1):
    if L[i] > L[i+1]:                    #If two items are out of order
      L[i], L[i+1] = L[i+1], L[i]        #Switch them
```

Example 1

```
data = [9, 5, 6, 7, 8]
bad_sort(data)
```



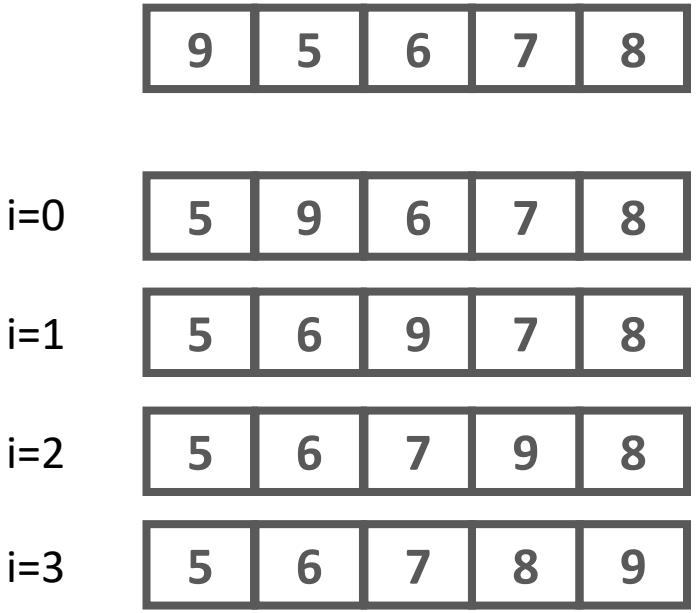
Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
  for i in range(len(L) - 1):
    if L[i] > L[i+1]:                    #If two items are out of order
      L[i], L[i+1] = L[i+1], L[i]        #Switch them
```

Example 1

```
data = [9, 5, 6, 7, 8]
bad_sort(data)
```



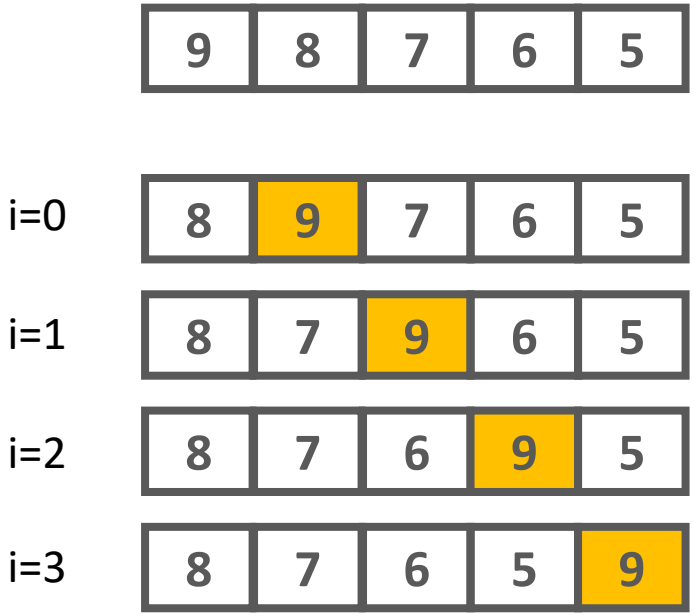
Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
  for i in range(len(L) - 1):
    if L[i] > L[i+1]:                    #If two items are out of order
      L[i], L[i+1] = L[i+1], L[i]        #Switch them
```

Example 2

```
data = [9, 8, 7, 6, 5]
bad_sort(data)
```



Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
  for i in range(len(L) - 1):
    if L[i] > L[i+1]:                    #If two items are out of order
      L[i], L[i+1] = L[i+1], L[i]        #Switch them
```

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1):
            if L[i] > L[i+1]:                    #If two items are out of order
                L[i], L[i+1] = L[i+1], L[i]      #Switch them
```

9	8	7	6	5

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1):
            if L[i] > L[i+1]:                    #If two items are out of order
                L[i], L[i+1] = L[i+1], L[i]      #Switch them
```

el = 0

	9	8	7	6	5
i = 0	9	8	7	6	5

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1):
            if L[i] > L[i+1]:                    #If two items are out of order
                L[i], L[i+1] = L[i+1], L[i]      #Switch them
```

el = 0

	9	8	7	6	5
i = 0	8	9	7	6	5

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1):
            if L[i] > L[i+1]:                    #If two items are out of order
                L[i], L[i+1] = L[i+1], L[i]      #Switch them
```

el = 0

	9	8	7	6	5
i = 0	8	9	7	6	5
i = 1	8	9	7	6	5

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1):
            if L[i] > L[i+1]:                    #If two items are out of order
                L[i], L[i+1] = L[i+1], L[i]      #Switch them
```

el = 0

	9	8	7	6	5
i = 0	8	9	7	6	5
i = 1	8	7	9	6	5

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1):
            if L[i] > L[i+1]:                    #If two items are out of order
                L[i], L[i+1] = L[i+1], L[i]      #Switch them
```

el = 0

	9	8	7	6	5
i = 0	8	9	7	6	5
i = 1	8	7	9	6	5
i = 2	8	7	9	6	5

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1):
            if L[i] > L[i+1]:                    #If two items are out of order
                L[i], L[i+1] = L[i+1], L[i]      #Switch them
```

el = 0

	9	8	7	6	5
i = 0	8	9	7	6	5
i = 1	8	7	9	6	5
i = 2	8	7	6	9	5

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1):
            if L[i] > L[i+1]:                    #If two items are out of order
                L[i], L[i+1] = L[i+1], L[i]      #Switch them
```

el = 0

	9	8	7	6	5
i = 0	8	9	7	6	5
i = 1	8	7	9	6	5
i = 2	8	7	6	9	5
i = 3	8	7	6	9	5

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1):
            if L[i] > L[i+1]:                    #If two items are out of order
                L[i], L[i+1] = L[i+1], L[i]      #Switch them
```

el = 0

	9	8	7	6	5
i = 0	8	9	7	6	5
i = 1	8	7	9	6	5
i = 2	8	7	6	9	5
i = 3	8	7	6	5	9

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1):
            if L[i] > L[i+1]:
                L[i], L[i+1] = L[i+1], L[i]    #If two items are out of order
                                               #Switch them
```

el = 0

9	8	7	6	5
8	9	7	6	5
8	7	9	6	5
8	7	6	9	5
8	7	6	5	9

el = 1

8	7	6	5	9

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1):
            if L[i] > L[i+1]:
                L[i], L[i+1] = L[i+1], L[i]
```

```
#If two items are out of order
#Switch them
```

el = 0 el = 1

i = 0

9	8	7	6	5
8	9	7	6	5
8	7	9	6	5
8	7	6	9	5
8	7	6	5	9

8	7	6	5	9
8	7	6	5	9

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1):
            if L[i] > L[i+1]:
                L[i], L[i+1] = L[i+1], L[i]
```

```
#If two items are out of order
#Switch them
```

el = 0 el = 1

i = 0

9	8	7	6	5
8	9	7	6	5
8	7	9	6	5
8	7	6	9	5
8	7	6	5	9

8	7	6	5	9
7	8	6	5	9

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1):
            if L[i] > L[i+1]:
                L[i], L[i+1] = L[i+1], L[i]
```

```
#If two items are out of order
#Switch them
```

el = 0 el = 1

	9	8	7	6	5	8	7	6	5	9
i = 0	8	9	7	6	5	7	8	6	5	9
i = 1	8	7	9	6	5	7	8	6	5	9
	8	7	6	9	5					
	8	7	6	5	9					

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1):
            if L[i] > L[i+1]:
                L[i], L[i+1] = L[i+1], L[i]
```

```
#If two items are out of order
#Switch them
```

el = 0 | el = 1

	9	8	7	6	5	8	7	6	5	9
i = 0	8	9	7	6	5	7	8	6	5	9
i = 1	8	7	9	6	5	7	6	8	5	9
	8	7	6	9	5					
	8	7	6	5	9					

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1):
            if L[i] > L[i+1]:
                L[i], L[i+1] = L[i+1], L[i]
```

