UCONN
UNIVERSITY OF CONNECTICUT

Department of Computer Science and Engineering

Data Structures and Object-Oriented Design

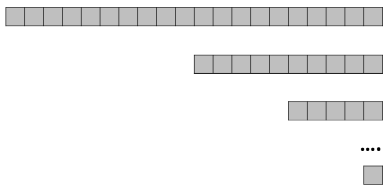
(CSE – 2050)

Hasan Baig

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

Quick Recap

- Binary Search Algorithm
  - Works on sorted data
  - Implementation using slicing → O(n)
  - Implementation using low/high indices → O(logn)



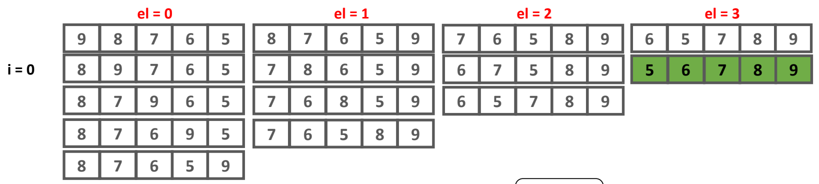
  - Determining if the data is sorted
  - Comparing each element with all elements ahead of it → O(n^2)
    - Comparing neighboring elements → O(n)

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

Quick Recap

- Quadratic Sorting Algorithms
  - Bubble sort algorithm

```
def bad_sort(L):
   for el in range(len(L) - 1):
     for i in range(len(L) - 1 - el):
       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
```



$O(n^2)$

Quick Recap

- Quadratic Sorting Algorithms
  - Selection sort algorithm
  - Select the smallest/largest element and move it to left/right respectively

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.

```
def SS_min(L):
    for i in range(len(L) - 1):
        min = i
        for j in range(i + 1, len(L)):
            if L[j] < L[min]:
                min = j
        #swap
        L[i], L[min] = L[min], L[i]
```

```
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]
```

Sorting Algorithms | Insertion Sort

- Another O(n^2) quadratic runtime algorithm
- Easy to implement
- More efficient than bubble sort and selection sort algorithms
  - Selection sort is better for applications where less number of write operations are required
- Online algorithm – sort array as it receives data (example from web)

Sorting Algorithms | Insertion Sort

- Compare the current element with all its preceding elements
- If the current element is smaller than its preceding element → Swap

12	4	-2	11	3	2

Sorting Algorithms | Insertion Sort

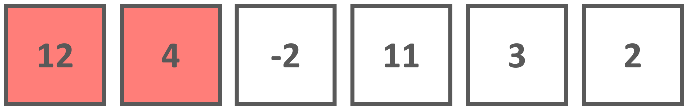
- Start off by comparing the element at index 1

