# Week 3 Exercise

测验, 18 个问题

1 point

1。
Consider this code:

```
for i in range(m):
    for j in range(n):
        print()
```

How many times is function `print` called?

- (●) `m * n`
- ( ) `m + n`
- ( ) `n`
- ( ) `m`

---

1 point

2。
Consider this code:

```
for i in range(m):
    print()

for j in range(n):
    print()
```

How many times is function `print` called?

- ( ) `m`
- (●) `m + n`
- ( ) `m * n`

# Week 3 Exercise

测验, 18 个问题

1 point

3。
Assume variable `L` refers to a list of items.

You have a problem you are trying to solve and you figured out two different approaches that would work.

```
# Approach 1:
for i in range(len(L)):
    for j in range(len(L)):
        # do a few assignment statements to accomplish
            the task.
```

```
# Approach 2:
for i in range(1000):
    for j in range(len(L)):
        # do a few assignment statements to accomplish
            the task.
```

When would **Approach 2** take fewer iterations than **Approach 1**?

- When `L` contains strings.
- When `L` is sorted.
- When `L` has more than 1000 items.
- When `L` has exactly 1000 items.

1 point

4。
For **linear search**, if we are searching for `7` , which list will cause the fewest number of iterations?

- `[2, 3, 4, 5, 6, 7]`
- `[6, 7, 4, 5, 2, 3]`
- `[2, 4, 6, 3, 5, 7]`
- `[7, 6, 5, 4, 3, 2]`

# Week 3 Exercise

测验, 18 个问题

1 point

5。

The list `[4, 2, 5, 6, 7, 3, 1]` is shown below after each pass of a sorting algorithm:

```
1  [1, 2, 5, 6, 7, 3, 4]
2  [1, 2, 5, 6, 7, 3, 4]
3  [1, 2, 3, 6, 7, 5, 4]
4  [1, 2, 3, 4, 7, 5, 6]
5  [1, 2, 3, 4, 5, 7, 6]
6  [1, 2, 3, 4, 5, 6, 7]
7  [1, 2, 3, 4, 5, 6, 7]
```

Which sorting algorithm is being executed?

- ○ insertion sort
- ◉ selection sort
- ○ bubble sort

---

1 point

6。

The list `[4, 2, 5, 6, 7, 3, 1]` is shown below after each pass of a sorting algorithm:

```
1  [2, 4, 5, 6, 3, 1, 7]
2  [2, 4, 5, 3, 1, 6, 7]
3  [2, 4, 3, 1, 5, 6, 7]
4  [2, 3, 1, 4, 5, 6, 7]
5  [2, 1, 3, 4, 5, 6, 7]
6  [1, 2, 3, 4, 5, 6, 7]
```

Which sorting algorithm is being executed?

- ○ insertion sort
- ○ selection sort
- ◉ bubble sort

---

1 point

# Week 3 Exercise

测验, 18 个问题

7。

The list `[4, 2, 5, 6, 7, 3, 1]` is shown below after each pass of a sorting algorithm:

```
1  [4, 2, 5, 6, 7, 3, 1]
2  [2, 4, 5, 6, 7, 3, 1]
3  [2, 4, 5, 6, 7, 3, 1]
4  [2, 4, 5, 6, 7, 3, 1]
5  [2, 4, 5, 6, 7, 3, 1]
6  [2, 3, 4, 5, 6, 7, 1]
7  [1, 2, 3, 4, 5, 6, 7]
```

Which sorting algorithm is being executed?

- ○ bubble sort
- ◉ insertion sort
- ○ selection sort

---

1 point

8。

List `[1, 5, 8, 7, 6, 1, 7]` is being sorted using **selection sort**. Here is what the list will look like after each of the first three passes:

- After the 1st pass: `[1, 5, 8, 7, 6, 1, 7]`
- After the 2nd pass: `[1, 1, 8, 7, 6, 5, 7]`
- After the 3rd pass: `[1, 1, 5, 7, 6, 8, 7]`

What will the list look like after the 4th pass?

- ○ `[1, 1, 5, 7, 6, 8, 7]`
- ◉ `[1, 1, 5, 6, 7, 8, 7]`
- ○ `[1, 1, 5, 6, 7, 7, 8]`

---

1 point

9。

# Week 3 Exercise

测验, 18 个问题

List `[6, 8, 2, 1, 1, 9, 4]` is being sorted using **insertion sort**. Here is what the list will look like after each of the first three passes:

- After the 1st pass: `[6, 8, 2, 1, 1, 9, 4]`
- After the 2nd pass: `[6, 8, 2, 1, 1, 9, 4]`
- After the 3rd pass: `[2, 6, 8, 1, 1, 9, 4]`

What will the list look like after the 4th pass?

- ○ `[1, 1, 2, 6, 8, 9, 4]`
- ○ `[1, 6, 8, 2, 1, 9, 4]`
- ● `[1, 2, 6, 8, 1, 9, 4]`

---

1 point

10。

In **bubble sort**, on the first pass through the list, which item gets moved to the far right?

- ● The largest item.
- ○ The item that was originally at the second-last index.
- ○ The item that was originally at index `0` .
- ○ The smallest item.
- ○ An odd number.

---

1 point

11。

# Week 3 Exercise

测验, 18 个问题

Here is the code for function **insert** with docstring and comments removed:

```
def insert(L, i):
    value = L[i]

    j = i
    while j != 0 and L[j - 1] > value:
        L[j] = L[j - 1]
        j = j - 1

    L[j] = value
```

In the following list, there is an **x** at index 5 . In this question, you will choose a value for that variable.

```
L = [2, 5, 6, 7, 8, x, 4]
```

The first 5 items are sorted.

If we call **insert(L, 5)** , that unknown value will be inserted into the sorted section, growing the sorted section by 1 item. Select a value for **x** that would be moved all the way to index 0 in the list.