```
#If two items are out of order
#Switch them
```

	el = 0					el = 1				
	9	8	7	6	5	8	7	6	5	9
i = 0	8	9	7	6	5	7	8	6	5	9
i = 1	8	7	9	6	5	7	6	8	5	9
i = 2	8	7	6	9	5	7	6	8	5	9
	8	7	6	5	9					

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1):
            if L[i] > L[i+1]:
                L[i], L[i+1] = L[i+1], L[i]
```

```
#If two items are out of order
#Switch them
```

	el = 0					el = 1				
	9	8	7	6	5	8	7	6	5	9
i = 0	8	9	7	6	5	7	8	6	5	9
i = 1	8	7	9	6	5	7	6	8	5	9
i = 2	8	7	6	9	5	7	6	5	8	9
	8	7	6	5	9					

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1):
            if L[i] > L[i+1]:
                L[i], L[i+1] = L[i+1], L[i]
```

```
#If two items are out of order
#Switch them
```

	el = 0					el = 1				
	9	8	7	6	5	8	7	6	5	9
i = 0	8	9	7	6	5	7	8	6	5	9
i = 1	8	7	9	6	5	7	6	8	5	9
i = 2	8	7	6	9	5	7	6	5	8	9
i = 3	8	7	6	5	9	7	6	5	8	9

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1):
            if L[i] > L[i+1]:
                L[i], L[i+1] = L[i+1], L[i]
```

```
#If two items are out of order
#Switch them
```

	el = 0					el = 1				
	9	8	7	6	5	8	7	6	5	9
i = 0	8	9	7	6	5	7	8	6	5	9
i = 1	8	7	9	6	5	7	6	8	5	9
i = 2	8	7	6	9	5	7	6	5	8	9
i = 3	8	7	6	5	9	7	6	5	8	9

Is this comparison needed?

Sorting Algorithms

First approach of sorting the data

```
def bad_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1):
            if L[i] > L[i+1]:                    #If two items are out of order
                L[i], L[i+1] = L[i+1], L[i]      #Switch them
```

	el = 0					el = 1				
	9	8	7	6	5	8	7	6	5	9
i = 0	8	9	7	6	5	7	8	6	5	9
i = 1	8	7	9	6	5	7	6	8	5	9
i = 2	8	7	6	9	5	7	6	5	8	9
i = 3	8	7	6	5	9	7	6	5	8	9

Is this comparison needed?
No! because the last item has already been placed at the right place in previous iteration.

Sorting Algorithms

First approach of sorting the data

```
1 def bad_sort(L):
2     for el in range(len(L) - 1):
3         for i in range(len(L) - 1):
4             if L[i] > L[i+1]:                #If two items are out of order
5                 L[i], L[i+1] = L[i+1], L[i]  #Switch them
```

	el = 0					el = 1				
	9	8	7	6	5	8	7	6	5	9
i = 0	8	9	7	6	5	7	8	6	5	9
i = 1	8	7	9	6	5	7	6	8	5	9
i = 2	8	7	6	9	5	7	6	5	8	9
i = 3	8	7	6	5	9	7	6	5	8	9

What update in the code will avoid making this comparison?

Sorting Algorithms

First approach of sorting the data

```
1 def bad_sort(L):
2     for el in range(len(L) - 1):
3         for i in range(len(L) - 1):
4             if L[i] > L[i+1]:                #If two items are out of order
5                 L[i], L[i+1] = L[i+1], L[i]  #Switch them
```

	el = 0					el = 1				
	9	8	7	6	5	8	7	6	5	9
i = 0	8	9	7	6	5	7	8	6	5	9
i = 1	8	7	9	6	5	7	6	8	5	9
i = 2	8	7	6	9	5	7	6	5	8	9
i = 3	8	7	6	5	9	7	6	5	8	9

What update in the code will avoid making this comparison?

Sorting Algorithms

First approach of sorting the data

```
1 def bad_sort(L):
2     for el in range(len(L) - 1):
3         for i in range(len(L) - 1 - el):
4             if L[i] > L[i+1]:                    #If two items are out of order
5                 L[i], L[i+1] = L[i+1], L[i]      #Switch them
```

	el = 0					el = 1				
	9	8	7	6	5	8	7	6	5	9
i = 0	8	9	7	6	5	7	8	6	5	9
i = 1	8	7	9	6	5	7	6	8	5	9
i = 2	8	7	6	9	5	7	6	5	8	9
i = 3	8	7	6	5	9	7	6	5	8	9

Sorting Algorithms | Bubble Sort

Such an algorithm is called Bubble sort algorithm

```
1 def bubble_sort(L):
2     for el in range(len(L) - 1):
3         for i in range(len(L) - 1 - el):
4             if L[i] > L[i+1]:                    #If two items are out of order
5                 L[i], L[i+1] = L[i+1], L[i]      #Switch them
```

	el = 0					el = 1				
	9	8	7	6	5	8	7	6	5	9
i = 0	8	9	7	6	5	7	8	6	5	9
i = 1	8	7	9	6	5	7	6	8	5	9
i = 2	8	7	6	9	5	7	6	5	8	9
i = 3	8	7	6	5	9	7	6	5	8	9

Sorting Algorithms | Bubble Sort

Such an algorithm is called Bubble sort algorithm

```
1 def bubble_sort(L):
2     for el in range(len(L) - 1):
3         for i in range(len(L) - 1 - el):
4             if L[i] > L[i+1]:                    #If two items are out of order
5                 L[i], L[i+1] = L[i+1], L[i]      #Switch them
```

el = 0

9	8	7	6	5
8	9	7	6	5
8	7	9	6	5
8	7	6	9	5
8	7	6	5	9

el = 1

8	7	6	5	9
7	8	6	5	9
7	6	8	5	9
7	6	5	8	9

el = 2

7	6	5	8	9

Sorting Algorithms | Bubble Sort

Such an algorithm is called Bubble sort algorithm

```
1 def bubble_sort(L):
2     for el in range(len(L) - 1):
3         for i in range(len(L) - 1 - el):
4             if L[i] > L[i+1]:                    #If two items are out of order
5                 L[i], L[i+1] = L[i+1], L[i]      #Switch them
```

	el = 0					el = 1					el = 2				
	9	8	7	6	5	8	7	6	5	9	7	6	5	8	9
i = 0	8	9	7	6	5	7	8	6	5	9	7	6	5	8	9
	8	7	9	6	5	7	6	8	5	9					
	8	7	6	9	5	7	6	5	8	9					
	8	7	6	5	9										

Sorting Algorithms | Bubble Sort

Such an algorithm is called Bubble sort algorithm

```
1 def bubble_sort(L):
2     for el in range(len(L) - 1):
3         for i in range(len(L) - 1 - el):
4             if L[i] > L[i+1]:                    #If two items are out of order
5                 L[i], L[i+1] = L[i+1], L[i]      #Switch them
```

Sorting Algorithms | Bubble Sort

Such an algorithm is called Bubble sort algorithm

```
1 def bubble_sort(L):
2     for el in range(len(L) - 1):
3         for i in range(len(L) - 1 - el):
4             if L[i] > L[i+1]:                    #If two items are out of order
5                 L[i], L[i+1] = L[i+1], L[i]      #Switch them
```

Sorting Algorithms | Bubble Sort

Such an algorithm is called Bubble sort algorithm

```
1 def bubble_sort(L):
2     for el in range(len(L) - 1):
3         for i in range(len(L) - 1 - el):
4             if L[i] > L[i+1]:                    #If two items are out of order
5                 L[i], L[i+1] = L[i+1], L[i]      #Switch them
```

	el = 0					el = 1					el = 2				
	9	8	7	6	5	8	7	6	5	9	7	6	5	8	9
i = 0	8	9	7	6	5	7	8	6	5	9	6	7	5	8	9
i = 1	8	7	9	6	5	7	6	8	5	9	6	5	7	8	9
	8	7	6	9	5	7	6	5	8	9					
	8	7	6	5	9										