12	4	-2	11	3	2

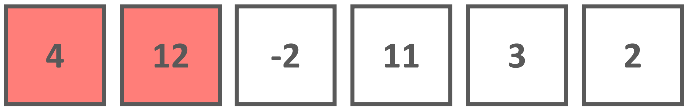
Sorting Algorithms | Insertion Sort

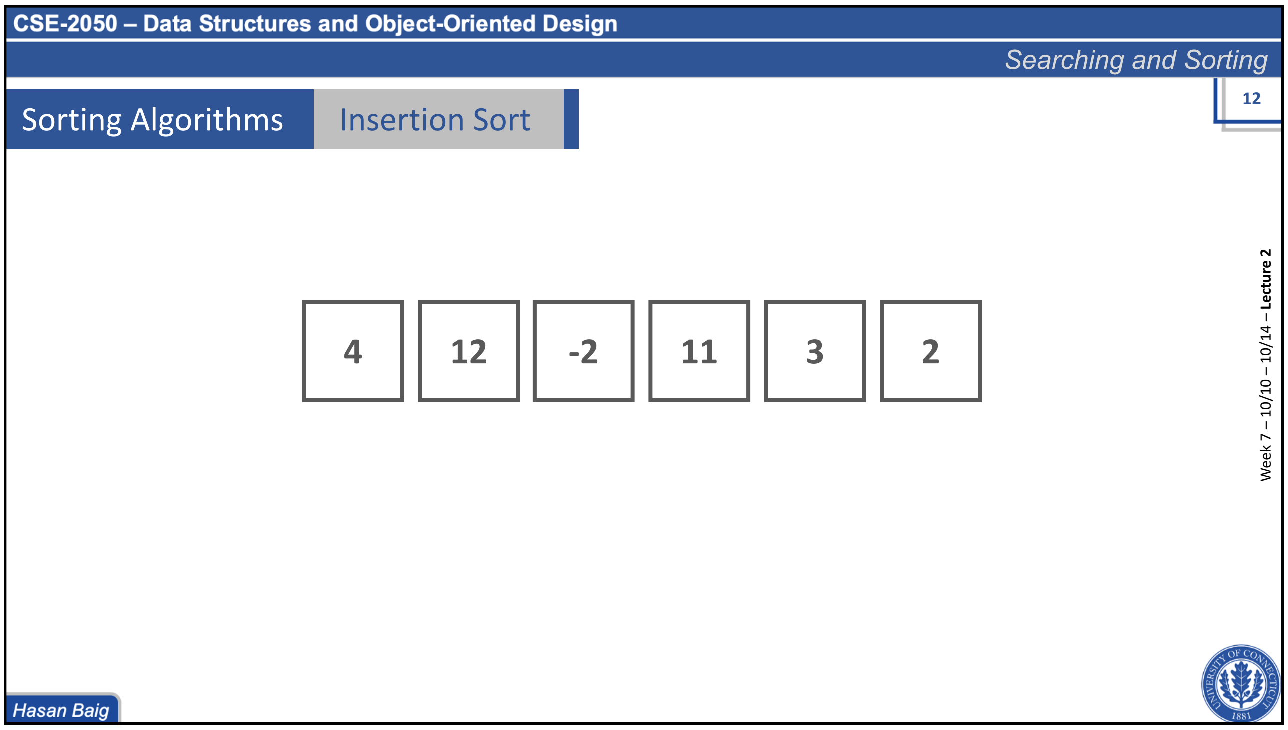
12	4	-2	11	3	2

Sorting Algorithms | Insertion Sort



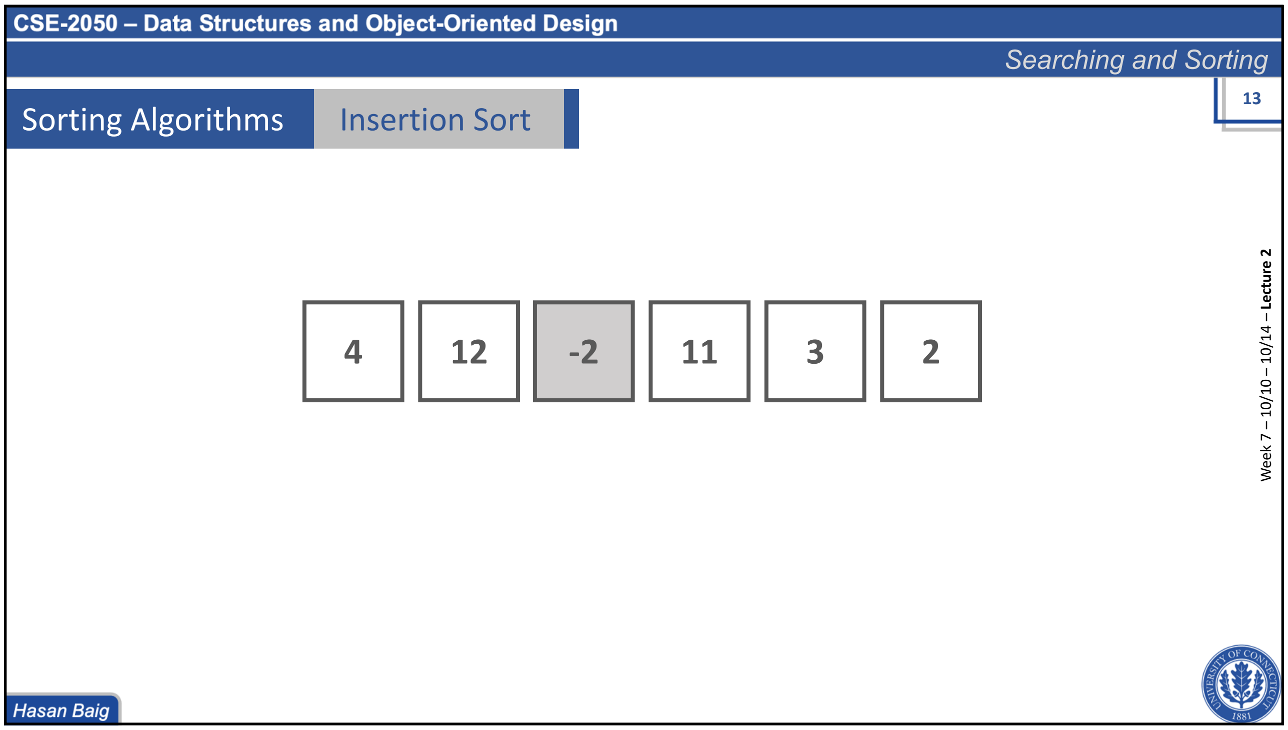
Sorting Algorithms | Insertion Sort





Sorting Algorithms | Insertion Sort

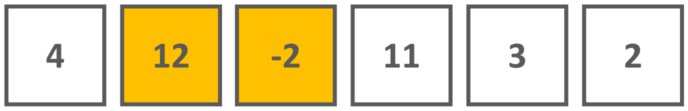
4	12	-2	11	3	2



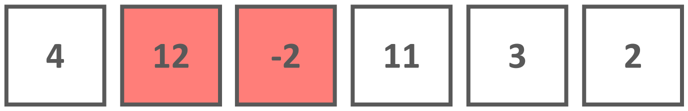
Sorting Algorithms | Insertion Sort

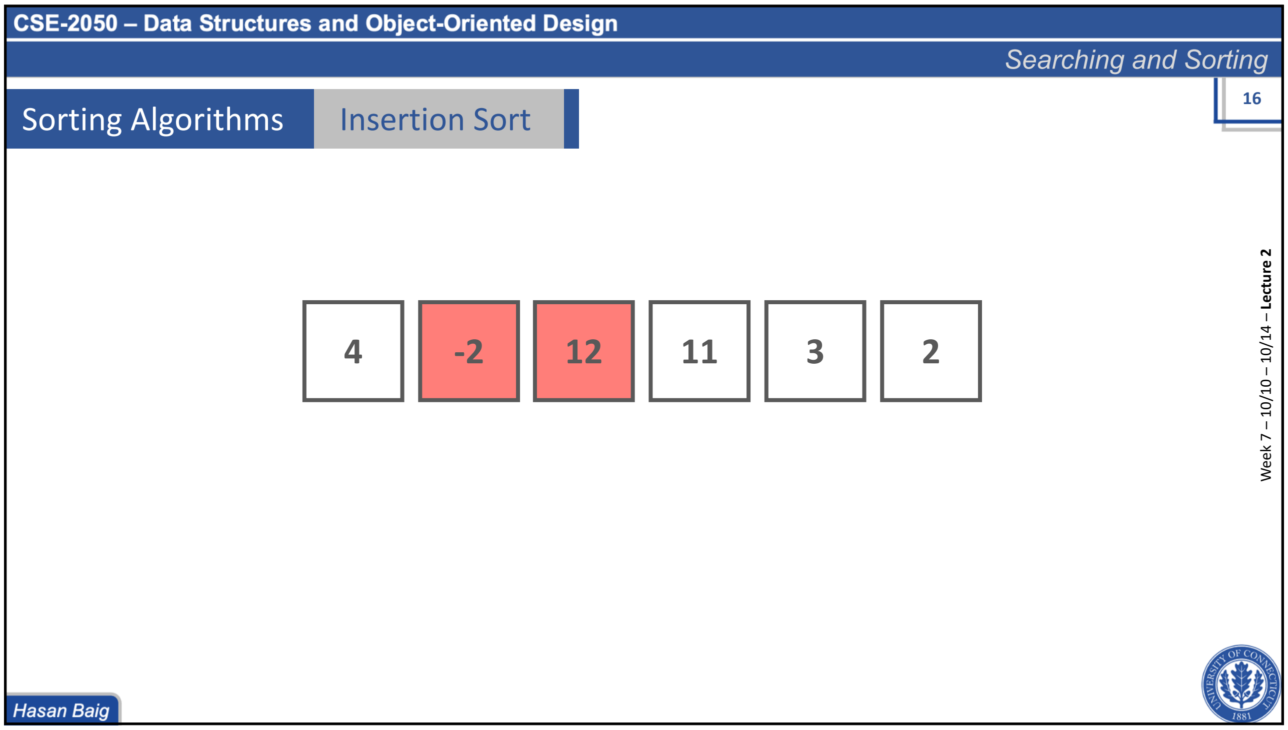
4	12	-2	11	3	2

Sorting Algorithms | Insertion Sort



Sorting Algorithms | Insertion Sort



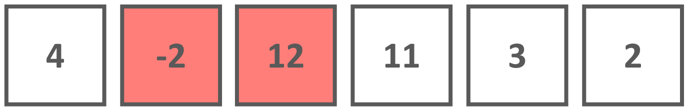


CSE-2050 – Data Structures and Object-Oriented Design

Searching and Sorting

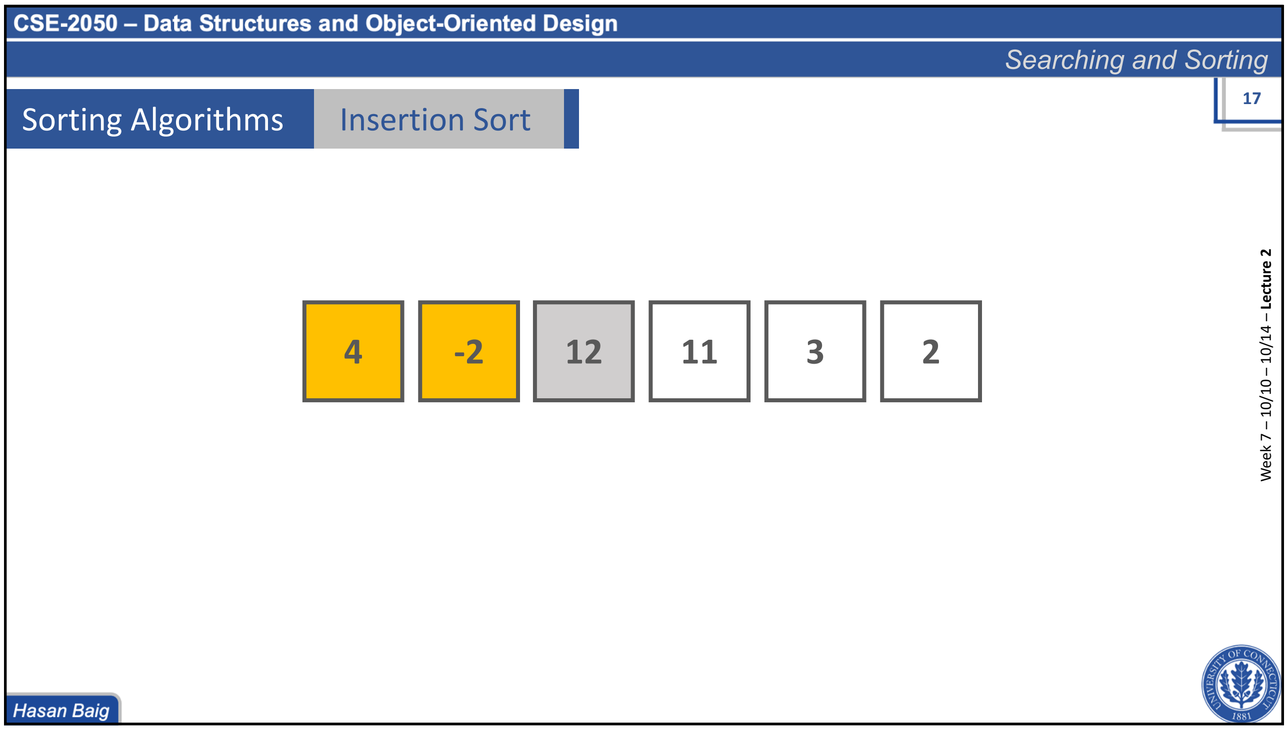
16

Sorting Algorithms — Insertion Sort



Week 7 – 10/10 – 10/14 – **Lecture 2**

Hasan Baig

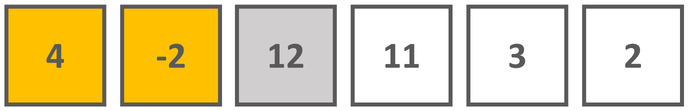


CSE-2050 – Data Structures and Object-Oriented Design

Searching and Sorting

17

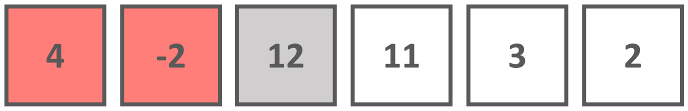
Sorting Algorithms — Insertion Sort



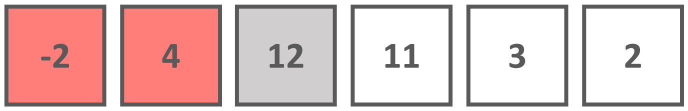
Week 7 – 10/10 – 10/14 – **Lecture 2**

