**Section 15.2 Problem: Computing Factorials**

15.1 Which of the following statements are true?

A. Every recursive function must have a base case or a stopping condition.

B. Every recursive call reduces the original problem, bringing it increasingly closer to a base case until it becomes that case.

C. Infinite recursion can occur if recursion does not reduce the problem in a manner that allows it to eventually converge into the base case.

D. Every recursive function must have a return value.

E. A recursive function is invoked differently from a non-recursive function.

ABC

Answer analysis:ABC

15.2 Fill in the code to complete the following function for computing factorial.

def factorial(n):

if n == 0: # Base case

return 1

else:

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ # Recursive call

A. n \* (n - 1)

B. n

C. n \* factorial(n - 1)

D. factorial(n - 1) \* n

CD

Answer analysis:CD

15.3 What are the base cases in the following recursive function?

def xfunction(n):

if n > 0:

print(n % 10)

xfunction(n // 10)

A. n > 0

B. n <= 0

C. no base cases

D. n < 0

B

Answer analysis:B

15.4 Analyze the following recursive function.

def factorial(n):

return n \* factorial(n - 1)

A. Invoking factorial(0) returns 0.

B. Invoking factorial(1) returns 1.

C. Invoking factorial(2) returns 2.

D. Invoking factorial(3) returns 6.

E. The function runs infinitely and causes a StackOverflowError.

E

Answer analysis:E

15.5 How many times is the factorial function in Listing 15.1 invoked for factorial(5)?

A. 3

B. 4

C. 5

D. 6

D

Answer analysis:D

**Section 15.3 Problem: Computing Fibonacci Numbers**

15.6 Which of the following statements are true?

A. The Fibonacci series begins with 0 and 1, and each subsequent number is the sum of the preceding two numbers in the series.

B. The Fibonacci series begins with 1 and 1, and each subsequent number is the sum of the preceding two numbers in the series.

C. The Fibonacci series begins with 1 and 2, and each subsequent number is the sum of the preceding two numbers in the series.

D. The Fibonacci series begins with 2 and 3, and each subsequent number is the sum of the preceding two numbers in the series.

A

Answer analysis:A

15.7 How many times is the fib function in Listing 15.2 invoked for fib(5)?

A. 14

B. 15

C. 25

D. 31

E. 32

B

Answer analysis:B

15.8 Fill in the code to complete the following function for computing a Fibonacci number.

def fib(index):

if index == 0: # Base case

return 0

elif index == 1: # Base case

return 1

else: # Reduction and recursive calls

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

A. fib(index - 1)

B. fib(index - 2)

C. fib(index - 1) + fib(index - 2)

D. fib(index - 2) + fib(index - 1)

CD

Answer analysis:CD

**Section 15.4 Problem Solving Using Recursion**

15.9 In the following function, what is the base case?

def xfunction(n):

if n == 1:

return 1

else

return n + xfunction(n - 1)

A. n is 1.

B. n is greater than 1.

C. n is less than 1.

D. no base case.

A

Answer analysis:A

15.10 What is the return value for xfunction(4) after calling the following function?

def xfunction(n):

if n == 1:

return 1;

else:

return n + xfunction(n - 1)

A. 12

B. 11

C. 10

D. 9

C

Answer analysis:C

15.11 Fill in the code to complete the following function for checking whether a string is a palindrome.

def isPalindrome(s):

if len(s) <= 1: # Base case

return True

elif \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

return False

else:

return isPalindrome(s.substring(1, len(s) - 1))

A. s[0] != s[-1]: # Base case

B. s[0] != s[len(s)]: # Base case

C. s[1] != s[len(s) - 1]: # Base case

D. s[1] != s[len(s)]: # Base case

A

Answer analysis:A

15.12 Analyze the following code:

def xfunction(x, length):

print(x[length - 1], end = " ")

xfunction(x, length - 1)

x = [1, 2, 3, 4, 5]

xfunction(x, 5)

A. The program displays 1 2 3 4 6.

B. The program displays 1 2 3 4 5 and then raises an index out of range exception.

C. The program displays 5 4 3 2 1.

D. The program displays 5 4 3 2 1 and then raises an index out of range exception.

D

Answer analysis:D

**Section 15.5 Recursive Helper functions**

15.13 Fill in the code to complete the following function for checking whether a string is a palindrome.

def isPalindrome(s):

return isPalindromeHelper(s, 0, len(s) - 1)

def isPalindromeHelper(s, low, high):

if high <= low: # Base case

return True

elif s[low] != s[high]: # Base case

return False

else:

