Describe the run-time bounds for last weeks tasks using bigO notation

Task 1 – My Code

from random import choice #import choice function only (1)

def listCreation(x): #function for list creation (1)

myList = [] (1)

i=0 (1)

while i < listQuantity: (N)

listItem = int(input("Enter list item ")) (N)

myList.append(listItem) (N)

i=i+1 (N)

print("Your selected list: " + str(myList)) (1)

Shuffler(myList) #call to shuffler function (1)

def Shuffler(theList): #function to shuffle created list (1)

list\_length = len(theList) (1)

newList = [] (1)

for i in range(list\_length): (N)

element = choice(theList) (N)

newList.append(element) (N)

theList.remove(element) (N)

print(newList) (1)

listQuantity = int(input("Enter the number of elements in the list ")) (1)

listCreation(listQuantity) #Call to listCreation function (1)

From my code above, we can see that some lines of code run once, while the loops run N number of times in the worst case scenario. Altogether the code has run time of 8N + 12. When we remove the constants and the multiplier we would write the run time as O(n), in bigO notation.

Task 2 – My Code

#function to calculate factorial of a number

def factorial(myNumber): (1)

try: #check datatype of input (1)

myNumber = int(myNumber) (1)

except ValueError: (1)

print("Incorrect datatype") (1)

return (1)

myFactorial = 1 (1)

while myNumber >= 1: (N)

myFactorial = myFactorial \* myNumber (N)

myNumber = myNumber – 1 (N)

print("Your factorial is " + str(myFactorial)) (1)

trailing(myFactorial) (1)

#function to count trailing 0's in factorial number

def trailing(factorialResult): (1)

count = 0 (1)

index = -1 (1)

factorialResult = str(factorialResult) (1)

for element in factorialResult: (N)

number = factorialResult[index] (N)

if int(number) == 0: #if the last no. is 0, increment count (N)

count = count + 1 (N)

index = index -1 (N)

else: (N)

break (N)

print ("Number of trailing 0's is " + str(count)) (1)

originalNumber = input("Enter a number ") #input number (1)

num = factorial(originalNumber) #call to the factorial function with number (1)

For the code above, again, we can see that some of the lines of code execute only once while the loops execute N times in the worst case. Altogether the run time for this program is 10N + 16. Once the constant and the multiplier is removed, we find that the bigO representation of the run time is O(n).