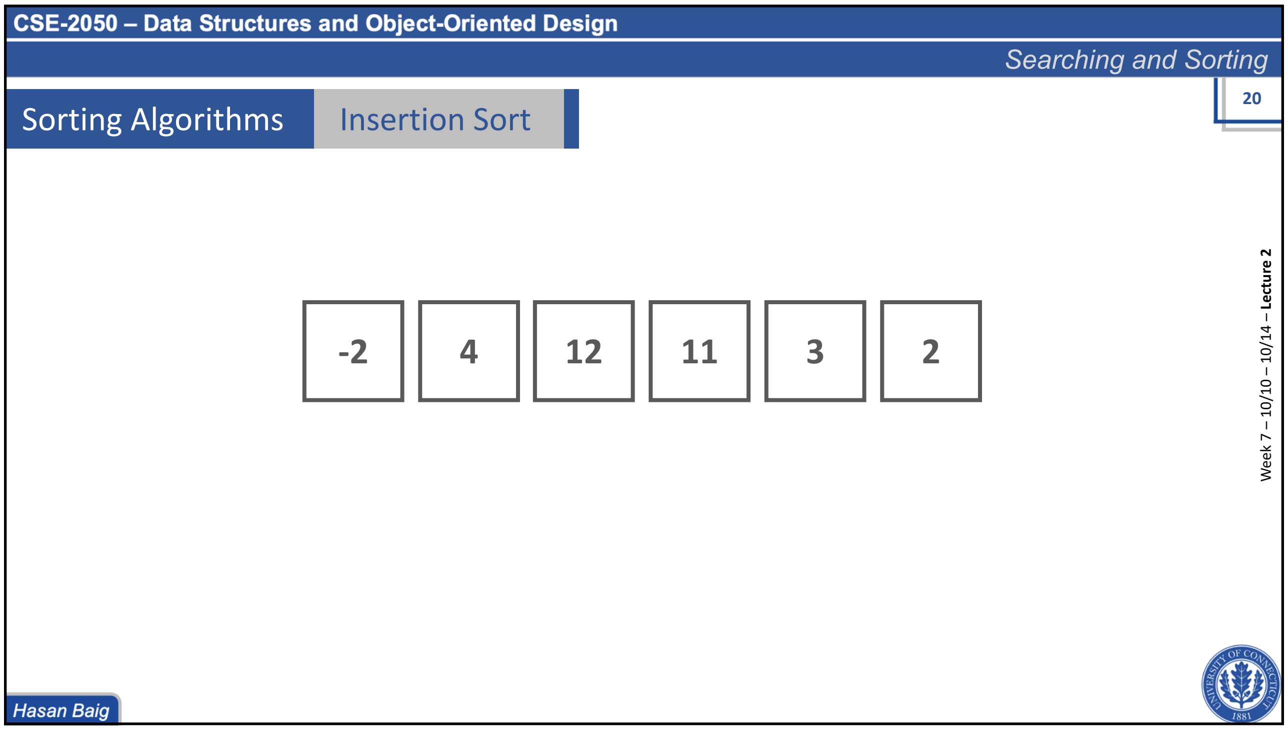
Hasan Baig

Sorting Algorithms | Insertion Sort



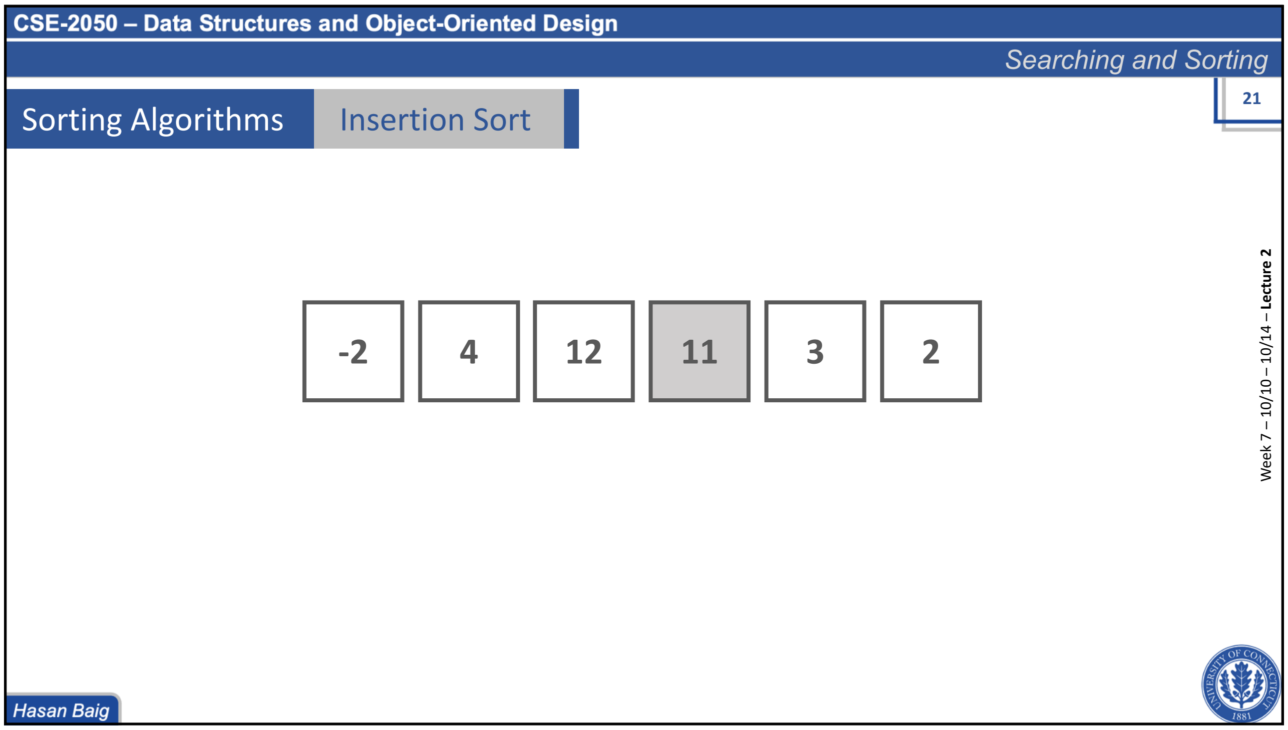
Sorting Algorithms | Insertion Sort





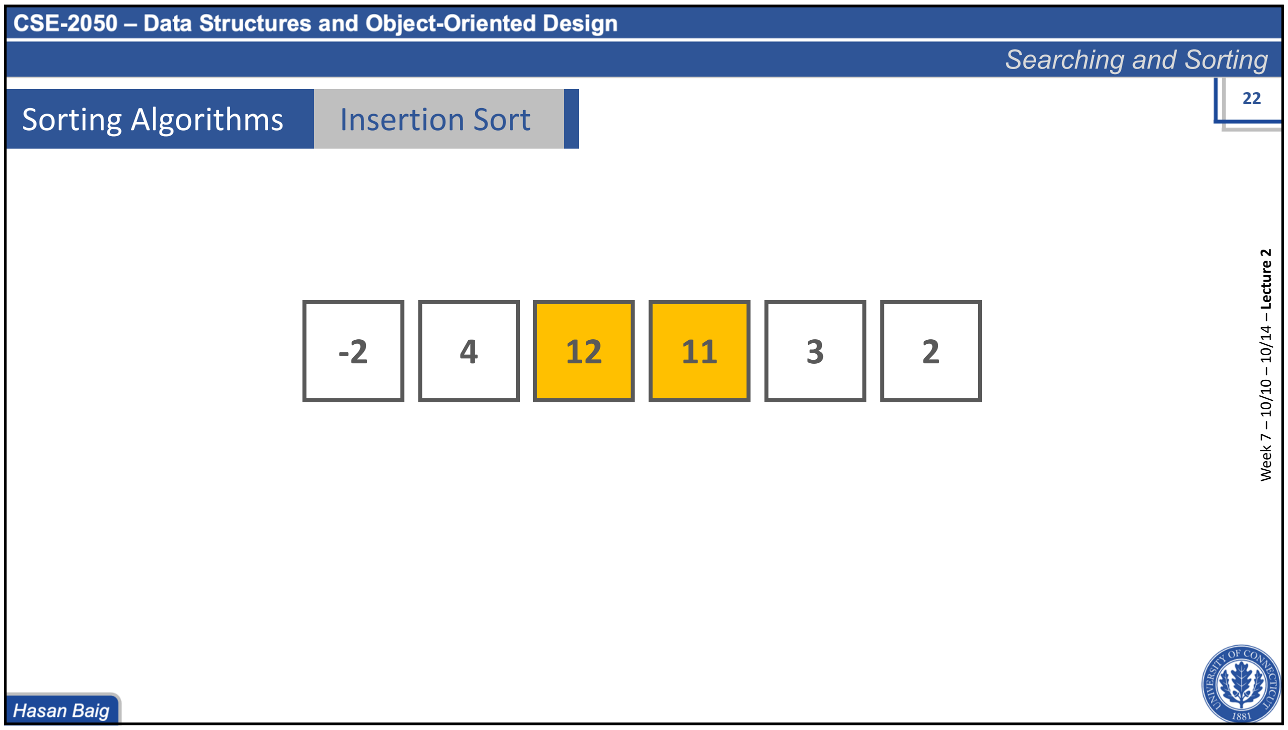
Sorting Algorithms | Insertion Sort

-2	4	12	11	3	2



Sorting Algorithms | Insertion Sort

-2	4	12	▨ 11	3	2



CSE-2050 – Data Structures and Object-Oriented Design

Searching and Sorting

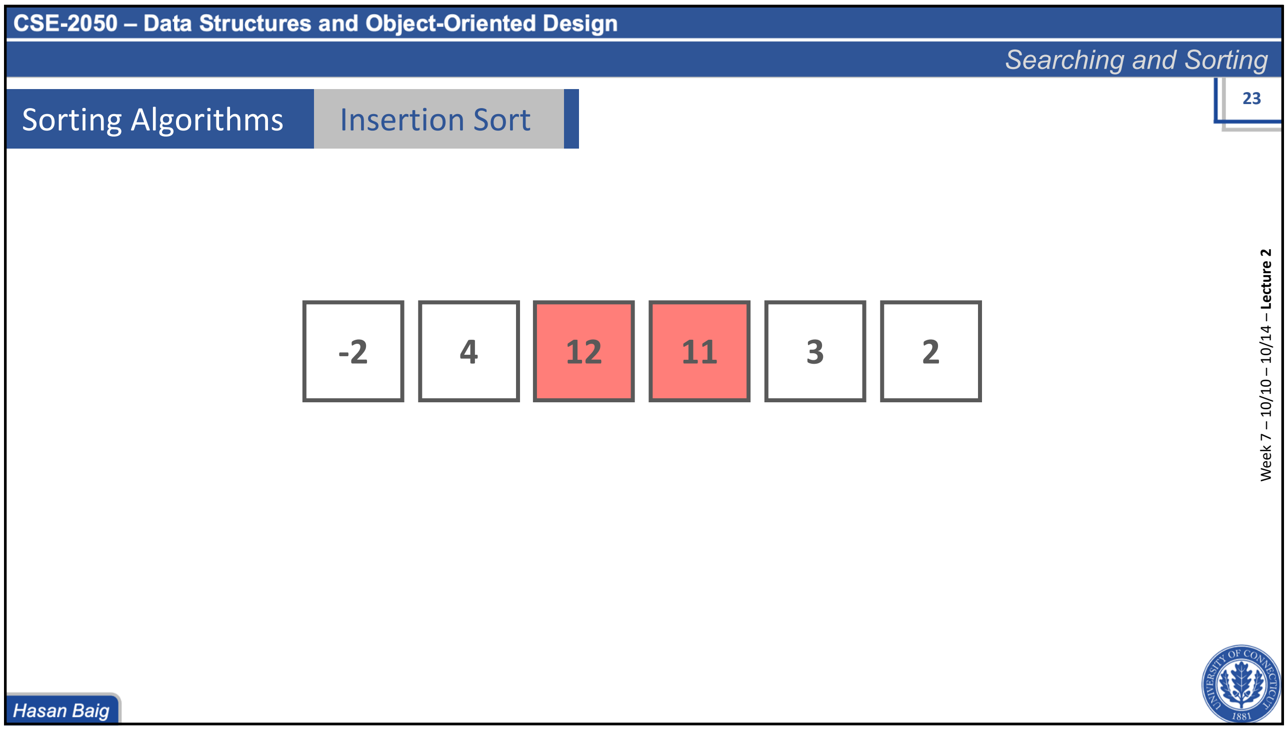
22

Sorting Algorithms | Insertion Sort

-2	4	12	11	3	2

Week 7 – 10/10 – 10/14 – **Lecture 2**

Hasan Baig



CSE-2050 – Data Structures and Object-Oriented Design

Searching and Sorting

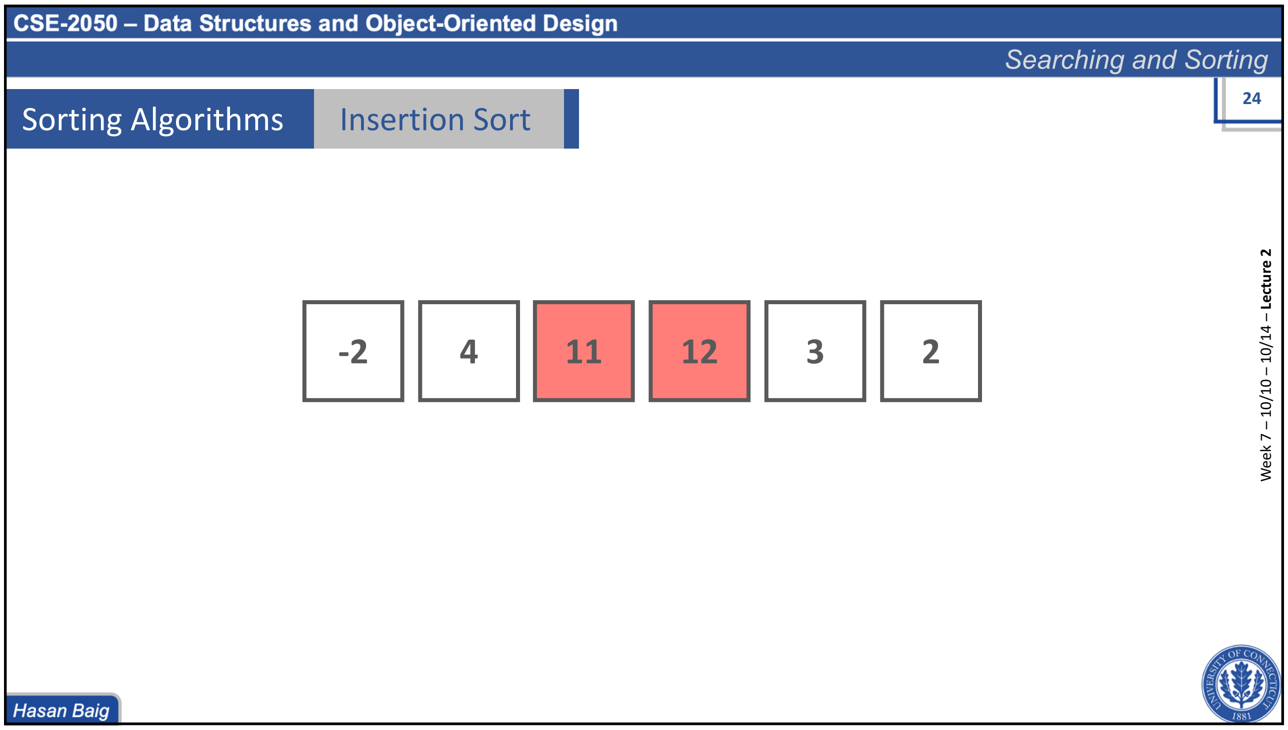
23

Sorting Algorithms | Insertion Sort

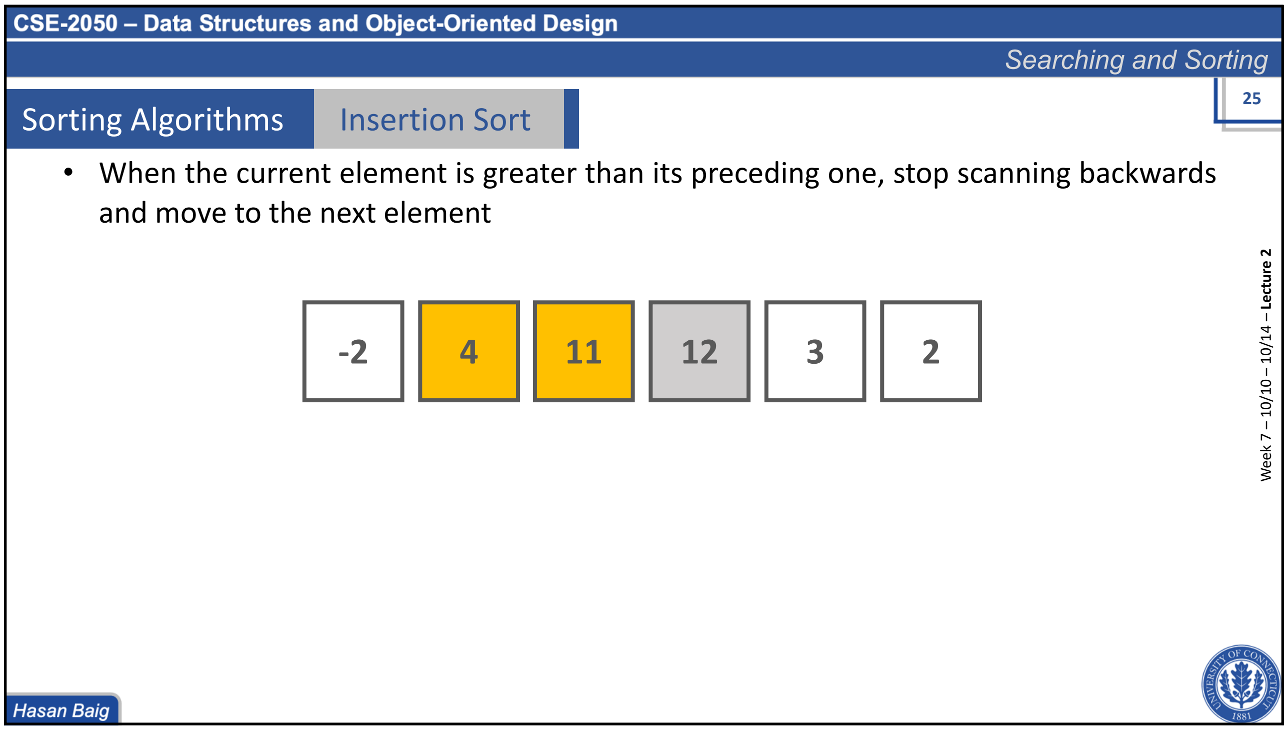
-2	4	12	11	3	2

Week 7 – 10/10 – 10/14 – **Lecture 2**

Hasan Baig

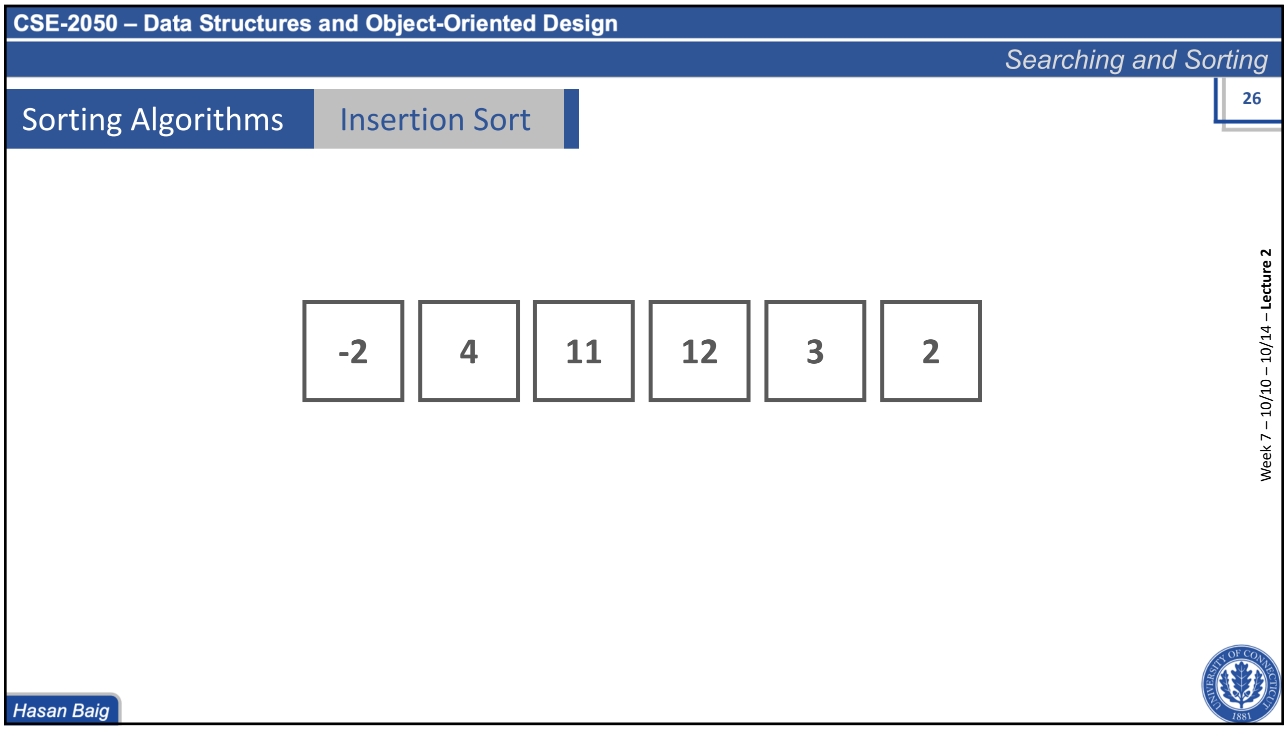


Sorting Algorithms | Insertion Sort



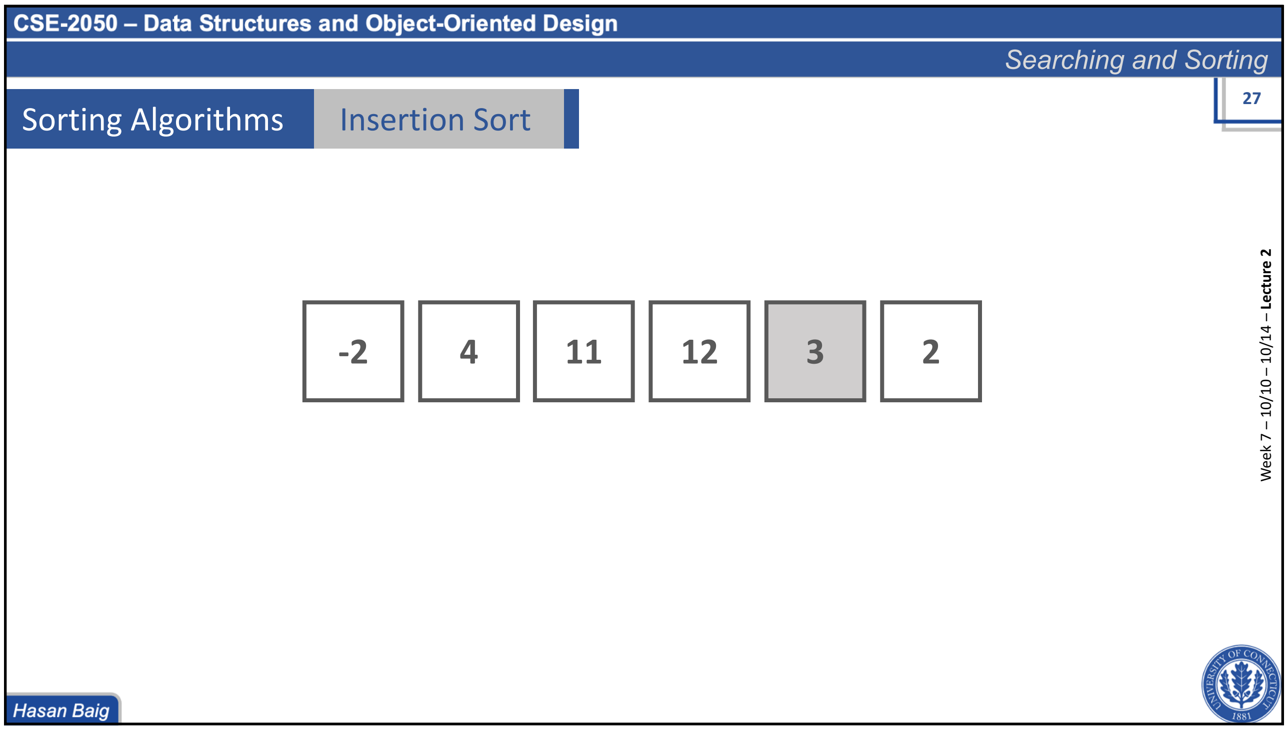
Sorting Algorithms | Insertion Sort

- When the current element is greater than its preceding one, stop scanning backwards and move to the next element