Sorting Algorithms | Bubble Sort

Such an algorithm is called Bubble sort algorithm

```
1 def bubble_sort(L):
2     for el in range(len(L) - 1):
3         for i in range(len(L) - 1 - el):
4             if L[i] > L[i+1]:                    #If two items are out of order
5                 L[i], L[i+1] = L[i+1], L[i]      #Switch them
```

el = 0

9	8	7	6	5
8	9	7	6	5
8	7	9	6	5
8	7	6	9	5
8	7	6	5	9

el = 1

8	7	6	5	9
7	8	6	5	9
7	6	8	5	9
7	6	5	8	9

el = 2

7	6	5	8	9
6	7	5	8	9
6	5	7	8	9

el = 3

6	5	7	8	9

Sorting Algorithms | Bubble Sort

Such an algorithm is called Bubble sort algorithm

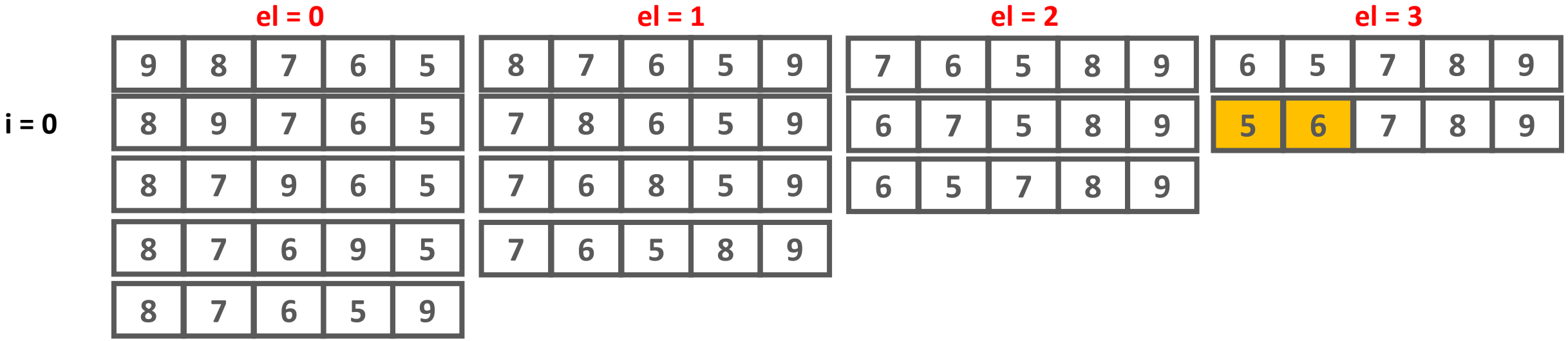
```
1 def bubble_sort(L):
2     for el in range(len(L) - 1):
3         for i in range(len(L) - 1 - el):
4             if L[i] > L[i+1]:                  #If two items are out of order
5                 L[i], L[i+1] = L[i+1], L[i]    #Switch them
```

	el = 0	el = 1	el = 2	el = 3
i = 0	9 8 7 6 5	8 7 6 5 9	7 6 5 8 9	6 5 7 8 9
	8 9 7 6 5	7 8 6 5 9	6 7 5 8 9	6 5 7 8 9
	8 7 9 6 5	7 6 8 5 9	6 5 7 8 9	
	8 7 6 9 5	7 6 5 8 9		
	8 7 6 5 9			

Sorting Algorithms | Bubble Sort

Such an algorithm is called Bubble sort algorithm

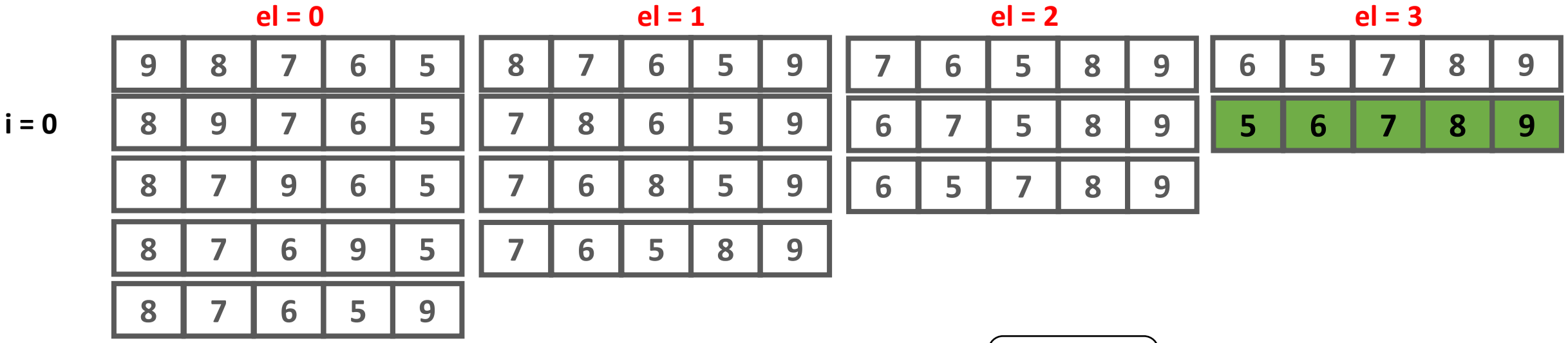
```
1 def bubble_sort(L):
2     for el in range(len(L) - 1):
3         for i in range(len(L) - 1 - el):
4             if L[i] > L[i+1]:                  #If two items are out of order
5                 L[i], L[i+1] = L[i+1], L[i]    #Switch them
```



Sorting Algorithms | Bubble Sort

Such an algorithm is called Bubble sort algorithm

```
1 def bubble_sort(L):
2     for el in range(len(L) - 1):
3         for i in range(len(L) - 1 - el):
4             if L[i] > L[i+1]:                  #If two items are out of order
5                 L[i], L[i+1] = L[i+1], L[i]    #Switch them
```



$O(n^2)$

Sorting Algorithms | Selection Sort

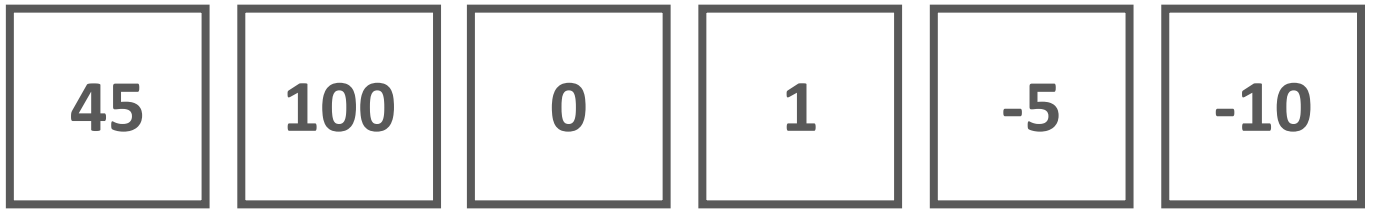
- Selection sort algorithm is another approach of sorting data in **$O(n^2)$** quadratic running time
- We can either find the smallest item and place it in the beginning

OR

- Find the biggest item and move it to end

Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

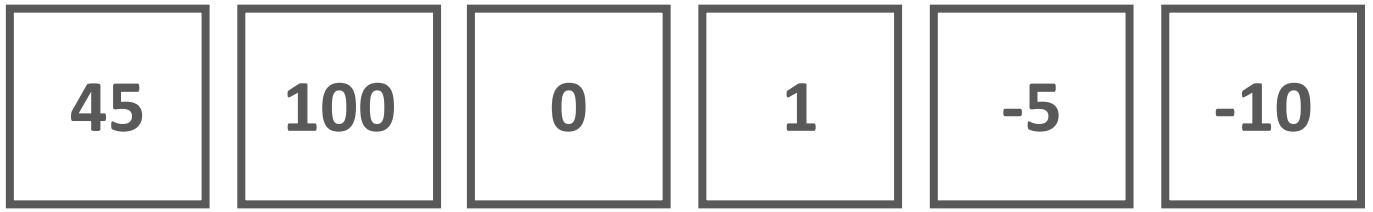


Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

- Assume that smallest element is initially placed at index 0
- Record the index of smallest element

