* Note: in future, state assumptions

Ainsley McLaughlin

9/9/24

CS 5012: Foundations of Computer Science

Asymptotic Complexity Exercise

Given the following code snippets, provide the worst case time complexity in the form of Big-O notation. Justify your response and state any assumptions made. Treat these functions as constant runtime: print(), append()

```
def measure(inputList):
    int n = len(inputList)
    int sum = 0;
    for i in range(0, n):
       for j in range(0, 5):
          sum+= j * inputList[i]
       for k in range(0, n):
          sum -= inputList[k]
```

int n = len(inputList) O(n)
int sum = 0; O(1)
for i in range(0, n): O(n)
for j in range(0, 5): O(5)O(n)=O(1)*n=O(n)
sum+= j * inputList[i]
for k in range(0, n): n*O(n)=O(n^2)
sum -= inputList[k]

O(5)·n = O(1)·n = O(n)

T(n)=O(n)+O(1)+O(n)+O(n)+O(n^2)

The asymptotic complexity of this algorithm is: O (n^2)

```
def addElement(ele):
    myList =[]
    myList.append(666)
    print myList
```

myList =[] O(1)
myList.append(666) O(1)
print myList O(1)

since print and append are constant run times=O(1)

The asymptotic complexity of this algorithm is: O (1)

T(n)=O(1)+O(1)+O(1)=O(1)

```
► num = 10

  def addOnesToTestList(num):
    testList = []
    for i in range(0,num):
        testList.append(1)
        print(testList)

    return testList
```

num = 10 O(1)

testList = [] O(1)

10 → O(num) ← assume constant

for i in range(0,num): O(10)=O(1)

testList.append(1) (10)O(1)=O(10=O(1))

print(testList) 10*O(1)=O(10)=O(1)

return testList O(1) return statements are O(1)

The asymptotic complexity of this algorithm is: O (1)

T(n)= O(1)+O(1)+O(1)+O(1)+O(1)+O(1)+O(1)+O(1) =O(1)

behaves as n

```
► testList = [1, 43, 31, 21, 6, 96, 48, 13, 25, 5]
```

imagine not fixed, will be modified

O(1)

```
  def someMethod(testList):
    for i in range(len(testList)):
        for j in range(i+1, len(testList)):
           if testList[j] < testList[i]:
             testList[j], testList[i] = testList[i], testList[j]
           print(testList)
```

for i in range(len(testList)): O(n)

for j in range(i+1, len(testList)): O(n)* O((n-i-1)/2=O(n^2)

if testList[j] < testList[i]: O(1)*n^2=O(n^2)

testList[j], testList[i] = testList[i], testList[j] O(1)*O(n^2) =O(n^2)

print(testList) O(1)*O(n^2)=O(n^2)

The asymptotic complexity of this algorithm is: O (n^2)

T(n)= O(1)+O(n)+O(n^2)+O(n^2)+O(n^2)+O(n^2) = O(n^2)

```
► def searchTarget(target_word):
  # Assume range variables are unrelated to size of aList

    for (i in range1):
        for (j in range2):
            for (k in range3):
                if (aList[k] == target_word):
                    return 1
        return -1
    return -1
```

for (i in range1): O(1)

for (j in range2): O(1)

for (k in range3): O(1)

if (aList[k] == target_word): O(1)

return 1 O(1)

return -1 O(1)

return -1 O(1)

The asymptotic complexity of this algorithm is: O (1)

T(n) = O(1)+O(1)+O(1)+O(1)+O(1)+O(1)+O(1) = O(1)

```
def someSearch(sortedList, target):
    left = 0
    right = len(sortedList)- 1

    while (left <= right):
        mid = (left + right)/2
        if (sortedList(mid)==target):
            return mid
        elif(sortedList(mid) < target):
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

O(1) O(1) $O(\log_2(n))$ $O(\log_2(n))$ $O(1)\cdot\log_2(n) = O(\log_2(n))$ $O(1)\cdot\log_2(n) = O(\log_2(n))$ $O(1)\cdot\log_2(n) = O(\log_2(n))$ $O(1)\cdot\log_2(n) = O(\log_2(n))$ $O(1)\cdot\log_2(n) = O(\log_2(n))$

The asymptotic complexity of this algorithm is: O ($\log_2(n)$)

$T(n) = O(1) + O(1) + O(\log_2(n)) + O(\log_2(n)) + O(\log_2(n)) \cdots O(\log_2(n)) = O(\log_2(n))$

```
#Assume data is a list of size n
total = 0
for j in range(n):
    total += data[j]
big = data[0]
for k in range(1,n):
    big = max(big,
data[k])
```

O(1) O(n) $O(1)\cdot n = O(n)$ O(1) O(n) $O(1)\cdot n = O(n)$

The asymptotic complexity of this algorithm is: O (n)

$T(n) = O(1) + O(n) + O(n) + O(1) + O(n) + O(n) = O(n)$

```
powers = 0
k = 1
while k < n:
    k = 2*k
    powers += 1
```

O(1) O(1) $O(\log_2(n))$ $O(1)\cdot\log_2(n) = O(\log_2(n))$ $O(1)\cdot\log_2(n) = O(\log_2(n))$

The asymptotic complexity of this algorithm is: O ($\log_2(n)$)

$T(n) = O(1) + O(1) + O(\log_2 n) + O(\log_2(n)) + O(\log_2 n) = O(\log_2(n))$

```
k = 1
while k < n:
    for j in range(k):
        steps += 1
    k = 2*k
```

O(1) $O(\log_2(n))$ $O(n)\cdot\log_2(n) = O(n\log_2(n))$ $O(1)\cdot n\log_2(n) = O(n\log_2(n))$ $O(1)\cdot\log_2(n) = O(\log_2(n))$

this is incorrect, this is n since k does not reach to log n times

The asymptotic complexity of this algorithm is: O ($n \log_2(n)$)

$T(n) = O(1) + O(\log_2(n)) + O(n \log n) + O(n \log n) + O(\log_2(n)) = O(n \log_2(n))$

```
for k in range(1,n):
    j = 1
    while j < k:
        total += 1
        j = 2 * j
```

$O(n)$

$O(1) \cdot n = O(n)$

$n \cdot O(\log_2(n)) = O(n \log_2(n))$

$O(1) \cdot n \log_2(n) = O(n \log_2(n))$

$O(1) \cdot n \log_2(n) = O(n \log_2(n))$

The asymptotic complexity of this algorithm is: O ($n \log_2(n)$)

$T(n) = O(n) + O(n) + O(n \log_2 n) + O(n \log_2 n) + O(n \log_2(n))$