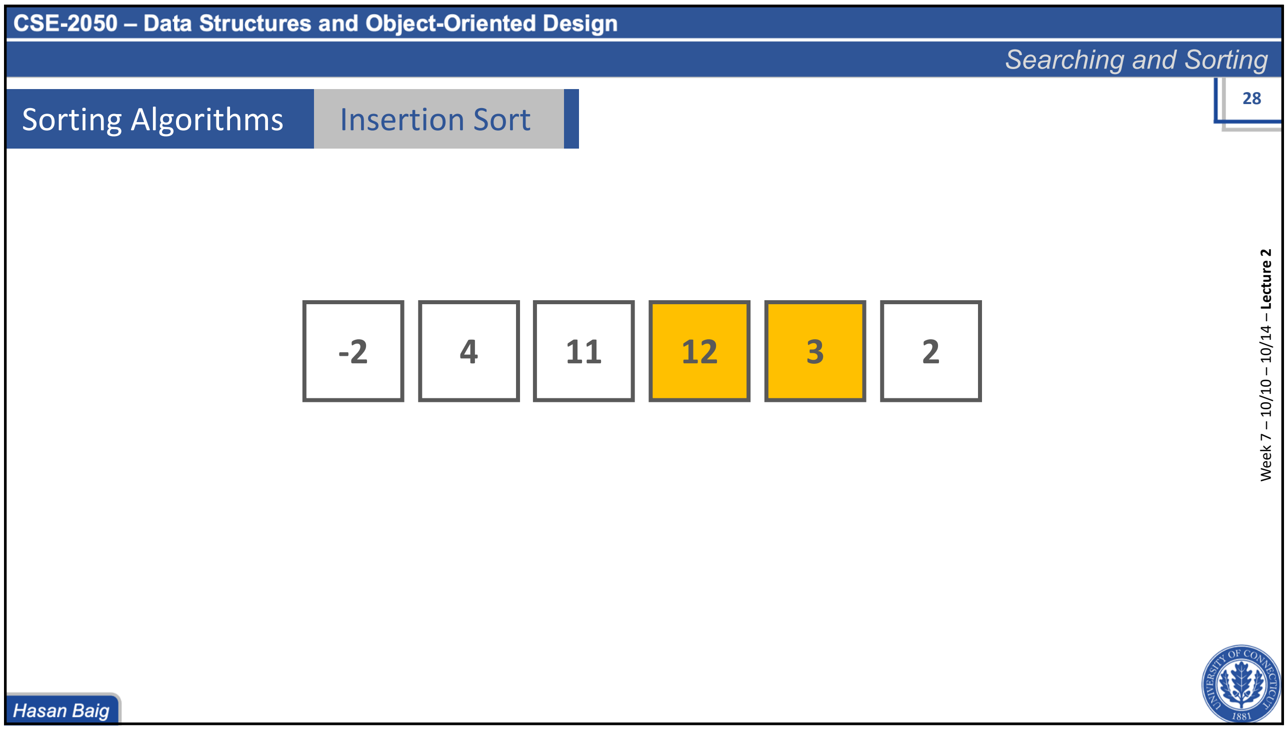
Sorting Algorithms | Insertion Sort

-2	4	11	12	3	2



Sorting Algorithms | Insertion Sort

-2	4	11	12	▨ 3	2



CSE-2050 – Data Structures and Object-Oriented Design

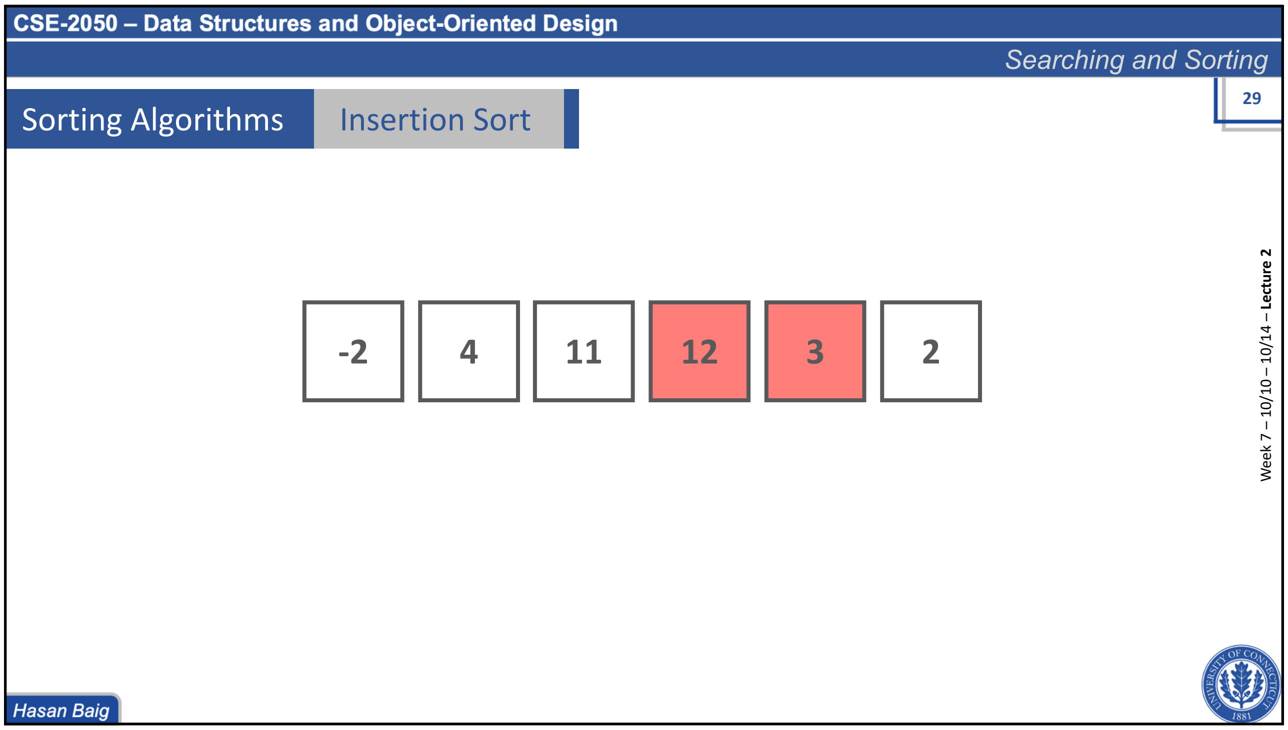
Searching and Sorting

Sorting Algorithms | Insertion Sort

-2	4	11	12	3	2

Week 7 – 10/10 – 10/14 – **Lecture 2**

Hasan Baig



CSE-2050 – Data Structures and Object-Oriented Design

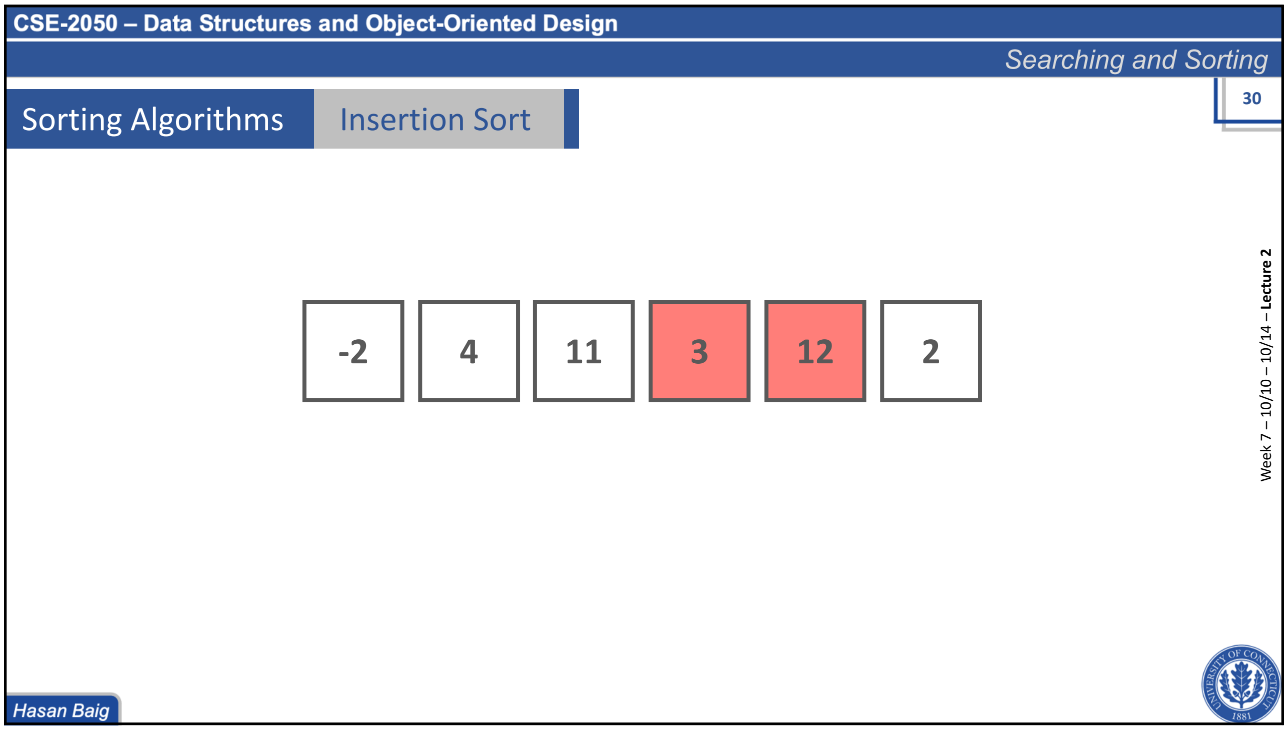
Searching and Sorting

Sorting Algorithms | Insertion Sort

-2	4	11	12	3	2

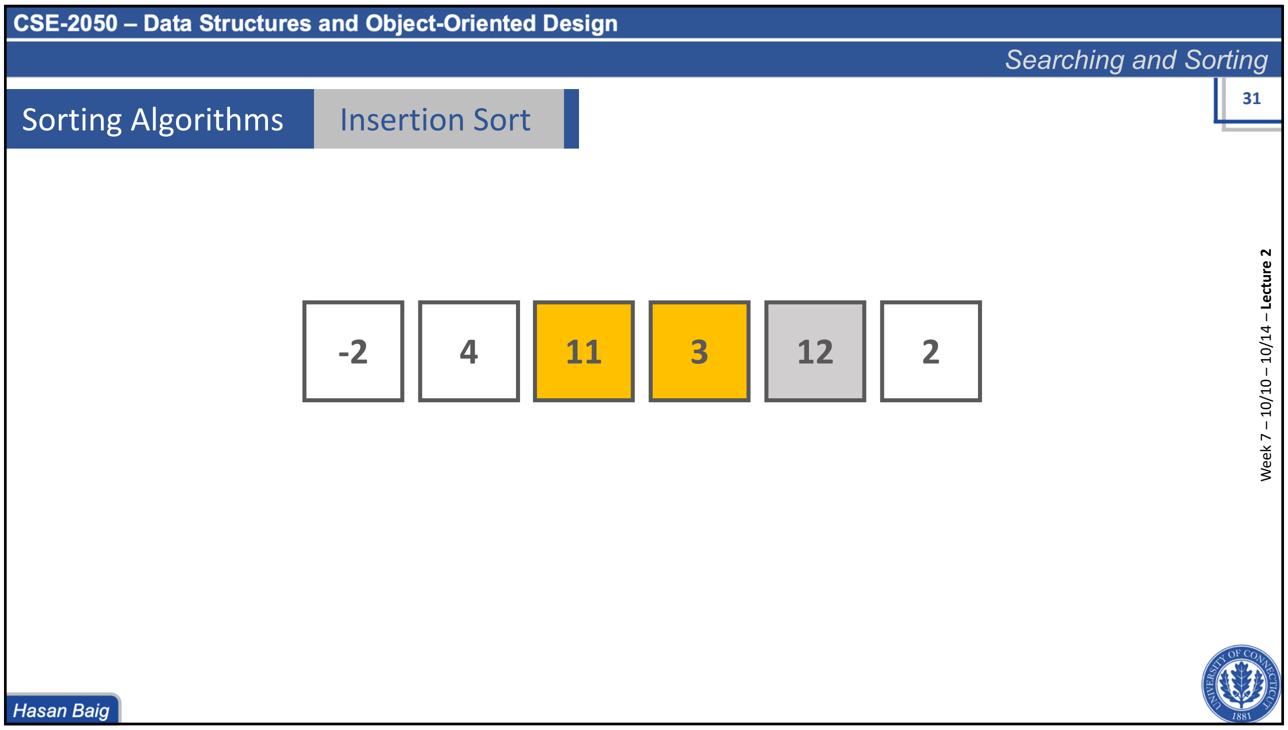
Week 7 – 10/10 – 10/14 – **Lecture 2**

Hasan Baig



Sorting Algorithms | Insertion Sort

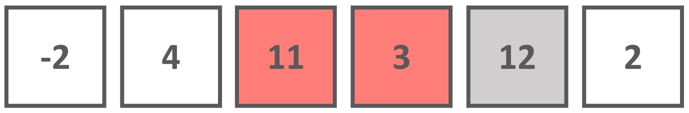
-2	4	11	🟥 3	🟥 12	2



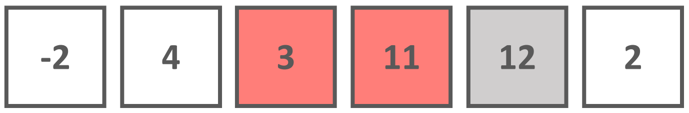
Sorting Algorithms | Insertion Sort

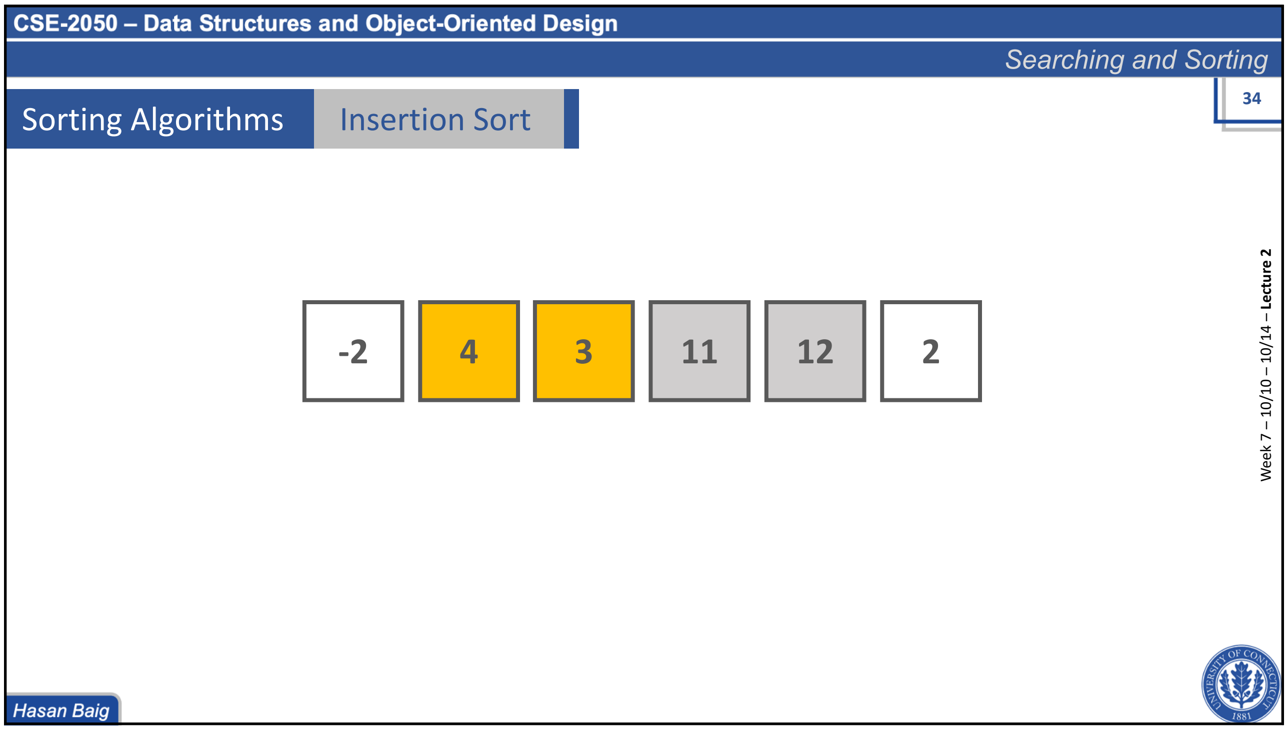
-2	4	🟨 11	🟨 3	▨ 12	2

Sorting Algorithms | Insertion Sort

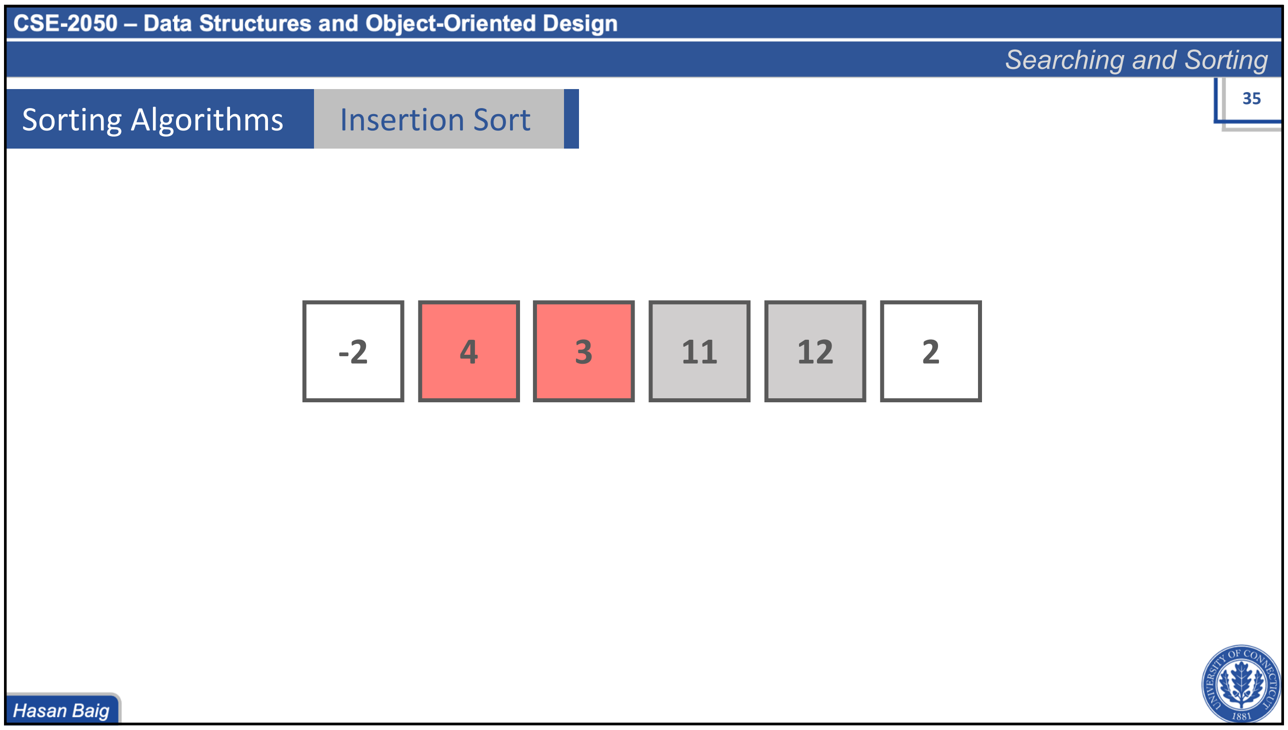


Sorting Algorithms | Insertion Sort

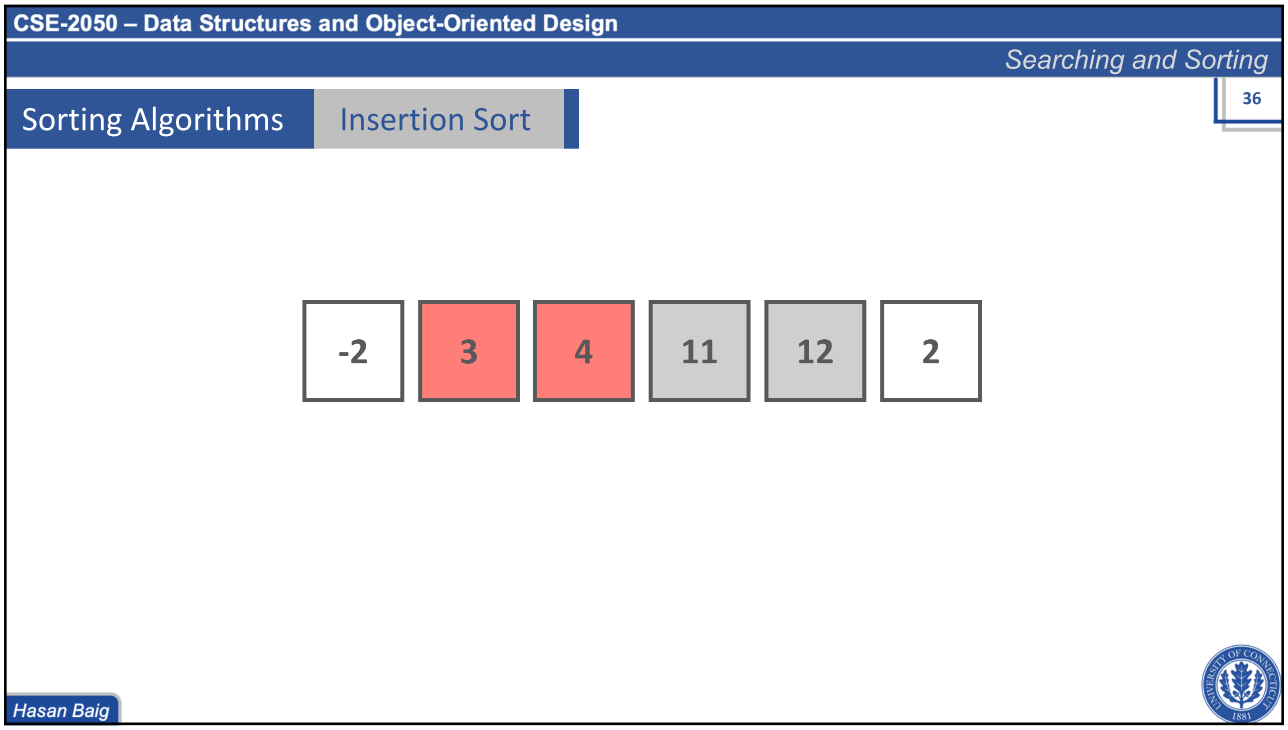




Sorting Algorithms | Insertion Sort