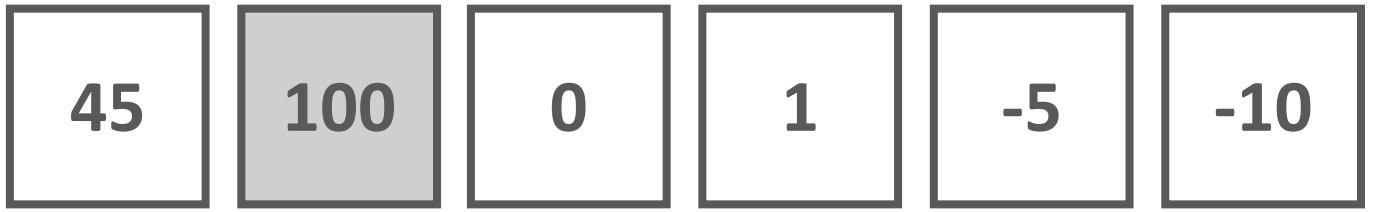
idx = 0



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

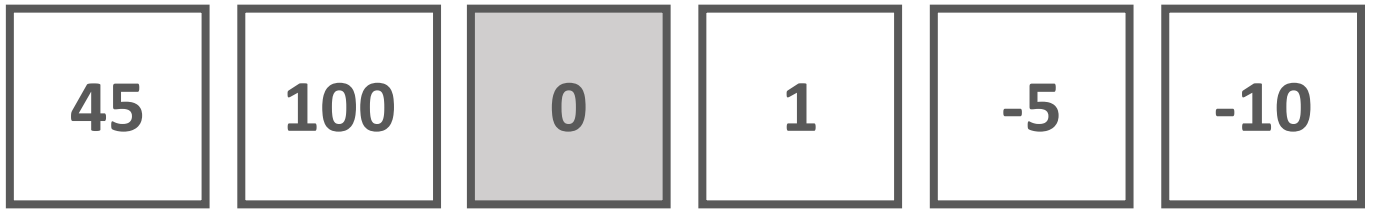
idx = 0



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

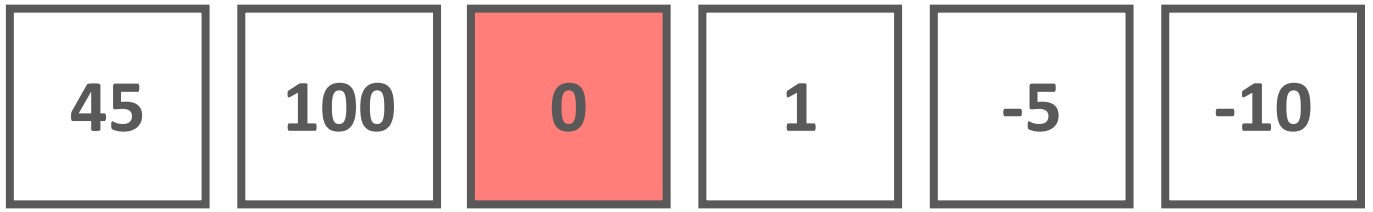
idx = 0



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

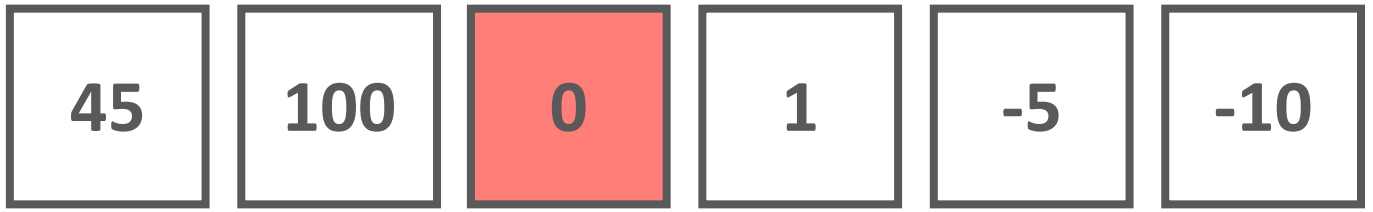
idx = 0



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

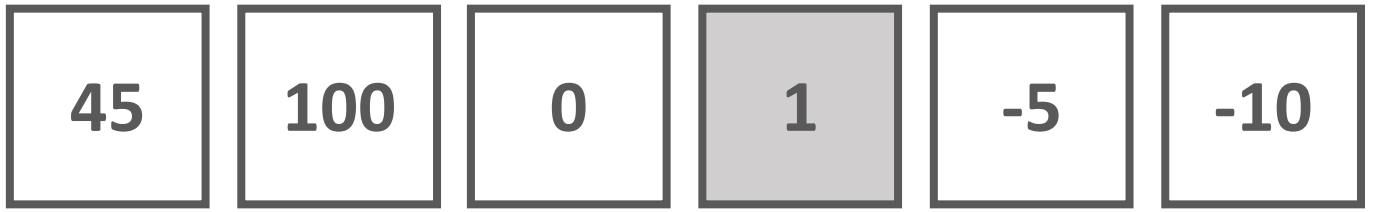
idx = 2



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

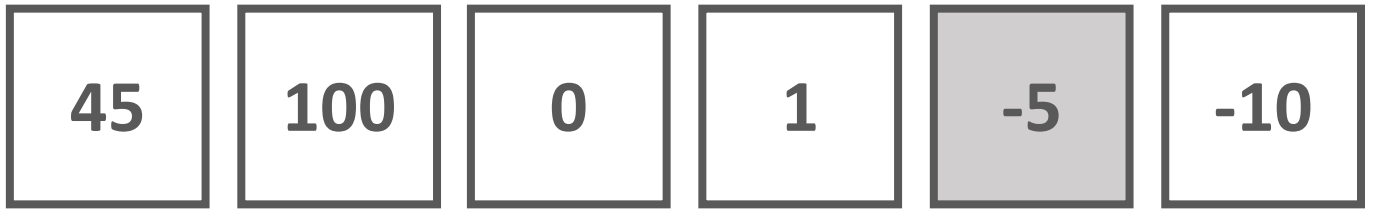
idx = 2



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

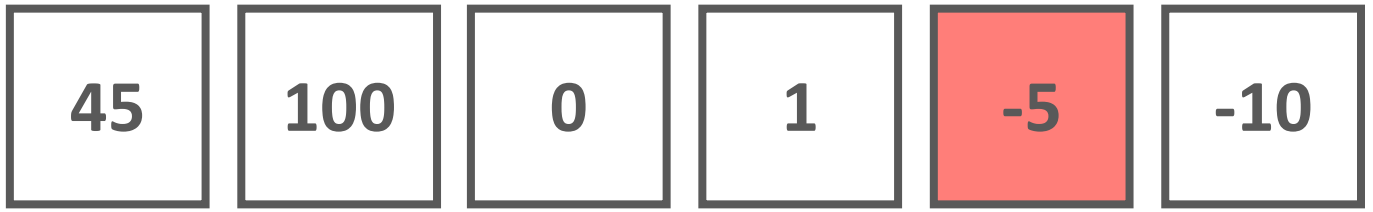
idx = 2



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

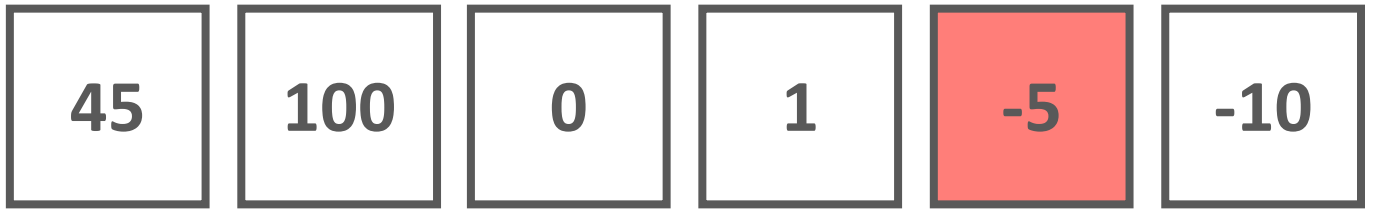
idx = 2



