edX.org will be down for maintenance starting at 5:00AM UTC-4, Monday 3/9. Maintenance will last approximately 2 hours. Updates will be available via https://twitter.com/edXstatus.



MITx: 6.00.1x Introduction to Computer Science and Programming Using Pyth...



Courseware

**Updates & News** 

Calendar

Wiki Discussion

**Progress** 

Below are four different functions for sorting a list of elements in increasing order. For simplicity, assume that the list only contain ints. For each, we are going to ask you about how the algorithm creates its output and about the worst case time complexity - or order of growth - of the algorithm. Answer the questions without running the code on your computer.

#### PROBLEM 3-1

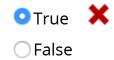
Answer the following 5 questions based on this code.

```
def sort1(lst):
    swapFlag = True
    iteration = 0
    while swapFlag:
        swapFlag = False
        for i in range(len(lst)-1):
            if lst[i] > lst[i+1]:
                temp = lst[i+1]
            lst[i+1] = lst[i]
            lst[i] = temp
            swapFlag = True

L = lst[:] # the next 3 questions assume this line just executed
        iteration += 1
    return lst
```

# PROBLEM 3-1 A (1 point possible)

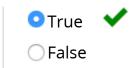
When we reach the marked spot in the code, and the variable <u>iteration</u> has value <u>n</u>, the smallest <u>n+1</u> elements of the sorted version of <u>lst</u> are in <u>L</u> in the correct order.



You have used 1 of 1 submissions

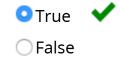
# PROBLEM 3-1 B (1/1 point)

When we reach the marked spot in the code, and the variable <u>iteration</u> has value <u>n</u>, the largest <u>n+1</u> elements of the sorted version of <u>lst</u> are in <u>L</u> in the correct order.



### PROBLEM 3-1 C (1/1 point)

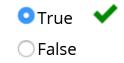
When we reach the marked spot in the code, and the variable <u>iteration</u> has value <u>n</u>, the first <u>n+1</u> elements of the original list, <u>lst</u>, appear in the correctly sorted places in L. The "correctly sorted places" refers to the order of the elements in the list, not the index.



You have used 1 of 1 submissions

### PROBLEM 3-1 D (1/1 point)

The function sorts the list list in place without using a new list.



You have used 1 of 1 submissions

### PROBLEM 3-1 E (1 point possible)

The complexity of this algorithm is:



You have used 1 of 1 submissions

#### PROBLEM 3-2

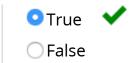
Answer the following 5 questions based on this code.

```
def sort2(lst):
    for iteration in range(len(lst)):
        minIndex = iteration
        minValue = lst[iteration]
        for j in range(iteration+1, len(lst)):
            if lst[j] < minValue:
                minIndex = j
                minValue = lst[j]
        temp = lst[iteration]
        lst[iteration] = minValue
        lst[minIndex] = temp</pre>
L = lst[:] # the next 3 questions assume this line just executed
```

return 1st

# PROBLEM 3-2 A (1/1 point)

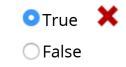
When we reach the marked spot in the code, and the variable <u>iteration</u> has value <u>n</u>, the smallest <u>n+1</u> elements of the sorted version of <u>lst</u> are in <u>L</u> in the correct order.



You have used 1 of 1 submissions

## PROBLEM 3-2 B (1 point possible)

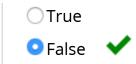
When we reach the marked spot in the code, and the variable iteration has value n, the largest n+1 elements of the sorted version of lst are in L in the correct order.



You have used 1 of 1 submissions

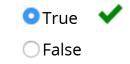
### PROBLEM 3-2 C (1/1 point)

When we reach the marked spot in the code, and the variable <u>iteration</u> has value <u>n</u>, the first <u>n+1</u> elements of the original list, <u>lst</u>, appear in the correctly sorted places in <u>L</u>. The "correctly sorted places" refers to the order of the elements in the list, not the index.



# PROBLEM 3-2 D (1/1 point)

The function sorts the list <code>lst</code> in place without using a new list.



You have used 1 of 1 submissions

# PROBLEM 3-2 E (1 point possible)

The complexity of this algorithm is:



You have used 1 of 1 submissions

### PROBLEM 3-3

Answer the following 5 questions based on this code.