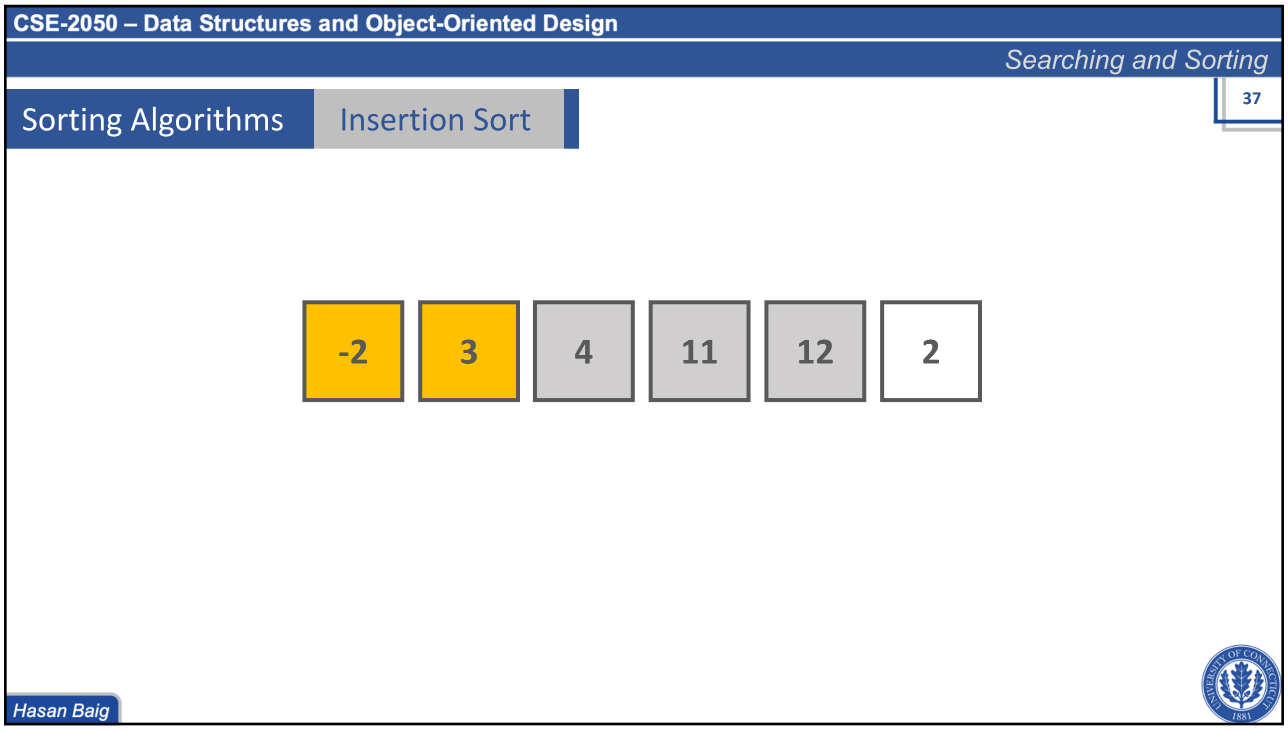


Sorting Algorithms | Insertion Sort



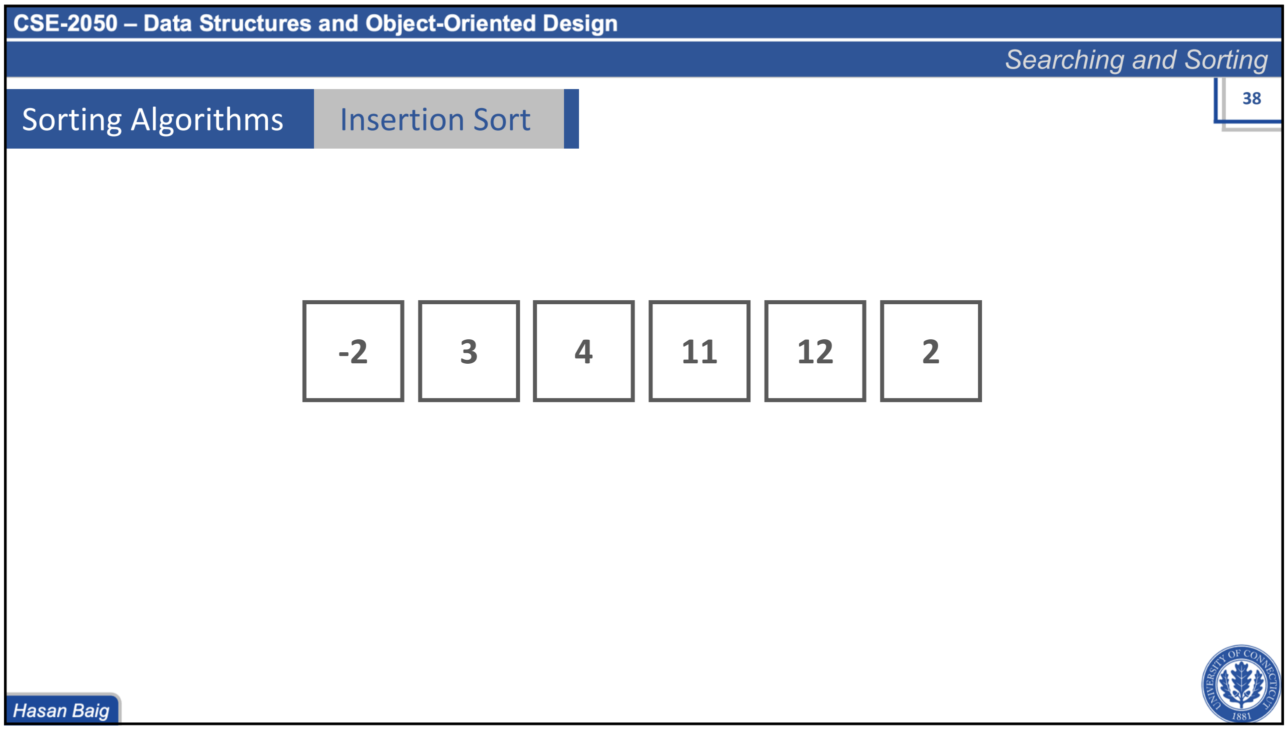
Sorting Algorithms | Insertion Sort

-2	3	4	11	12	2



Sorting Algorithms | Insertion Sort

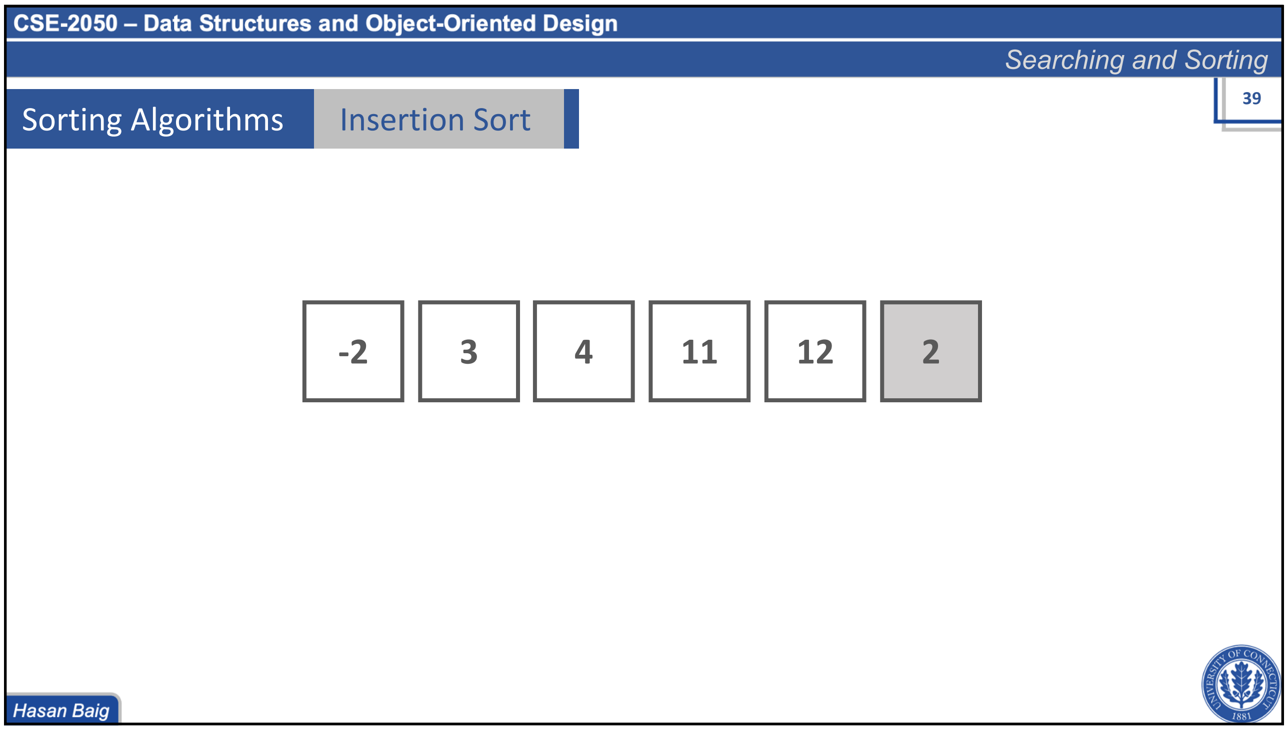
-2	3	4	11	12	2



Sorting Algorithms | Insertion Sort

-2	3	4	11	12	2

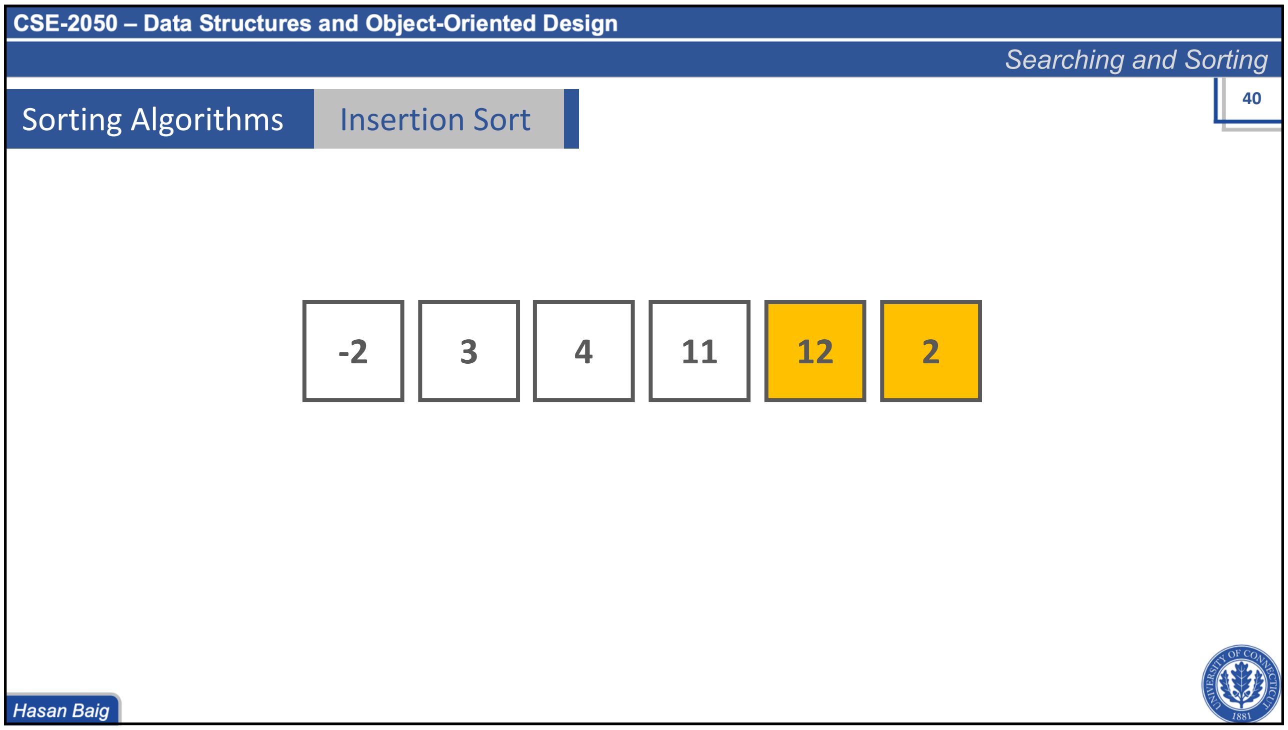
Hasan Baig



Sorting Algorithms | Insertion Sort

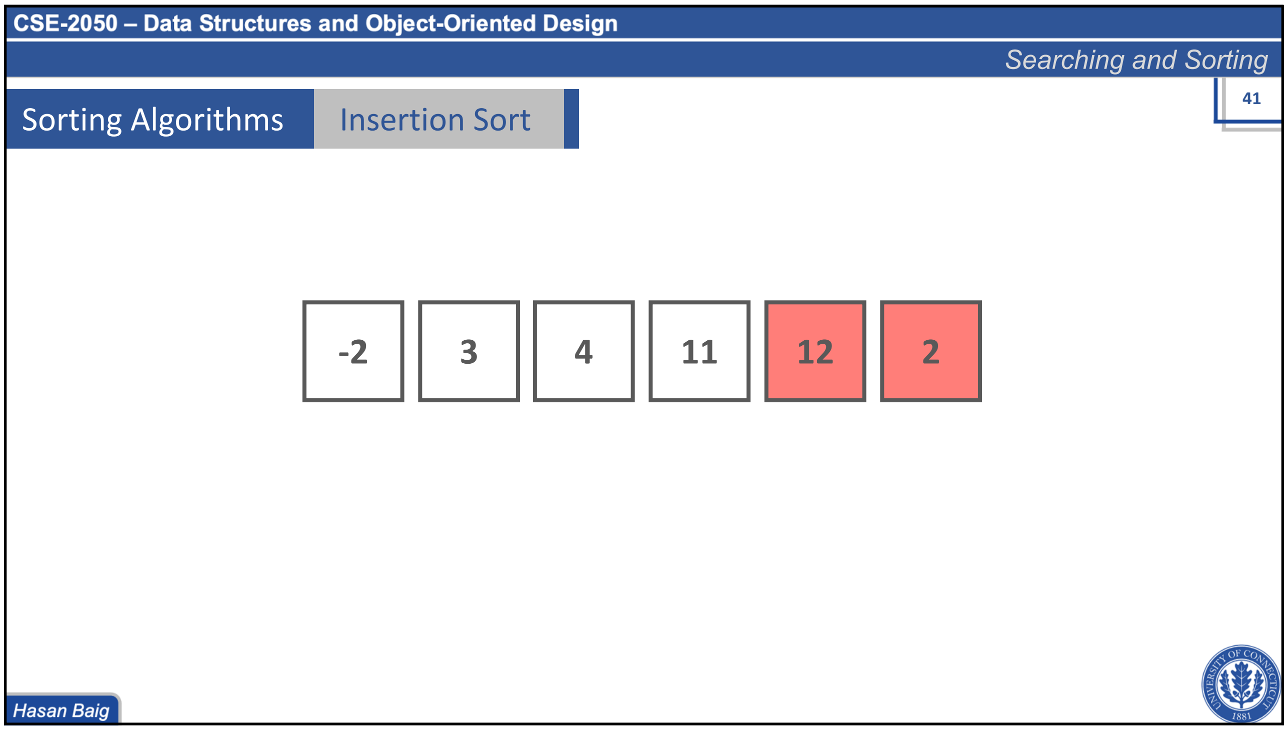
-2	3	4	11	12	▨ 2

Hasan Baig



Sorting Algorithms | Insertion Sort

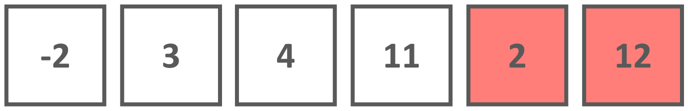
-2	3	4	11	12	2



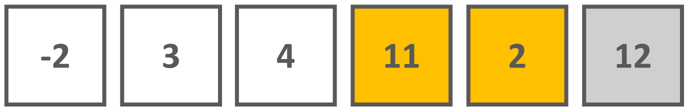
Sorting Algorithms | Insertion Sort

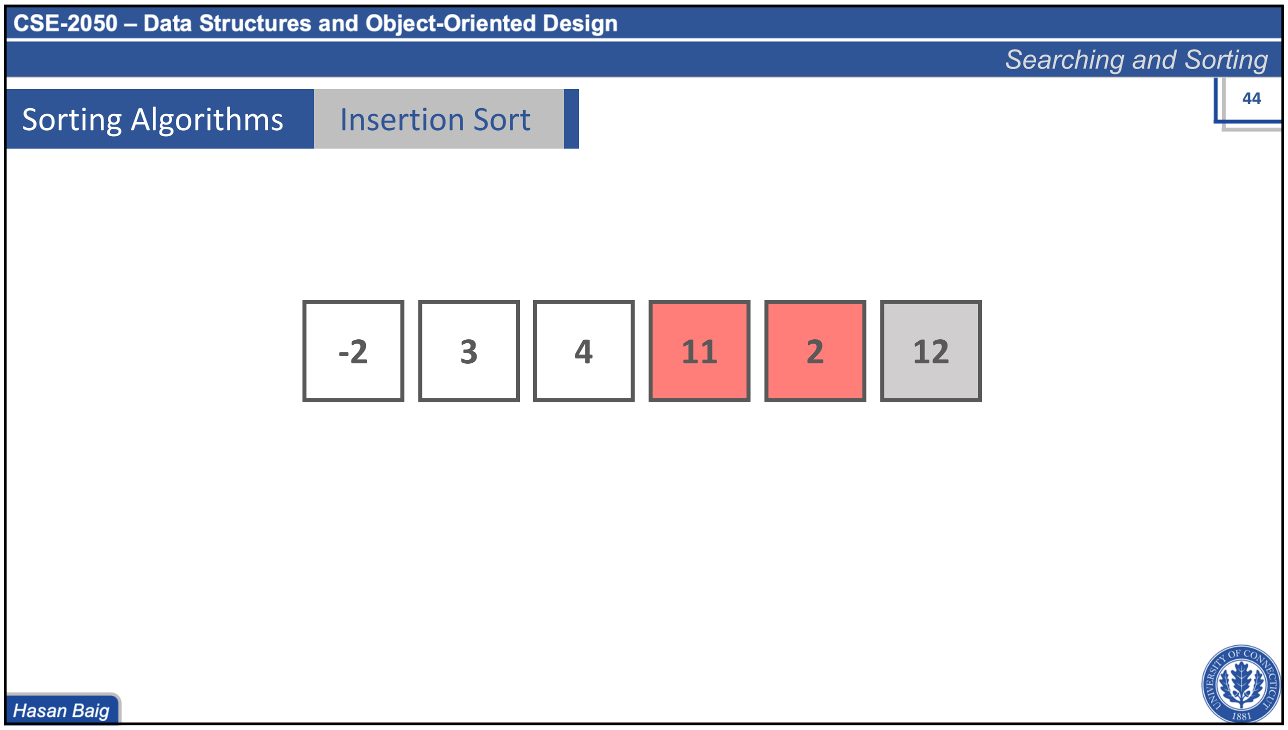
-2	3	4	11	12	2

Sorting Algorithms | Insertion Sort

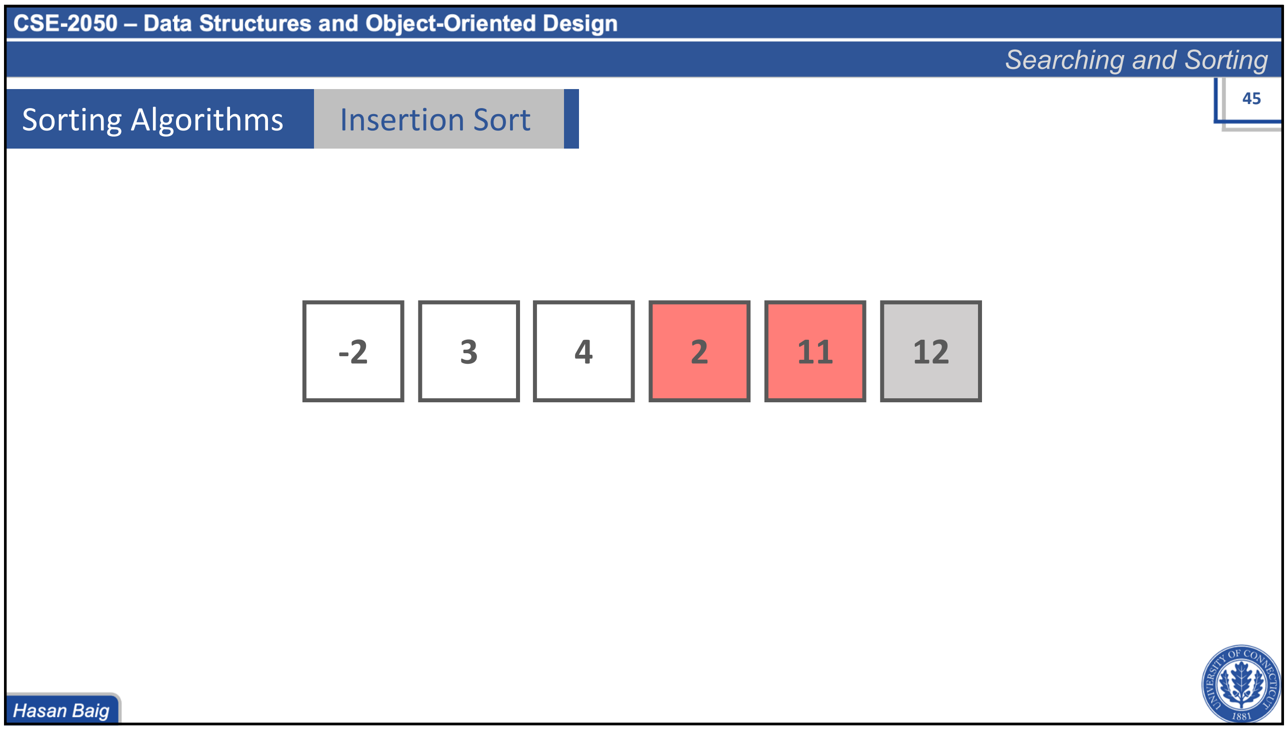


Sorting Algorithms | Insertion Sort



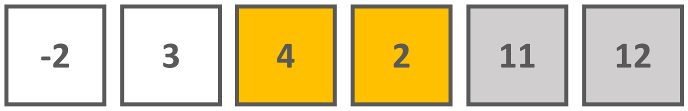


Sorting Algorithms | Insertion Sort

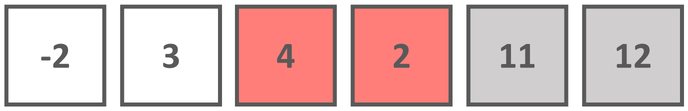


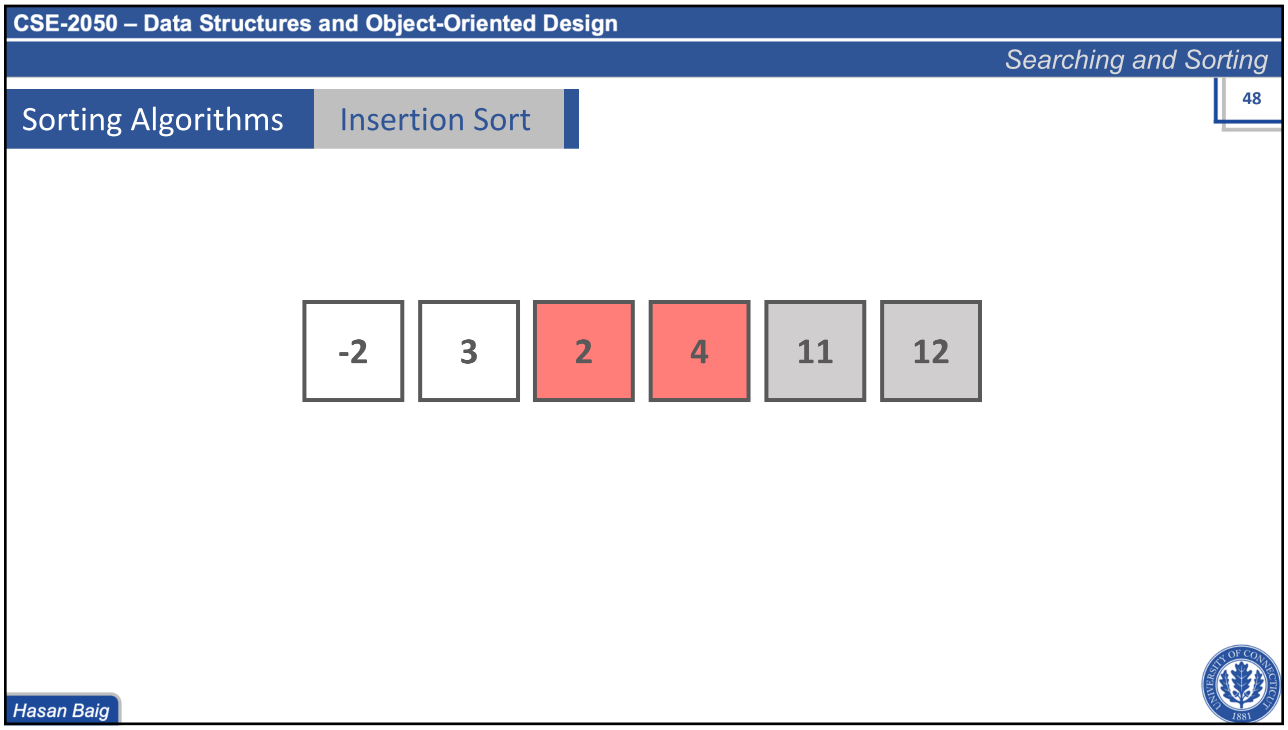
Sorting Algorithms | Insertion Sort

Sorting Algorithms | Insertion Sort