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

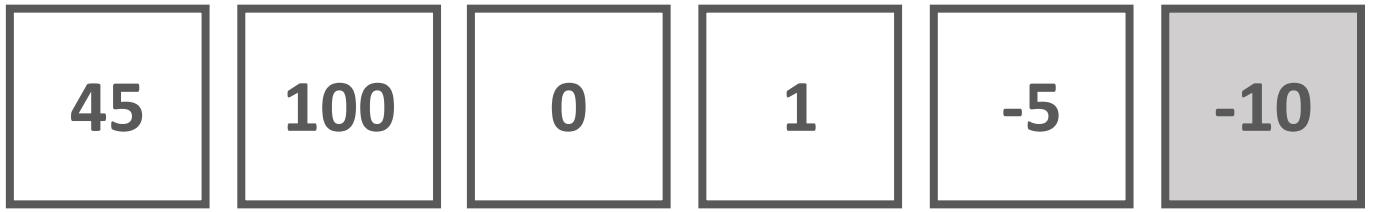
idx = 4



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

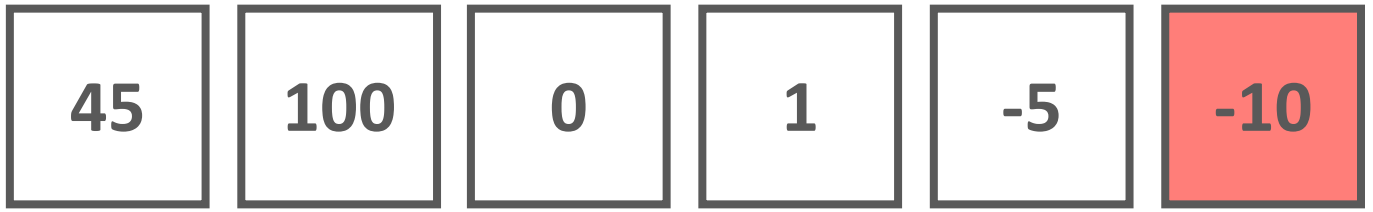
idx = 4



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

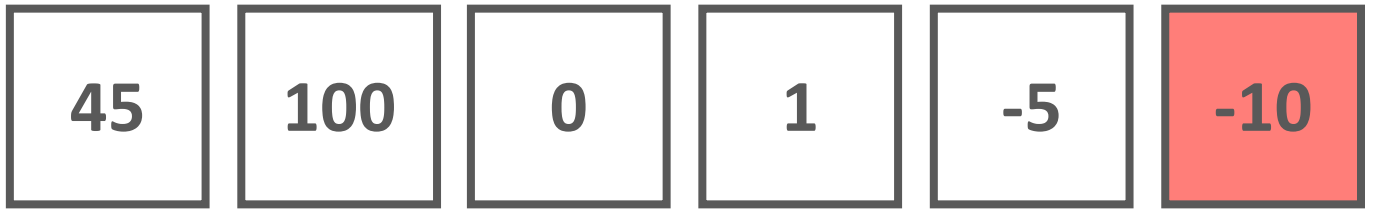
idx = 4



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

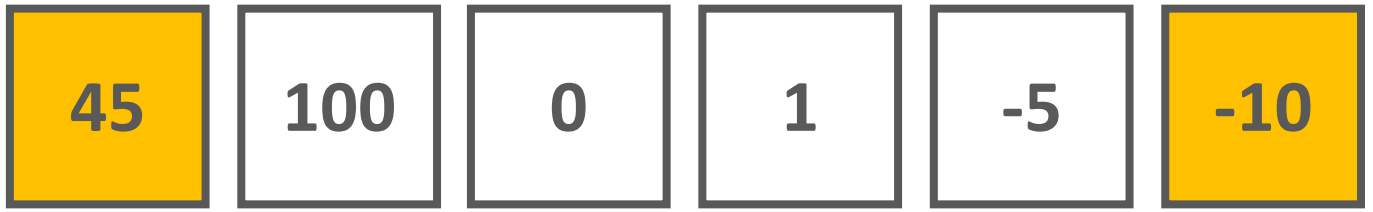
idx = 5



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

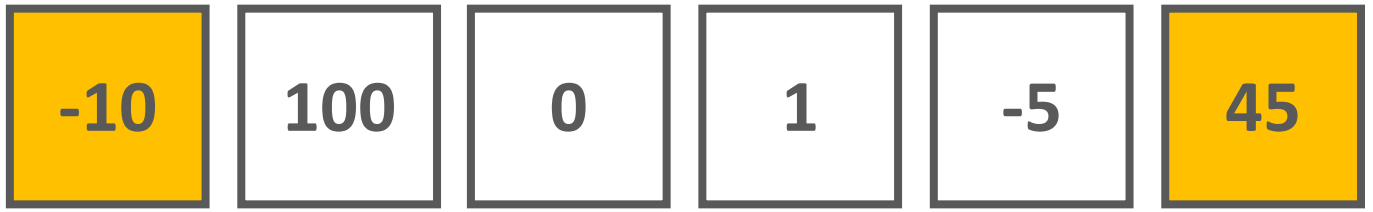
idx = 5



Sorting Algorithms | Selection Sort

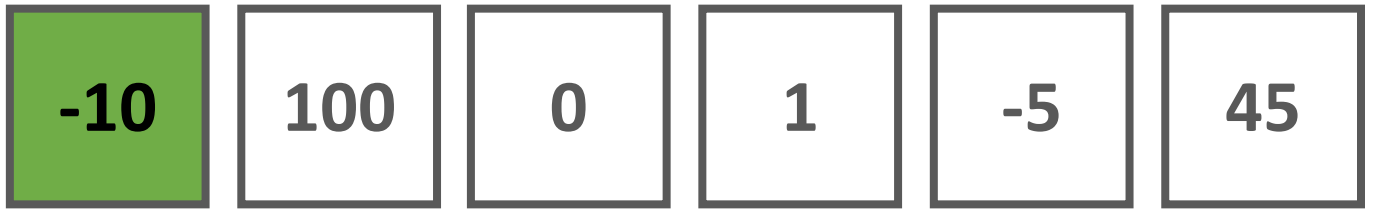
Finding the smallest element and move it to the beginning of data

idx = 5



Sorting Algorithms | Selection Sort

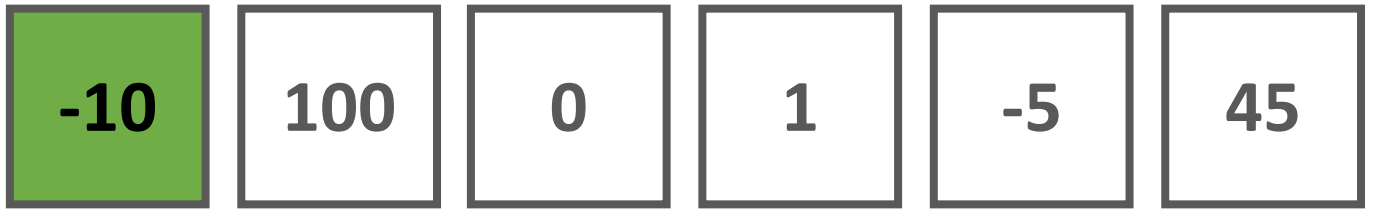
Finding the smallest element and move it to the beginning of data