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

A. isPalindromeHelper(s)

B. isPalindromeHelper(s, low, high)

C. isPalindromeHelper(s, low + 1, high)

D. isPalindromeHelper(s, low, high - 1)

E. isPalindromeHelper(s, low + 1, high - 1)

E

Answer analysis:E

15.14 Fill in the code to complete the following function for sorting a list.

def sort(lst):

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ # Sort the entire list

def sortHelper(lst, low, high):

if low < high:

# Find the smallest number and its index in lst[low .. high]

indexOfMin = low

min = lst[low]

for i in range(low + 1, high + 1):

if lst[i] < min:

min = lst[i]

indexOfMin = i

# Swap the smallest in list(low .. high) with list(low)

lst[indexOfMin] = lst[low]

lst[low] = min

# Sort the remaining list(low+1 .. high)

sortHelper(lst, low + 1, high)

A. sortHelper(lst)

B. sortHelper(lst, len(lst) - 1)

C. sortHelper(lst, 0, len(lst) - 1)

D. sortHelper(lst, 0, len(lst) - 2)

C

Answer analysis:C

15.15 Fill in the code to complete the following function for binary search.

def recursiveBinarySearch(lst, key):

low = 0

high = len(lst) - 1

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def recursiveBinarySearchHelper(lst, key, low, high):

if low > high: # The list has been exhausted without a match

return ?low - 1

mid = (low + high) // 2

if key < lst[mid]:

return recursiveBinarySearchHelper(lst, key, low, mid - 1)

elif key == lst[mid]:

return mid

else:

return recursiveBinarySearchHelper(lst, key, mid + 1, high)

A. recursiveBinarySearchHelper(lst, key)

B. recursiveBinarySearchHelper(lst, key, low + 1, high - 1)

C. recursiveBinarySearchHelper(lst, key, low - 1, high + 1)

D. recursiveBinarySearchHelper(lst, key, low, high)

D

Answer analysis:D

15.16 What will displayed by the following code?

def main():

times = count("abcabc", 'a')

print(ch + " appears " + str(times) + (" times " if times > 1 else " time ") + "in " + s)

def count(s, a):

return countHelper(s, a, len(s) - 1)

def countHelper(s, a, high):

result = 0;

if high > 0:

result = countHelper(s, a, high - 1) + (1 if s[high] == a else 0)

return result;

main()

A. a appears 1 times in abcdabc

B. a appears 2 times in abcdabc

C. a appears 1 time in abcdabc

D. a appears 2 time in abcdabc

C

Answer analysis:C

**Section 15.7 Tower of Hanoi**

15.17 How many times is the recursive moveDisks function invoked for 3 disks?

A. 3

B. 7

C. 10

D. 14

B

Answer analysis:B

15.18 How many times is the recursive moveDisks function invoked for 4 disks?

A. 5

B. 10

C. 15

D. 20

C

Answer analysis:C

15.19 Analyze the following two programs:

A:

def xfunction(length):

if length > 1:

print(length - 1, end = " ")

xfunction(length - 1)

xfunction(5)

B:

def xfunction(length):

while length > 1:

print(length - 1, end = " ")

xfunction(length - 1)

xfunction(5)

A. The two programs produce the same output 5 4 3 2 1.

B. The two programs produce the same output 1 2 3 4 5.

C. The two programs produce the same output 4 3 2 1.

D. The two programs produce the same output 1 2 3 4.

E. Program A produces the output 4 3 2 1 and Program B runs infinitely.

E

Answer analysis:E

**Section 15.10 Recursion versus Iteration**

15.20 Which of the following statements are true?

A. Recursive functions run faster than non-recursive functions.

B. Recursive functions usually take more memory space than non-recursive functions.

C. A recursive function can always be replaced by a non-recursive function.

D. In some cases, however, using recursion enables you to give a natural, straightforward, simple solution to a program that would otherwise be difficult to solve.

BCD

Answer analysis:BCD

**Section 15.11 Tail Recursion**

15.21 Analyze the following functions;

def f1(n):

if n == 0:

return 0

else:

return n + f1(n - 1)

def f2(n, result):

if n == 0:

return result

else:

return f2(n - 1, n + result)

print(f1(3))

print(f2(3, 0))

A. f1 is tail recursion, but f2 is not

B. f2 is tail recursion, but f1 is not

C. f1 and f2 are both tail recursive

D. Neither f1 nor f2 is tail recursive

B

Answer analysis:B

15.22 Show the output of the following code:

def f2(n, result):

if n == 0:

return 0

else:

return f2(n - 1, n + result)

print(f2(2, 0))

A. 0

B. 1

C. 2

D. 3

A

Answer analysis:A