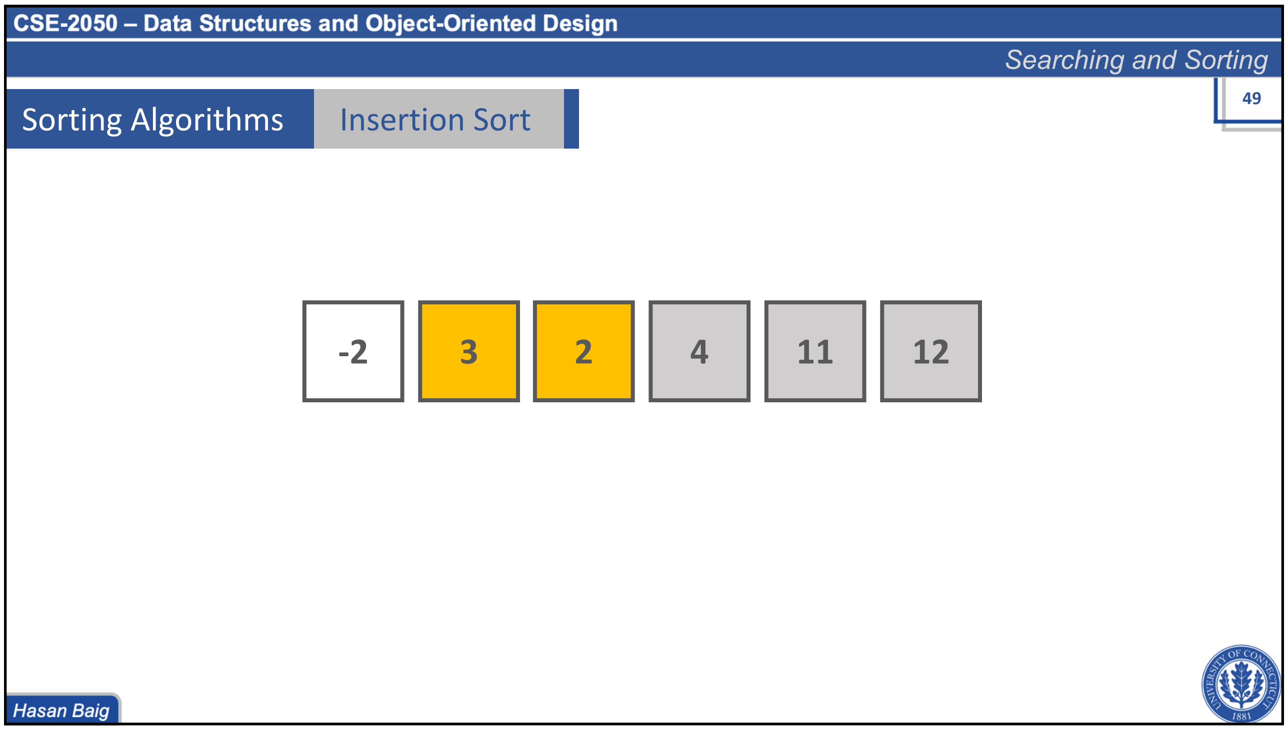
Sorting Algorithms | Insertion Sort





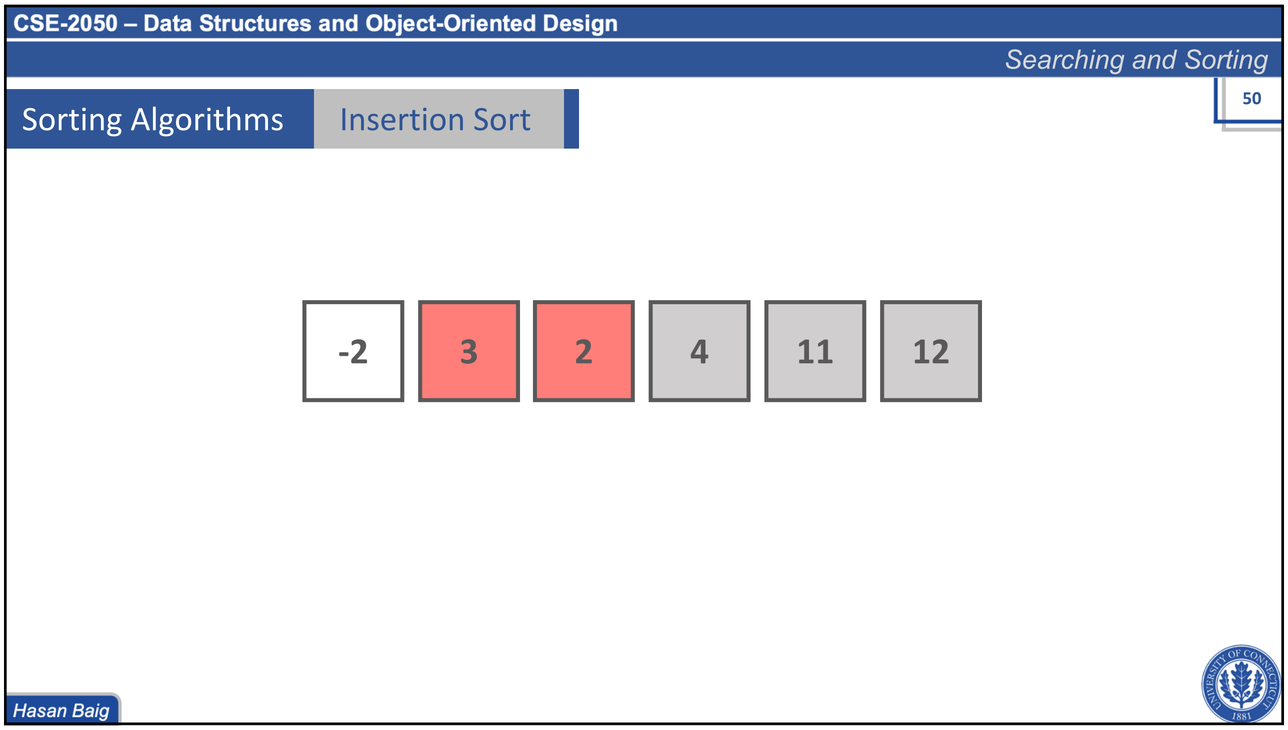
Sorting Algorithms | Insertion Sort

-2	3	2	4	11	12



Sorting Algorithms | Insertion Sort

-2	3	2	4	11	12



CSE-2050 – Data Structures and Object-Oriented Design

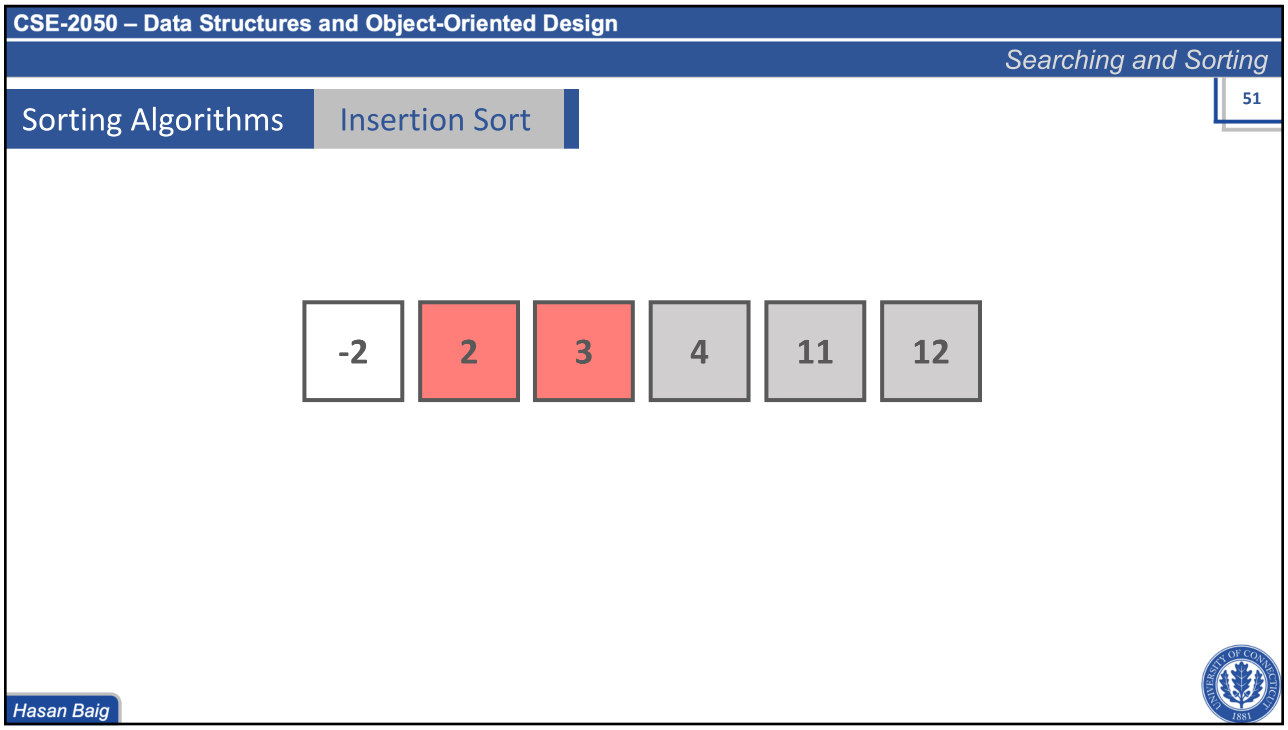
Searching and Sorting

50

Sorting Algorithms | Insertion Sort

-2	3	2	4	11	12

Hasan Baig



CSE-2050 – Data Structures and Object-Oriented Design

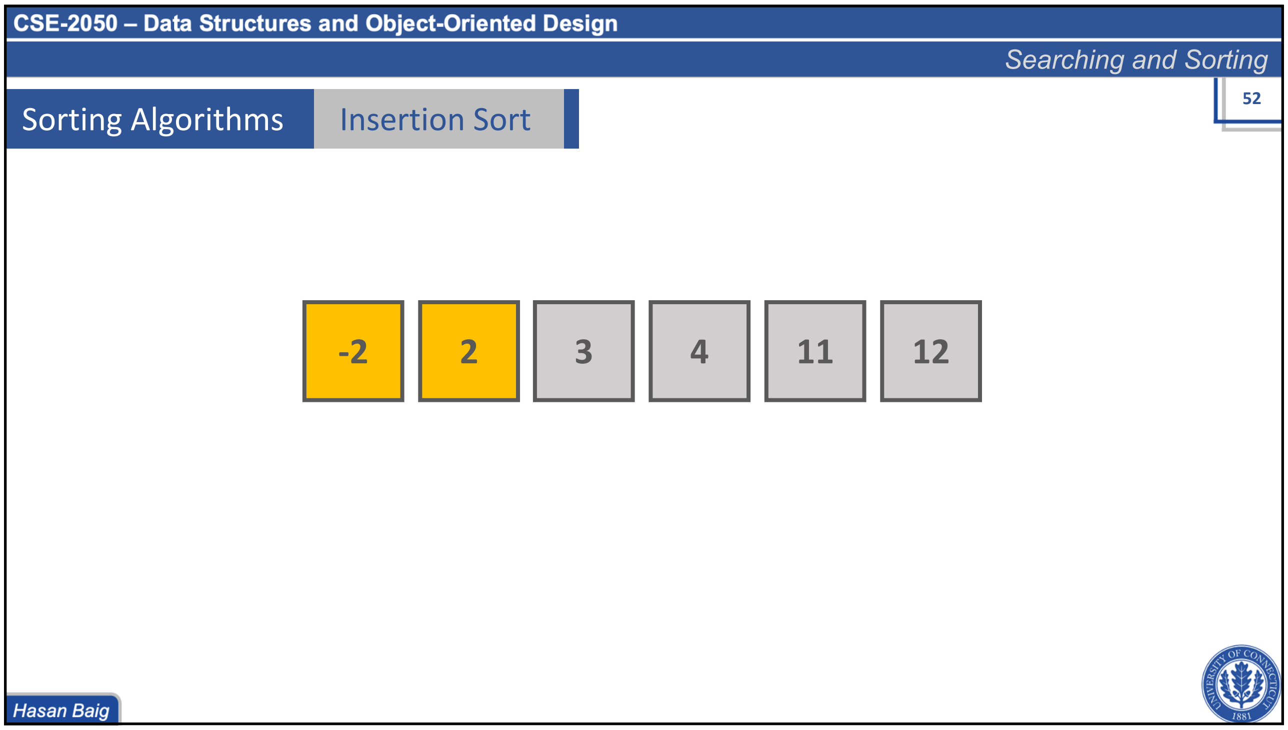
Searching and Sorting

51

Sorting Algorithms | Insertion Sort

-2	2	3	4	11	12

Hasan Baig



CSE-2050 – Data Structures and Object-Oriented Design

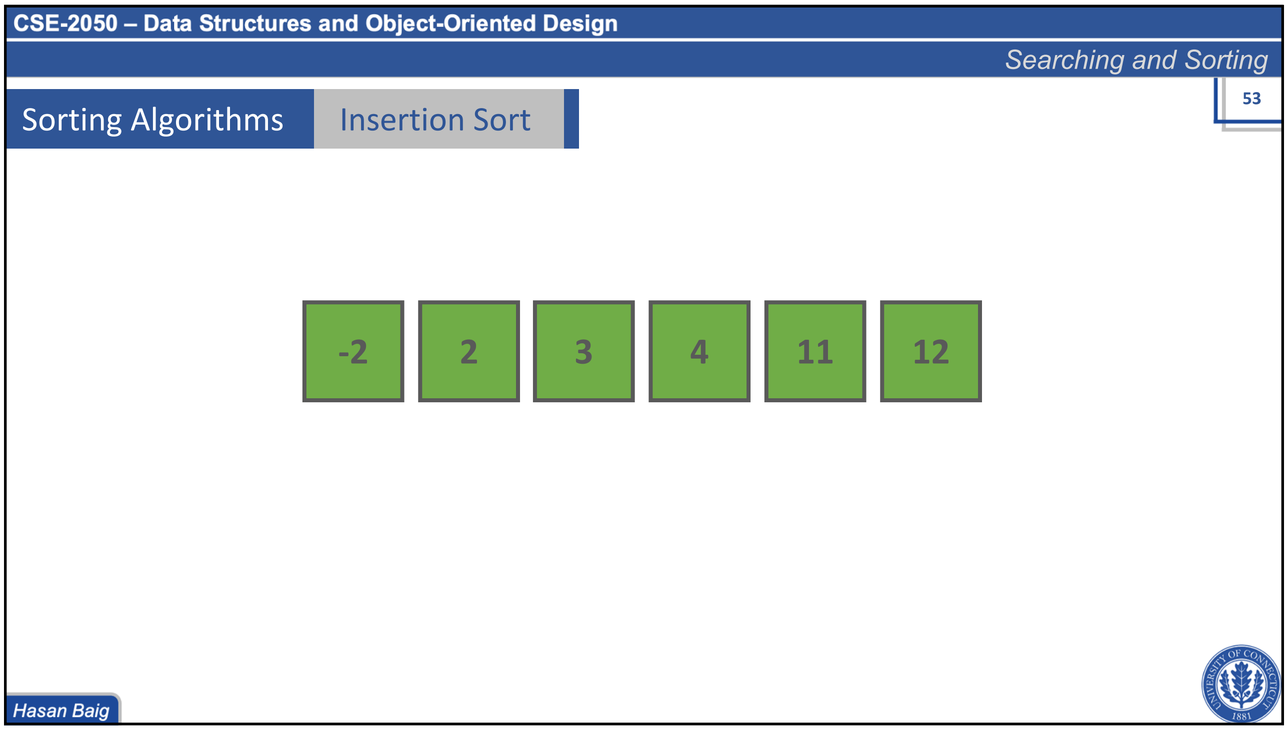
Searching and Sorting

52

Sorting Algorithms | Insertion Sort

-2	2	3	4	11	12

Hasan Baig



CSE-2050 – Data Structures and Object-Oriented Design

Searching and Sorting

53

Sorting Algorithms | Insertion Sort

-2	2	3	4	11	12

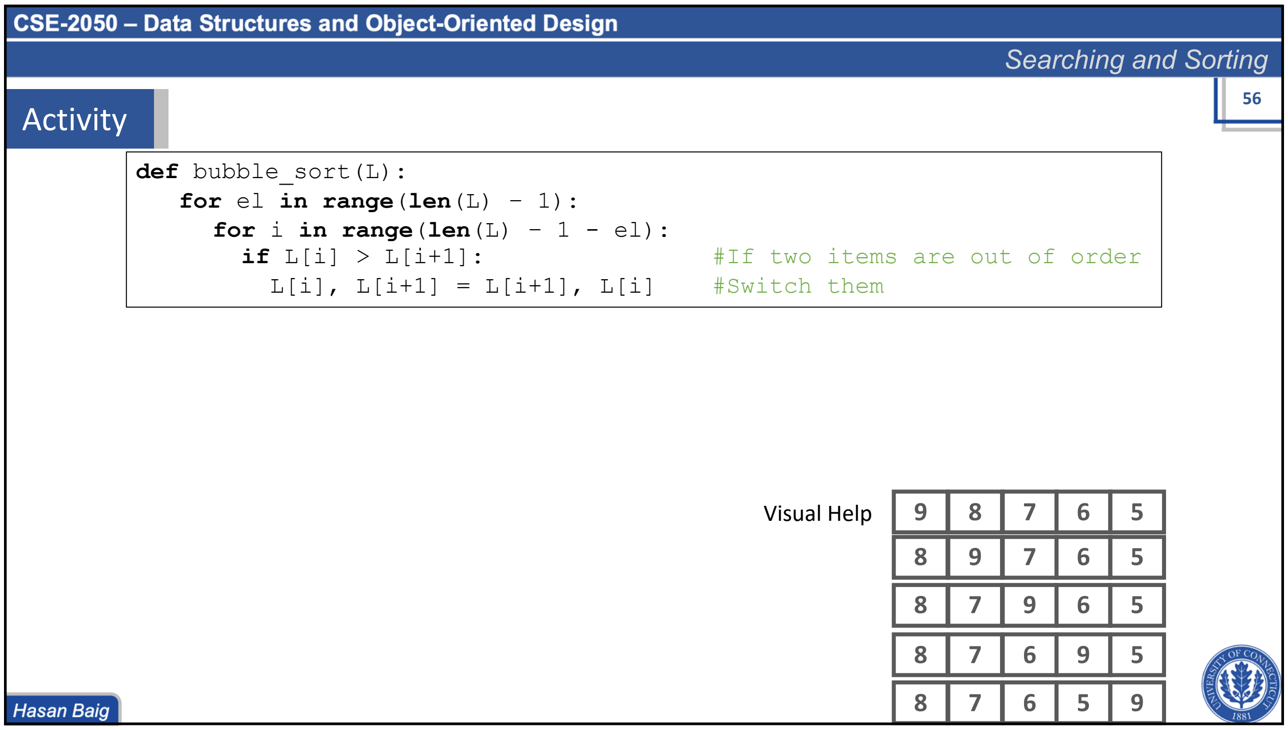
Hasan Baig

Summary of Quadratic Sorting Algorithms

- **Bubble sort**