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

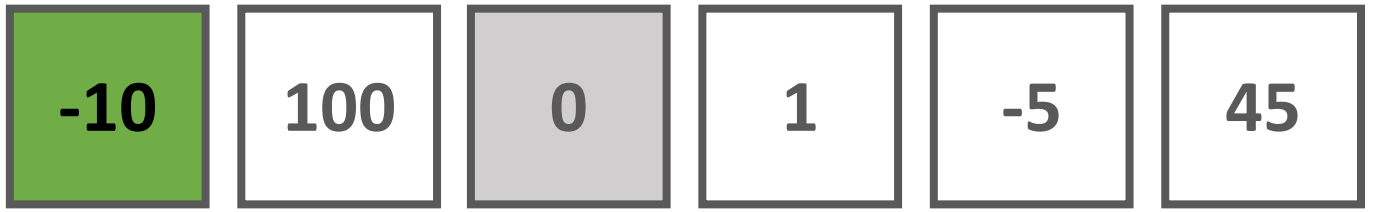
idx = 1



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

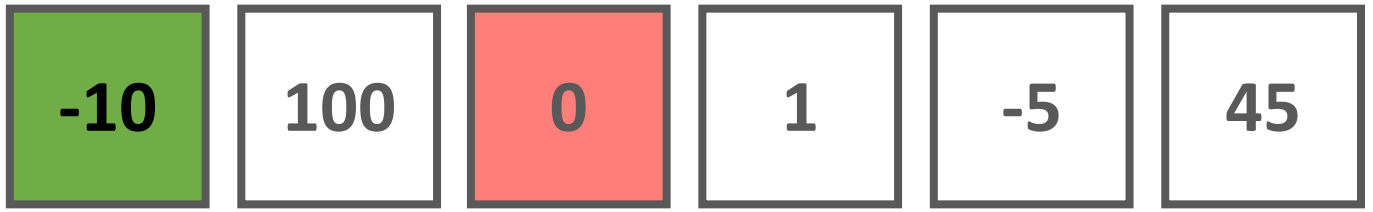
idx = 1



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

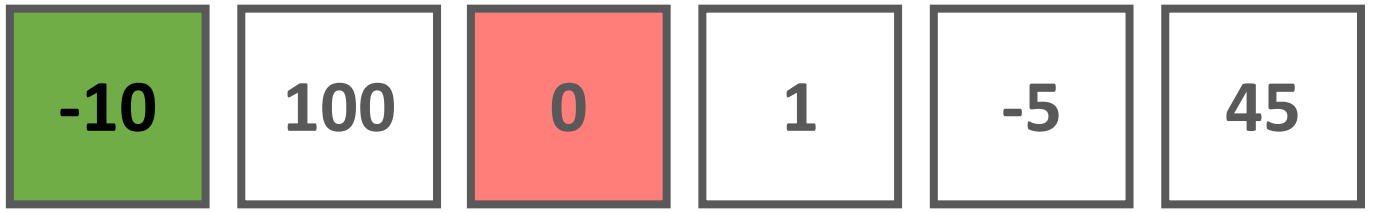
idx = 1



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

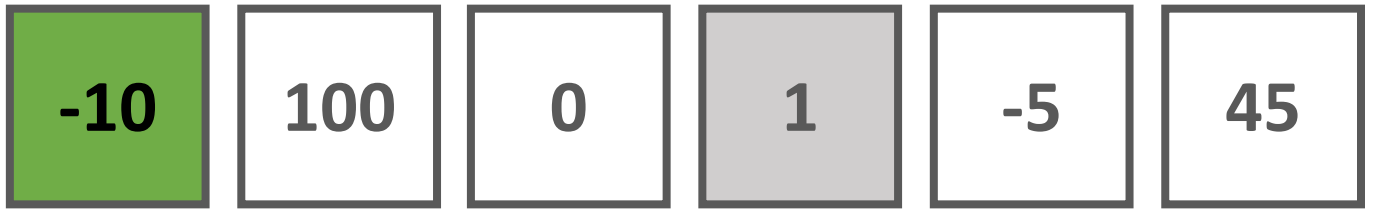
idx = 2



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

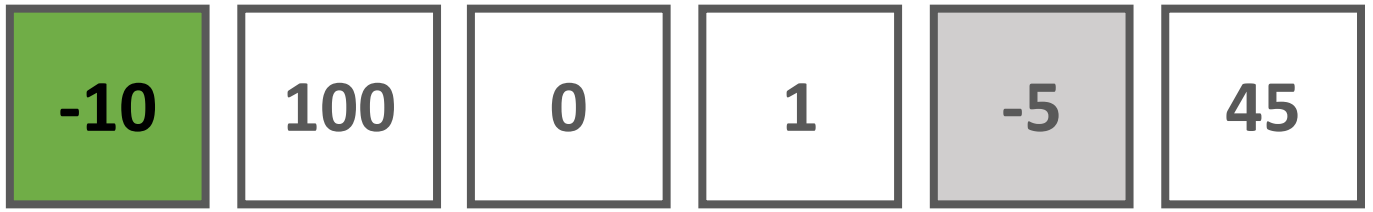
idx = 2



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

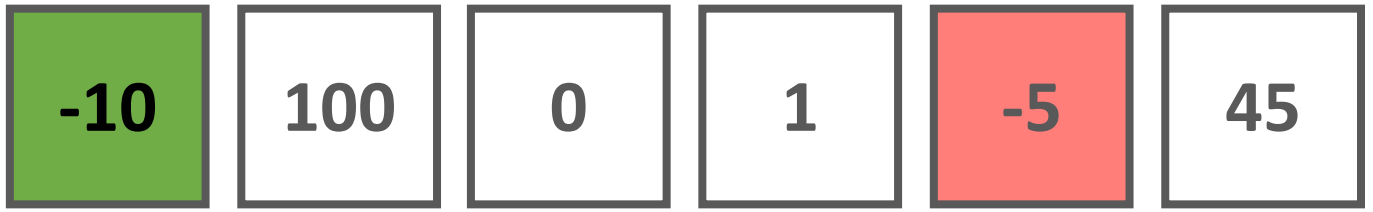
idx = 2



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

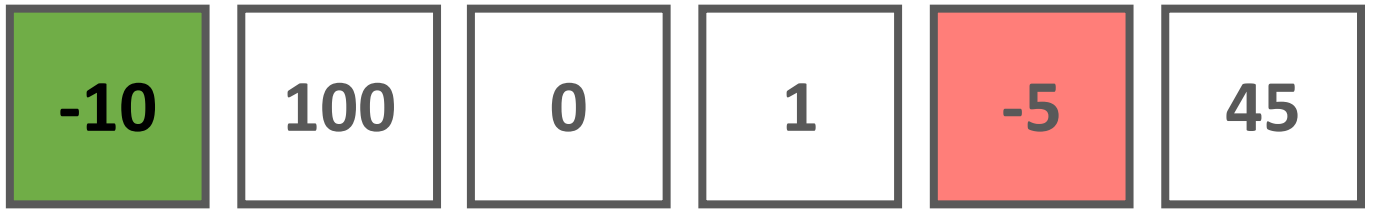
idx = 2



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

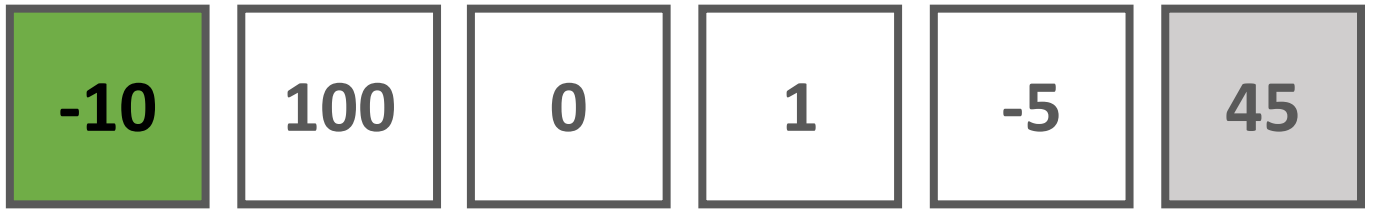
idx = 4



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

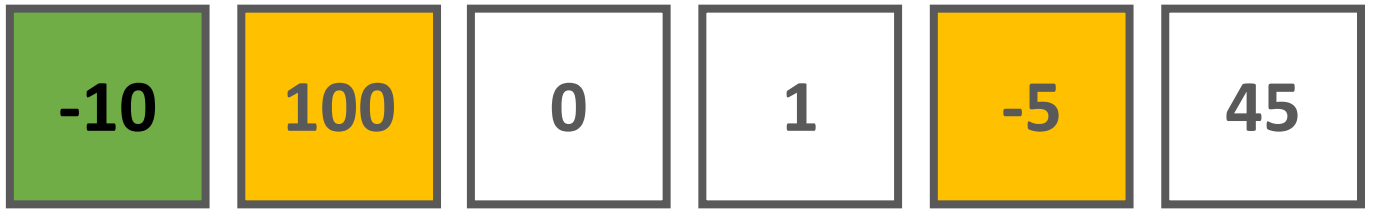
idx = 4



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

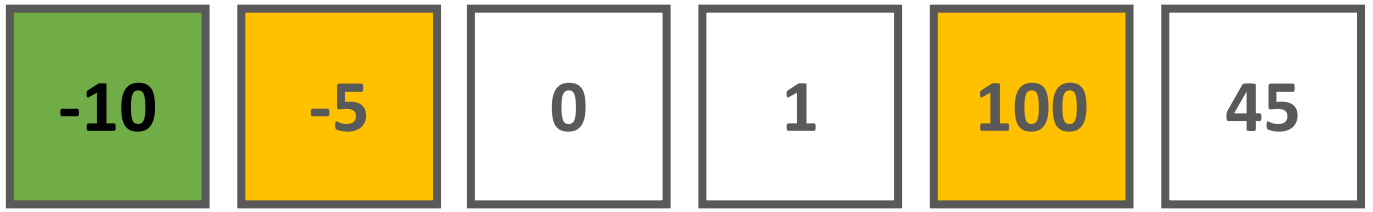
idx = 4



Sorting Algorithms | Selection Sort

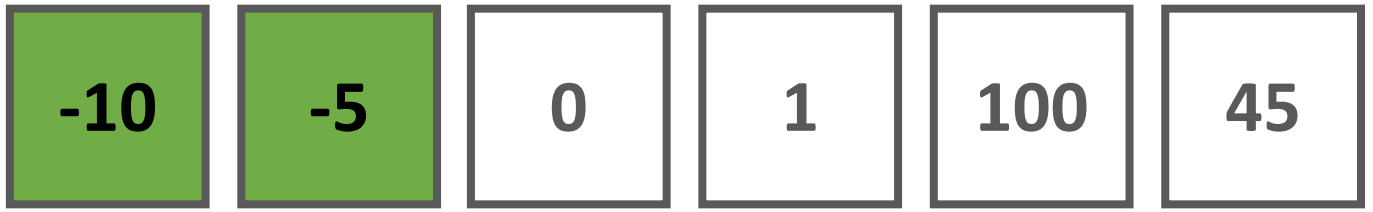