- 1
- 3
- 9
- 4

---

1 point

12。

# Week 3 Exercise

测验, 18 个问题

Here is the code for function **insert** with docstring and comments removed:

```
def insert(L, i):
    value = L[i]

    j = i
    while j != 0 and L[j - 1] > value:
        L[j] = L[j - 1]
        j = j - 1

    L[j] = value
```

In the following list, there is an **x** at index 5 . In this question, you will choose a value for that variable.

```
L = [2, 5, 6, 7, 8, x, 4]
```

The first 5 items are sorted.

If we call **insert(L, 5)** , that unknown value will be inserted into the sorted section, growing the sorted section by 1 item. Select the value for **x** that would not move.

- 3
- 4
- 0
- 9

---

1 point

13。

# Week 3 Exercise

测验, 18 个问题

Here is the code for function **insert** with docstring and comments removed:

```
def insert(L, i):
    value = L[i]

    j = i
    while j != 0 and L[j - 1] > value:
        L[j] = L[j - 1]
        j = j - 1

    L[j] = value
```

In general, function call **insert(L, i)** might move the item at index **i** all the way to index 0 in the list (if that item is smaller than everything in the sorted section); it might not move it at all (if that item is larger than everything in the sorted section); or it might be moved partway (if that item is neither smaller nor larger than everything in the sorted section).

The while loop can be terminated for one of two reasons: **j == 0** or **L[j - 1] <= value** . In which situation does the loop terminate because **j == 0** ?

- (●) When the item at index **i** is smaller than everything in the sorted section.
- ( ) When the item at index **i** is larger than everything in the sorted section.
- ( ) When the item at index **i** is neither smaller nor larger than everything in the sorted section.

---

1 point

14。

# Week 3 Exercise

测验, 18 个问题

Here is the code for function **insert** :

```
1  def insert(L, i):
2      value = L[i]
3
4      j = i
5      while j != 0 and L[j - 1] > value:
6          L[j] = L[j - 1]
7          j = j - 1
8
9     L[j] = value
```

For function call **insert(L, i)** , in the worst case, the item at index **i** is moved all the way to index **0** . Variable **j** starts off at **i** and is decreased by 1 on each iteration of the while loop until it reaches **0** .

In this worst-case situation, how many times is the body of the while loop executed?

- **i**
- **2 * i**
- **i + 1**
- **i - 1**

---

1 point

15。

# Week 3 Exercise

测验, 18 个问题

Here is the code for function `insertion_sort`:

```
1  def insertion_sort(L):
2      for i in range(len(L)):
3          insert(L, i)
```

This question is about the *worst-case* running time for this code. (The worst case for insertion sort happens when a list is sorted in reverse, from largest to smallest.)

- On the first iteration of this loop, `i` refers to `0`, so `insert(L, 0)` is called, and the while loop in function `insert` iterates 0 times.
- On the second iteration, `insert(L, 1)` is called, and the while loop in function `insert` iterates 1 time.
- On the last iteration, `insert(L, len(L) - 1)` is called, and the while loop in function `insert` iterates `len(L) - 1` times.

In total, how many times is the body of the while loop in function `insert` executed during one call on function `insertion_sort`?

- (●) `0 + 1 + ... + len(L) - 1`
- ( ) `5 + 10 + ... + 5 * (len(L) - 1)`
- ( ) `len(L) * (0 + 1 + ... + len(L) - 1)`
- ( ) `len(L) - 1`

---

1 point

16。

# Week 3 Exercise

测验, 18 个问题

In the worst case, on a call on function `insertion_sort(L)`, the total number of times the loop body in function `insert` is executed is this:

```
0 + 1 + 2 + 3 + ... + (len(L) - 3) + (len(L) - 2) +
    (len(L) - 1)
```

The 0 doesn't affect the sum, so we can simplify to this:

```
1 + 2 + 3 + ... + (len(L) - 3) + (len(L) - 2) + (len(L
    ) - 1)
```

We can add the first and last items together, and the second and second-last items together, and so on:

```
1 + (len(L) - 1)    # The 1 and the -1 cancel, leaving
    len(L)
+ 2 + (len(L) - 2)    # The 2 and the -2 cancel,
    leaving len(L)
+ 3 + (len(L) - 3)    # The 3 and the -3 cancel,
    leaving len(L)+ ...
```

Every line in the equation adds up to `len(L)`.

***Roughly*** how many lines in the equation are there, and what is the total number of times the loop body is executed?

(Hint: work this out using a smaller example, such as a length **9** list, and then generalize.)

- (●) Number of lines: `len(L) / 2`

  Total number of times the loop body is executed: `len(L) * len(L) / 2`

- ( ) Number of lines: `len(L) / 10`

  Total number of times the loop body is executed: `len(L) * len(L) / 10`

- ( ) Number of lines: `len(L)`

  Total number of times the loop body is executed: `len(L) * len(L)`

---

1 point

17。

# Week 3 Exercise

测验, 18 个问题

For a call on function `insertion_sort(L)` , in the *worst case*, select the running time:

- (●) Quadratic in the length of list `L` .
- ( ) Linear in the length of list `L` .
- ( ) The running time is not proportional to the length of list `L` .

---

1 point

18。

For a call on function `insertion_sort(L)` , in the *best case* (where list `L` is already sorted), how many times is the body of the while loop in function `insert` executed?

- ( ) `10 * len(L)`
- (●) `0`
- ( ) `len(L)`
- ( ) `len(L) * len(L) / 2`

---

☑ 我（**伟臣 沈**）了解提交不是我自己完成的作业 将永远不会通过此课程或导致我的 Coursera 帐号被关闭。 了解荣誉准则的更多信息

提交测试

---