  - Iterates over every pair in collection, swaps out of order pairs
  - After x iterations, the last x items are in their final (sorted) place
- **Selection sort**
  - Iterates over every unsorted item in collection, selects the next smallest/biggest
  - After x iterations, the last x items are in their final (sorted) place
- **Insertion sort**
  - Iterates over a progressively growing sorted section of the list
  - Bubbles the next un-sorted item into place
  - After x iterations, the first x items are sorted but may not be in their final place.

Summary of Quadratic Sorting Algorithms

- **In Bubble sort:**
  - Large items at the beginning move to their correct positions quickly – "Rabbits"
  - Small items at the end can only move one position/pass – "Turtles"
  - How can we do better?



CSE-2050 – Data Structures and Object-Oriented Design

Searching and Sorting

56

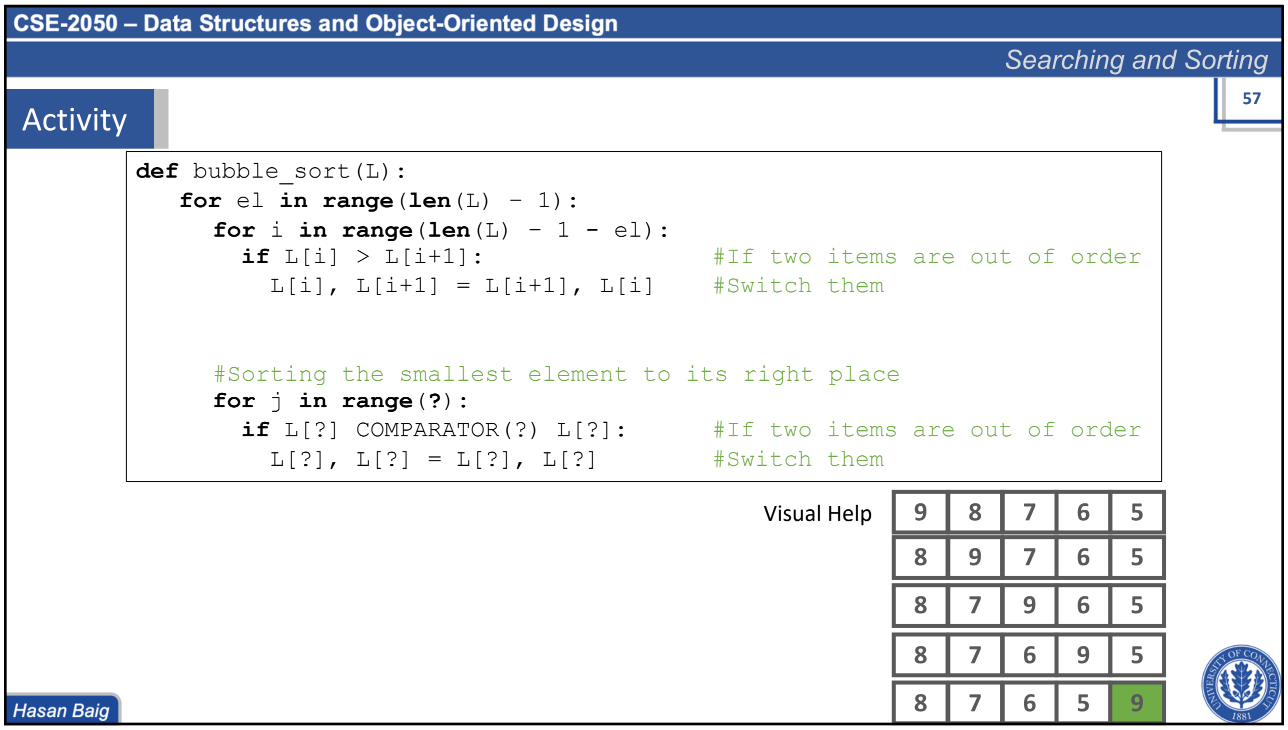
Activity

```
def bubble_sort(L):
   for el in range(len(L) - 1):
     for i in range(len(L) - 1 - el):
       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
```