Finding the smallest element and move it to the beginning of data

idx = 4



Sorting Algorithms | Selection Sort

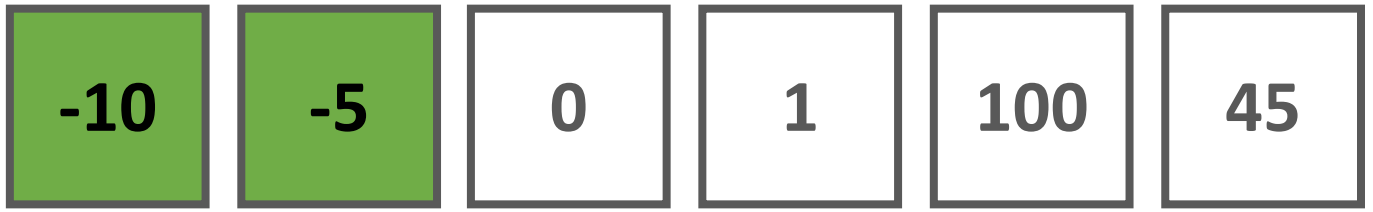
Finding the smallest element and move it to the beginning of data



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

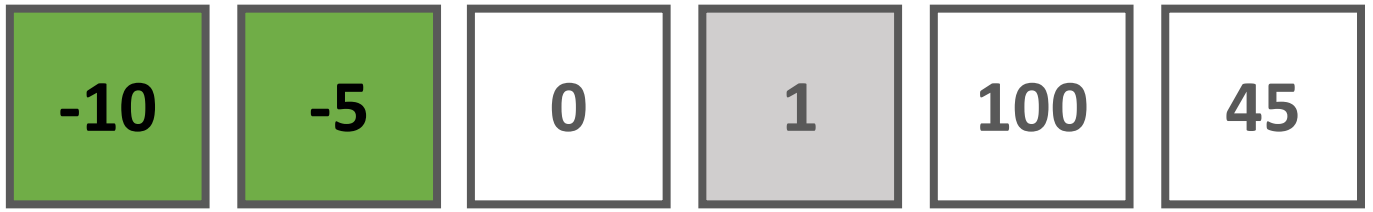
idx = 2



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

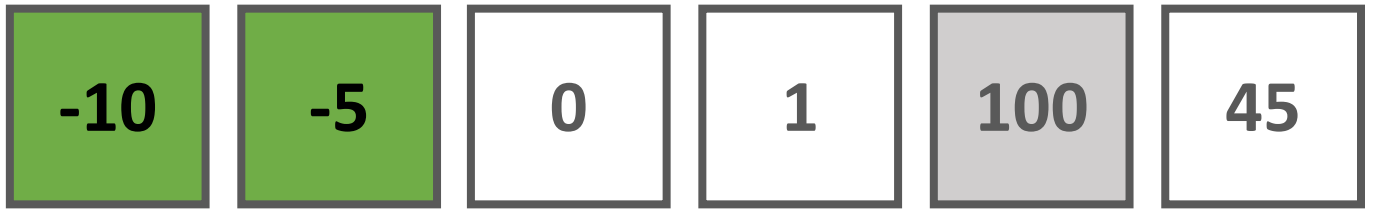
idx = 2



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

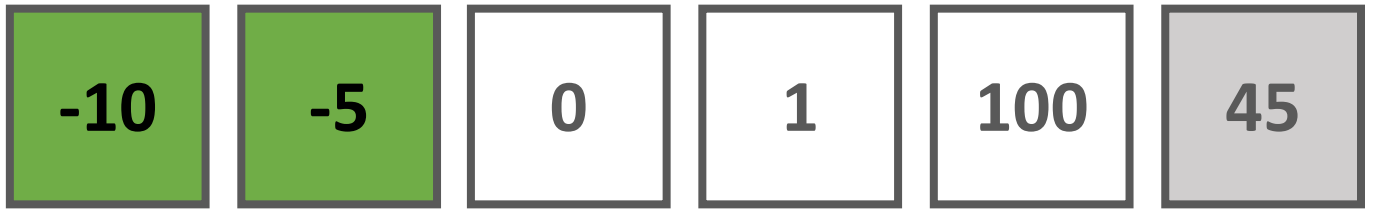
idx = 2



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

idx = 2



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

idx = 2

Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

idx = 3

Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

idx = 3

Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

idx = 3

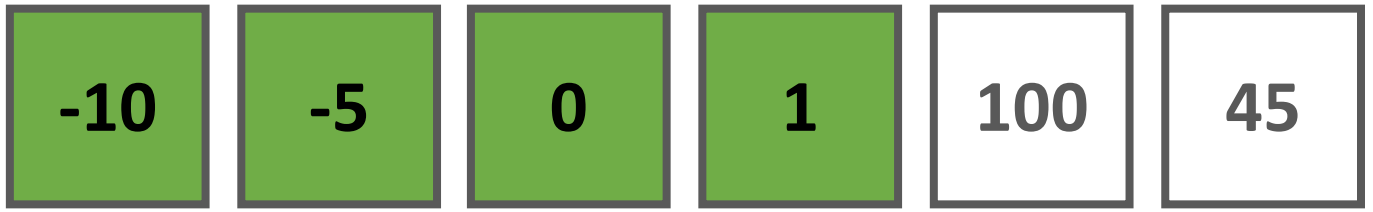
Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

idx = 3

Sorting Algorithms | Selection Sort

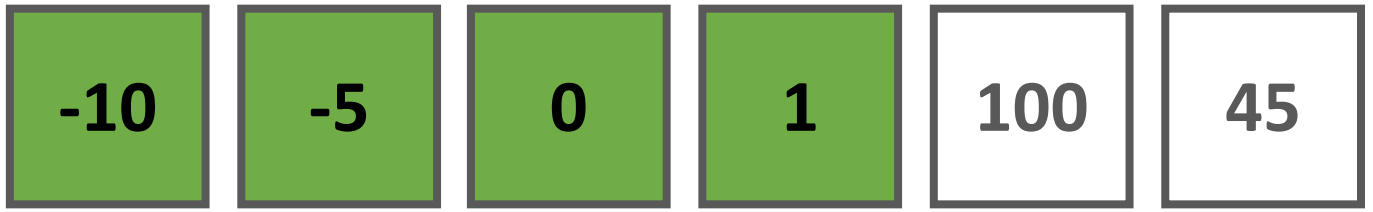
Finding the smallest element and move it to the beginning of data