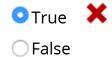
```
def sort3(lst):
    out = []
    for iteration in range(0,len(lst)):
        new = lst[iteration]
        inserted = False
        for j in range(len(out)):
            if new < out[j]:
                out.insert(j, new)
                inserted = True
                break
    if not inserted:
        out.append(new)

L = out[:] # the next 3 questions assume this line just executed</pre>
```

return out

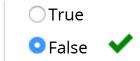
# PROBLEM 3-3 A (1 point possible)

When we reach the marked spot in the code, and the variable <u>iteration</u> has value <u>n</u>, the smallest <u>n+1</u> elements of the sorted version of <u>lst</u> are in <u>L</u> in the correct order.



You have used 1 of 1 submissions

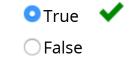
When we reach the marked spot in the code, and the variable <u>iteration</u> has value <u>n</u>, the largest <u>n+1</u> elements of the sorted version of <u>lst</u> are in <u>L</u> in the correct order.



You have used 1 of 1 submissions

## PROBLEM 3-3 C (1/1 point)

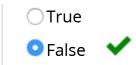
When we reach the marked spot in the code, and the variable <u>iteration</u> has value <u>n</u>, the first <u>n+1</u> elements of the original list, <u>lst</u>, appear in the correctly sorted places in <u>L</u>. The "correctly sorted places" refers to the order of the elements in the list, not the index.



You have used 1 of 1 submissions

### PROBLEM 3-3 D (1/1 point)

The function sorts the list <code>lst</code> in place without creating a new list.



PROBLEM 3-3 E (1 point possible)

The complexity of this algorithm is:



You have used 1 of 1 submissions

# PROBLEM 3-4

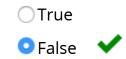
Answer the following 5 questions based on this code.

```
def sort4(lst):
    def unite(11, 12):
        if len(11) == 0:
            return 12
        elif len(12) == 0:
            return 11
        elif 11[0] < 12[0]:
            return [11[0]] + unite(11[1:], 12)
        else:
            return [12[0]] + unite(11, 12[1:])
    if len(lst) == 0 or len(lst) == 1:
        return 1st
    else:
        front = sort4(lst[:len(lst)/2])
        back = sort4(lst[len(lst)/2:])
        L = 1st[:] # the next 3 questions assume this line just executed
```

return unite(front, back)

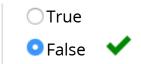
## PROBLEM 3-4 A (1/1 point)

When we reach the marked spot in the code on the n th recursive call of sort4, the smallest n+1 elements of the sorted version of lst are in l in the correct order.



### PROBLEM 3-4 B (1/1 point)

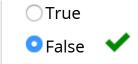
When we reach the marked spot in the code on the n th recursive call of sort4, the largest n+1 elements of the sorted version of lst are in L in the correct order.



You have used 1 of 1 submissions

## PROBLEM 3-4 C (1/1 point)

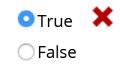
When we reach the marked spot in the code on the n th recursive call of sort4, the first n+1 elements of the original list, lst, appear in the correctly sorted places in L. The "correctly sorted places" refers to the order of the elements in the list, not the index.



You have used 1 of 1 submissions

### PROBLEM 3-4 D (1 point possible)

The function sorts the list lst in place without creating a new list.



PROBLEM 3-4 E (1/1 point)

The complexity of this algorithm is:

O(n log n) 🛊

You have used 1 of 1 submissions





EdX offers interactive online classes and MOOCs from the world's best universities. Online courses from MITx, HarvardX, BerkeleyX, UTx and many other universities. Topics include biology, business, chemistry, computer science, economics, finance, electronics, engineering, food and nutrition, history, humanities, law, literature, math, medicine, music, philosophy, physics, science, statistics and more. EdX is a non-profit online initiative created by founding partners Harvard and MIT.

© 2015 edX Inc.

#### About edX

About

News

Contact

FAQ

edX Blog

Donate to edX

#### Follow Us



Twitter



Facebook



Meetup



LinkedIn



**Q**+ Google+

EdX, Open edX, and the edX and Open edX logos are registered trademarks or trademarks of edX lnc.

Jobs at edX

Terms of Service and Honor Code

Privacy Policy (Revised 10/22/2014)