Visual Help

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

Hasan Baig



CSE-2050 – Data Structures and Object-Oriented Design

Searching and Sorting

57

Activity

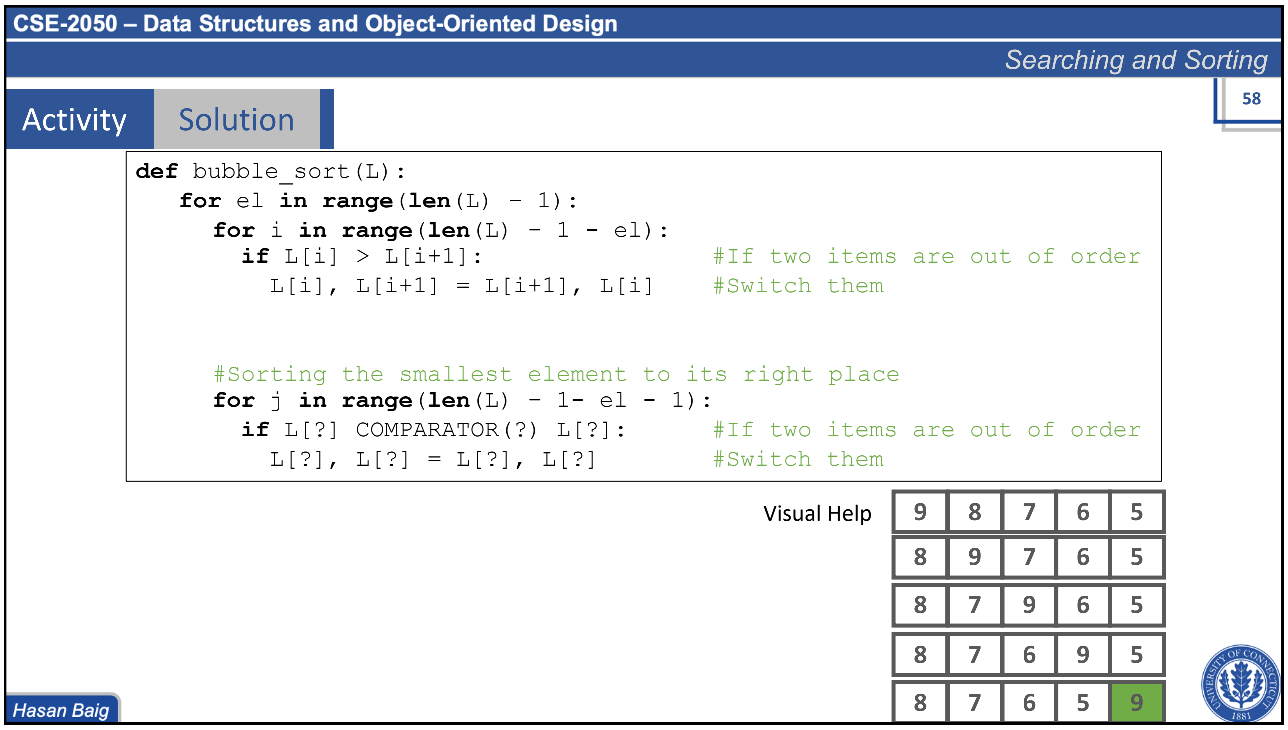
```
def bubble_sort(L):
   for el in range(len(L) - 1):
     for i in range(len(L) - 1 - el):
       if L[i] > L[i+1]:              #If two items are out of order
         L[i], L[i+1] = L[i+1], L[i]    #Switch them


     #Sorting the smallest element to its right place
     for j in range(?):
       if L[?] COMPARATOR(?) L[?]:    #If two items are out of order
         L[?], L[?] = L[?], L[?]        #Switch them
```

Visual Help

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

Hasan Baig



Activity Solution

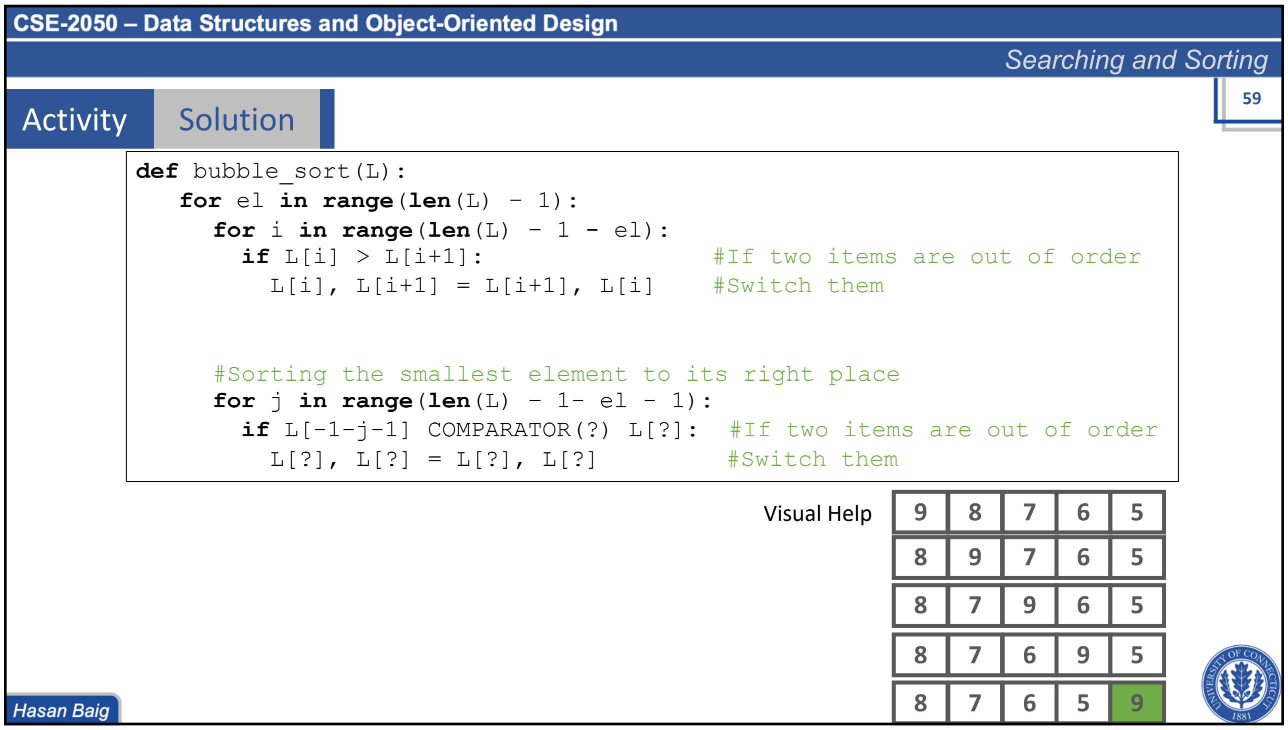
```
def bubble_sort(L):
    for el in range(len(L) - 1):
      for i in range(len(L) - 1 - el):
        if L[i] > L[i+1]:                #If two items are out of order
          L[i], L[i+1] = L[i+1], L[i]    #Switch them


      #Sorting the smallest element to its right place
      for j in range(len(L) - 1- el - 1):
        if L[?] COMPARATOR(?) L[?]:      #If two items are out of order
          L[?], L[?] = L[?], L[?]        #Switch them
```

Visual Help

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

Hasan Baig



Activity Solution

```
def bubble_sort(L):
    for el in range(len(L) - 1):
      for i in range(len(L) - 1 - el):
        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 the smallest element to its right place
      for j in range(len(L) - 1- el - 1):
        if L[-1-j-1] COMPARATOR(?) L[?]:  #If two items are out of order
          L[?], L[?] = L[?], L[?]         #Switch them
```

Visual Help

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

Hasan Baig

Activity Solution

```
def bubble_sort(L):
   for el in range(len(L) - 1):
     for i in range(len(L) - 1 - el):
       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 the smallest element to its right place
     for j in range(len(L) - 1- el - 1):
       if L[-1-j-1] COMPARATOR(?) L[-1-j-2]:  #If two items are out of order
         L[?], L[?] = L[?], L[?]               #Switch them
```

Visual Help

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

Activity Solution

```
def bubble_sort(L):
   for el in range(len(L) - 1):
     for i in range(len(L) - 1 - el):
       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 the smallest element to its right place
     for j in range(len(L) - 1- el - 1):
       if L[-1-j-1] < L[-1-j-2]:        #If two items are out of order
         L[?], L[?] = L[?], L[?]        #Switch them
```

Visual Help

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

Activity Solution

```
def bubble_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1 - el):
            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 the smallest element to its right place
        for j in range(len(L) - 1- el - 1):
            if L[-1-j-1] < L[-1-j-2]:              #If two items are out of order
                L[-1-j-1], L[-1-j-2] = L[-1-j-2], L[-1-j-1]         #Switch them
```

Visual Help

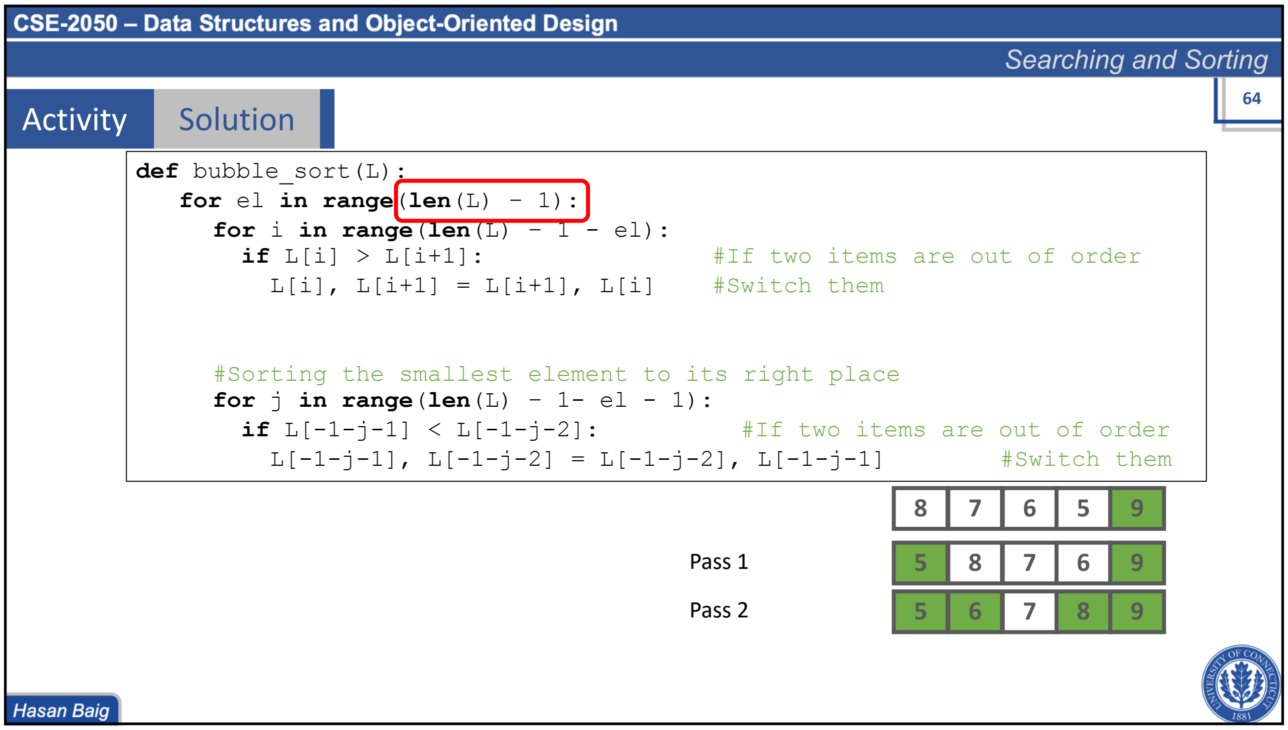
9	8	7	6	5
8	9	7	6	5
8	7	9	6	5
8	7	6	9	5
8	7	6	5	9

Activity Solution

```
def bubble_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1 - el):
            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 the smallest element to its right place
        for j in range(len(L) - 1- el - 1):
            if L[-1-j-1] < L[-1-j-2]:              #If two items are out of order
                L[-1-j-1], L[-1-j-2] = L[-1-j-2], L[-1-j-1]         #Switch them
```

	8	7	6	5	9
Pass 1	5	8	7	6	9
Pass 2	5	6	7	8	9



CSE-2050 – Data Structures and Object-Oriented Design

Searching and Sorting

64

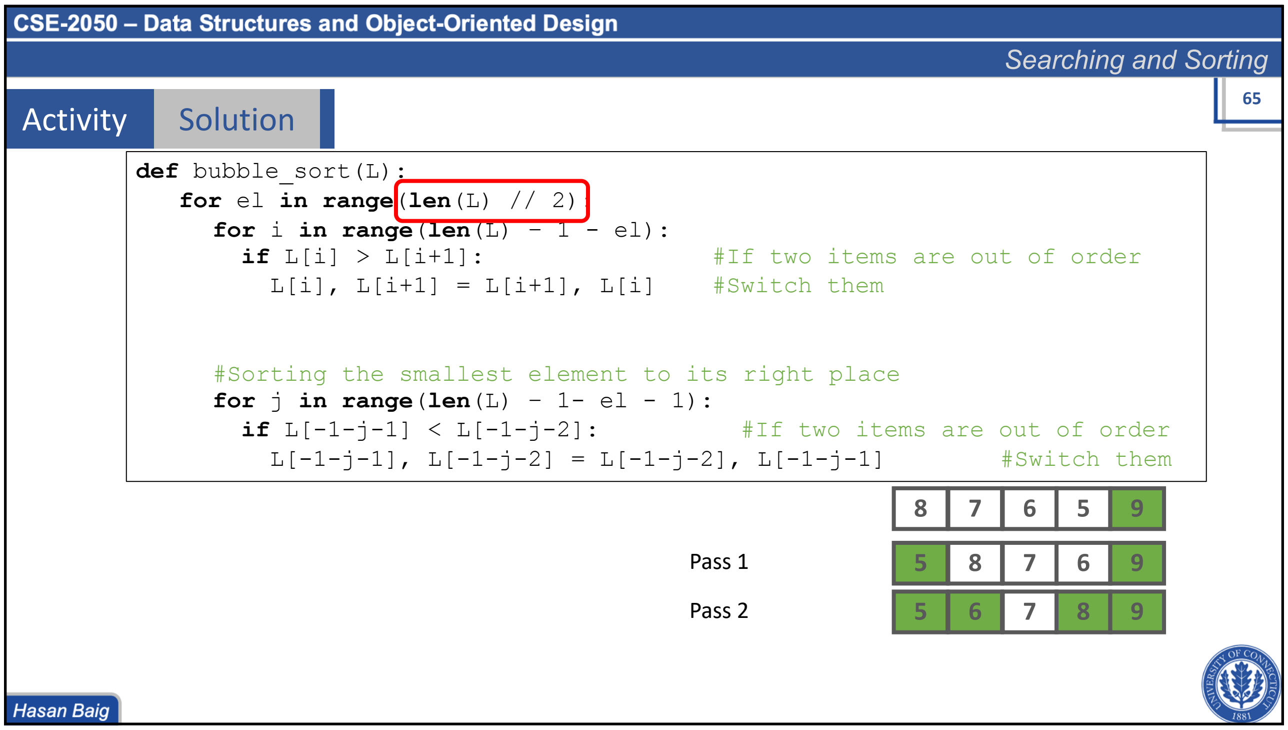
Activity Solution

```
def bubble_sort(L):
    for el in range(len(L) - 1):
        for i in range(len(L) - 1 - el):
            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 the smallest element to its right place
        for j in range(len(L) - 1- el - 1):
            if L[-1-j-1] < L[-1-j-2]:              #If two items are out of order
                L[-1-j-1], L[-1-j-2] = L[-1-j-2], L[-1-j-1]         #Switch them
```

	8	7	6	5	9
Pass 1	5	8	7	6	9
Pass 2	5	6	7	8	9

Hasan Baig



CSE-2050 – Data Structures and Object-Oriented Design

Searching and Sorting

65

Activity Solution

```
def bubble_sort(L):
    for el in range(len(L) // 2):
        for i in range(len(L) - 1 - el):
            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 the smallest element to its right place
        for j in range(len(L) - 1- el - 1):
            if L[-1-j-1] < L[-1-j-2]:              #If two items are out of order
                L[-1-j-1], L[-1-j-2] = L[-1-j-2], L[-1-j-1]         #Switch them
```

	8	7	6	5	9
Pass 1	5	8	7	6	9
Pass 2	5	6	7	8	9

Hasan Baig

Cocktail Sort

```
def cocktail_sort(L):
   for el in range(len(L) // 2):
     for i in range(len(L) - 1 - el):
       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 the smallest element to its right place
     for j in range(len(L) - 1- el - 1):
       if L[-1-j-1] < L[-1-j-2]:             #If two items are out of order
         L[-1-j-1], L[-1-j-2] = L[-1-j-2], L[-1-j-1]        #Switch them
```