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

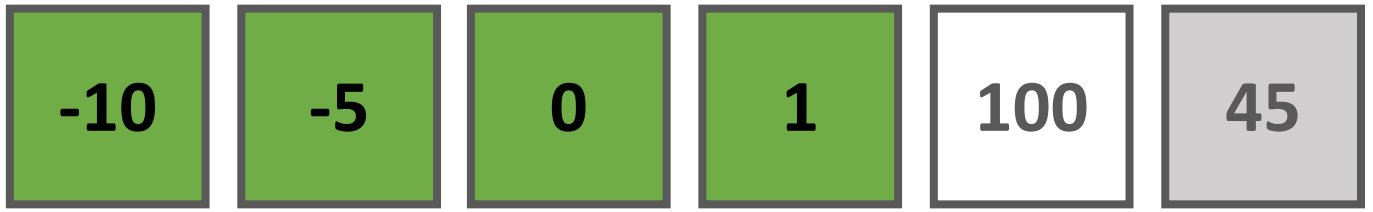
idx = 4



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

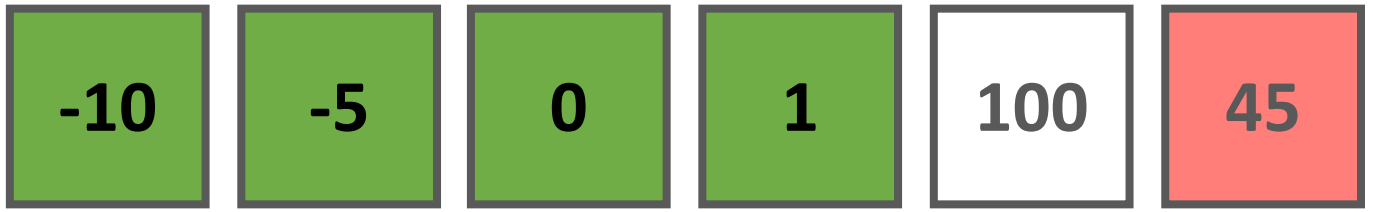
idx = 4



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

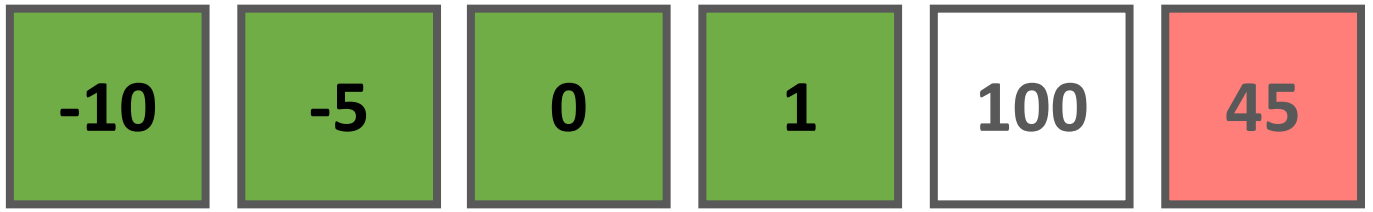
idx = 4



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

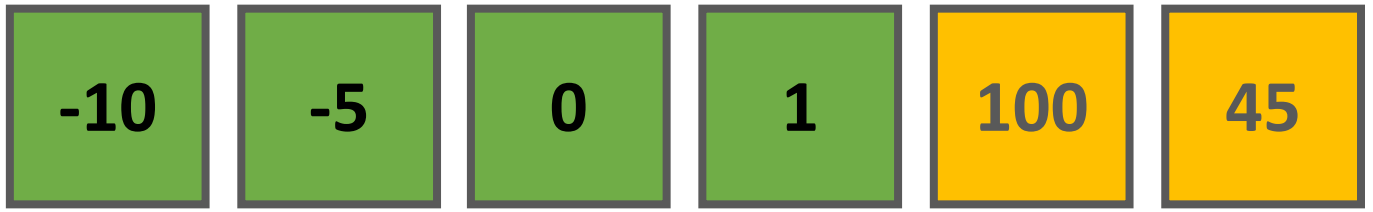
idx = 5



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data

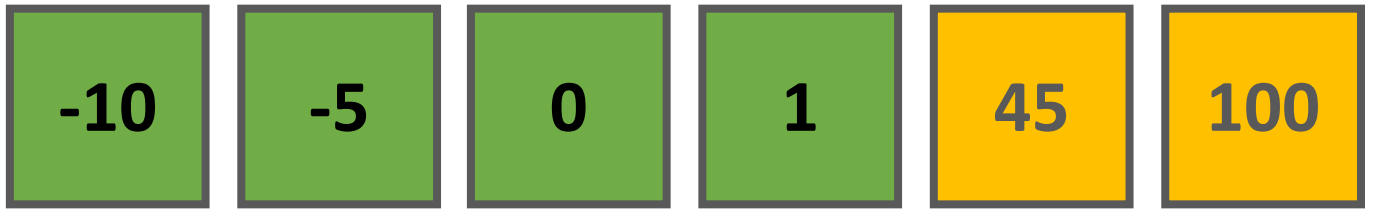
idx = 5



Sorting Algorithms | Selection Sort

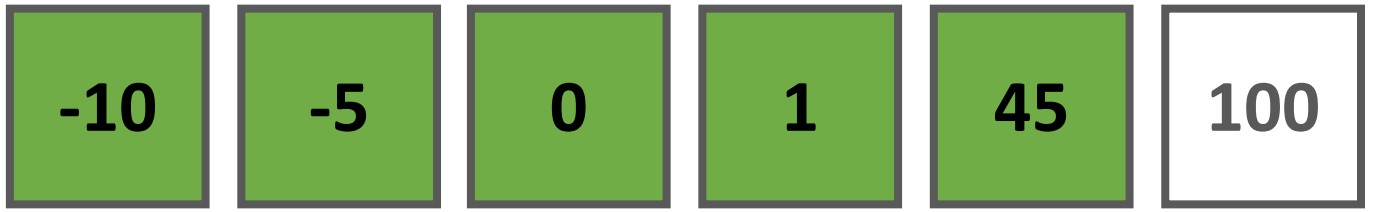
Finding the smallest element and move it to the beginning of data

idx = 5



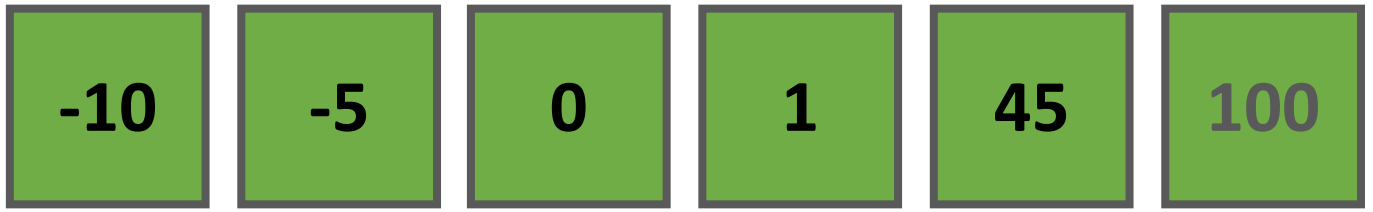
Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data



Sorting Algorithms | Selection Sort

Finding the smallest element and move it to the beginning of data



Sorting Algorithms | Selection Sort

Pseudocode

1. Find the smallest element and record its index.
2. Swap the recorded smallest element with the left most (unsorted) item in the array.
3. Repeat 1,2 until all the elements are placed at the right position.

Sorting Algorithms | Selection Sort

Pseudocode

1. Find the smallest element and record its index.
2. Swap the recorded smallest element with the left most (unsorted) item in the array.
3. Repeat 1,2 until all the elements are placed at the right position.

```
1  def SS_min(L):
2      for i in range(len(L) - 1):
3          min = i
4          for j in range(i + 1, len(L)):
5              if L[j] < L[min]:
6                  min = j
7          #swap
8          L[i], L[min] = L[min], L[i]
```

Activity

Implement selection sort algorithm by sorting the “largest” element in the data container.
Hint: The largest element has to be replaced with the right most unsorted element.

Activity Solution

Implement selection sort algorithm by sorting the “largest” element in the data container.

```
def SS_max(L):
    for i in range(len(L) - 1):
        max = 0
        for j in range(1, len(L)-i):
            if L[j] > L[max]:
                max = j
        #swap
        L[-1 - i], L[max] = L[max], L[-1 - i]
```