sort() is a method of list object, therefore x.sort() will change x.

sorted is an indepenct function. It will create a new list and return it.

PROBLEM 2-1

(1 point possible)

Examine the following code and answer the following question without running it.

x = [2,1]

y = [2,1]

z = False

if x == y:

if sorted(x) == sorted(y):

if x.sort() == y.sort():

z = x.sort() == sorted(y)

print z

The preceding Python statements will print False because

Top of Form

x is not equal to ysorted(x) is not equal to sorted(y)x.sort() is not equal to y.sort()**x.sort() is not equal to sorted(y)**

Bottom of Form

**Hide Answer**Reveal Answer

*You have used 0 of 1 submissions*

PROBLEM 2-2

(1 point possible)

Pick the word below that best matches this item: generator

Top of Form

Iteration AssertionDivide-and-conquer***O*(*n*)**Attribute

Bottom of Form

**Hide Answer**Reveal Answer

*You have used 0 of 1 submissions*

PROBLEM 2-3

(1 point possible)

Pick the word below that best matches this item: exception

Top of Form

Iteration**Flow of control** Divide-and-conquer*O*(1)Attribute

Bottom of Form

**Hide Answer**Reveal Answer

*You have used 0 of 1 submissions*

PROBLEM 2-4

(1 point possible)

Pick the word below that best matches this item: self.title from the Problem Set 7 NewsStoryclass

Top of Form

IterationFlow of controlDivide-and-conquerTime complexity**Attribute**

Bottom of Form

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 iteration has value n, the smallest n+1 elements of the sorted version of lst are in L in the correct order.

Top of Form

**True**False

Bottom of Form

**Hide Answer**Reveal Answer

*You have used 0 of 1 submissions*

PROBLEM 3-1 B

(1 point possible)

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

Top of Form

True **False**

Bottom of Form

**Hide Answer**Reveal Answer

*You have used 0 of 1 submissions*

PROBLEM 3-1 C

(1 point possible)

When we reach the marked spot in the code, and the variable iteration has value n, the firstn+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.

Top of Form

True **False**

Bottom of Form

**Hide Answer**Reveal Answer

*You have used 0 of 1 submissions*

PROBLEM 3-1 D

(1 point possible)

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

Top of Form

**True** False

Bottom of Form

**Hide Answer**Reveal Answer

*You have used 0 of 1 submissions*

PROBLEM 3-1 E

(1 point possible)

The complexity of this algorithm is:

Top of Form

 O(n^2) Status: unsubmitted

Bottom of Form

**Hide Answer**Reveal Answer

*You have used 0 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

L = lst[:] # the next 3 questions assume this line just executed

return lst

PROBLEM 3-2 A

(1 point possible)

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

Top of Form

- unanswered

**True**False

Bottom of Form

**Show Answer**Reveal Answer

*You have used 0 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 largestn+1 elements of the sorted version of lst are in L in the correct order.

Top of Form

- unanswered

True**False**

Bottom of Form

**Show Answer**Reveal Answer

*You have used 0 of 1 submissions*

PROBLEM 3-2 C

(1 point possible)

When we reach the marked spot in the code, and the variable iteration has value n, the firstn+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.

Top of Form

- unanswered

True**False**

Bottom of Form

**Show Answer**Reveal Answer

*You have used 0 of 1 submissions*

PROBLEM 3-2 D

(1 point possible)

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

Top of Form

- unanswered

**True**False

Bottom of Form

**Show Answer**Reveal Answer

*You have used 0 of 1 submissions*

PROBLEM 3-2 E

(1 point possible)

The complexity of this algorithm is:

Top of Form

 - unanswered  **O(n^2)**

Bottom of Form

**Show Answer**Reveal Answer

*You have used 0 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

return out

PROBLEM 3-3 A

(1 point possible)

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

Top of Form

- unanswered

**True**False

Bottom of Form

**Show Answer**Reveal Answer

*You have used 0 of 1 submissions*

PROBLEM 3-3 B

(1 point possible)

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

Top of Form

- unanswered

True**False**

Bottom of Form

**Show Answer**Reveal Answer

*You have used 0 of 1 submissions*

PROBLEM 3-3 C

(1 point possible)

When we reach the marked spot in the code, and the variable iteration has value n, the firstn+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.

Top of Form

- unanswered

**True**False

Bottom of Form

**Show Answer**Reveal Answer

*You have used 0 of 1 submissions*

PROBLEM 3-3 D

(1 point possible)

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

Top of Form

- unanswered

True**False**

Bottom of Form

**Show Answer**Reveal Answer

*You have used 0 of 1 submissions*

PROBLEM 3-3 E

(1 point possible)

The complexity of this algorithm is:

Top of Form

 - unanswered **O(n^2)**

Bottom of Form

**Show Answer**Reveal Answer

*You have used 0 of 1 submissions*

PROBLEM 3-4

Answer the following 5 questions based on this code.

def sort4(lst):

def unite(l1, l2):

if len(l1) == 0:

return l2

elif len(l2) == 0:

return l1

elif l1[0] < l2[0]:

return [l1[0]] + unite(l1[1:], l2)

else:

return [l2[0]] + unite(l1, l2[1:])

if len(lst) == 0 or len(lst) == 1:

return lst

else:

front = sort4(lst[:len(lst)/2])

back = sort4(lst[len(lst)/2:])

L = lst[:] # the next 3 questions assume this line just executed

return unite(front, back)

PROBLEM 3-4 A

(1 point possible)

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

Top of Form

- unanswered

True**False**

Bottom of Form

**Show Answer**Reveal Answer

*You have used 0 of 1 submissions*

PROBLEM 3-4 B

(1 point possible)

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

Top of Form

- unanswered

True**False**

Bottom of Form

**Show Answer**Reveal Answer

*You have used 0 of 1 submissions*

PROBLEM 3-4 C

(1 point possible)

When we reach the marked spot in the code on the nth recursive call of sort4, the first n+1elements 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.

Top of Form

- unanswered

True**False**

Bottom of Form

**Show Answer**Reveal Answer

*You have used 0 of 1 submissions*

PROBLEM 3-4 D

(1 point possible)

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

Top of Form

- unanswered

True**False**

Bottom of Form

**Show Answer**Reveal Answer

*You have used 0 of 1 submissions*

PROBLEM 3-4 E

(1 point possible)

The complexity of this algorithm is:

Top of Form

 - O(n log n) Bottom of Form