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]
```

O(1) --> Assignments are constant
O(1)
O(n) --> b/c the for loop runs n times
n * O(1) = O(n) --> b/c the loop runs 5 times exactly
n * O(1) *O(1) = O(n)
n * O(n) = O(n^2) --> b/c it's a nested for loop that runs n times
n * n * O(1) = O(n^2)

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

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

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

O(1) --> Instantiating an empty list
O(1) --> constant run time
O(1) --> constant run time

The asymptotic complexity of this algorithm is: O (____1____)

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

Assume that num is a fixed!!!!

```
num = 10
```
O(1)

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

    return testList
```
O(1) --> Instantiating an empty list

O(num) = O(10) = O(1)

O(1) --> constant run time

O(1) --> constant run time

O(1) --> constant run time

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

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

Assume that num is NOT fixed!!!

```
testList = [1, 43, 31, 21, 6, 96, 48, 13, 25, 5]
```
O(1) --> instantiating a list

```
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)
```
O(n) --> b/c the for loop runs n times

n*O(n) = O(n^2)

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

n*n*O(1)*O(1)

n*n*O(1)

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)

```
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
```
O(1)

O(1) * O(1) = O(1)

O(1)*O(1)*O(1) = O(1)

O(1)*O(1)*O(1)*O(1) = O(1)

O(1)

O(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)

```
def someSearch(sortedList, target):
    left = 0                                          O(1)
    right = len(sortedList)- 1                        O(1) --> Assignments are constant

    while (left <= right):                            O(log n) --> b/c each iteration divides by 2
        mid = (left + right)/2                        log n * O(1) = O(log n)
        if (sortedList(mid)==target):                 log n * O(1) = O(log n)
            return mid                                log n * O(1) = O(log n)
        elif(sortedList(mid) < target):               log n * O(1) = O(log n)
            left = mid + 1                            log n * O(1) = O(log n)
        else:
            right = mid - 1                           log n * O(1) = O(log n)
    return -1                                         O(1)
```

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

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

```
#Assume data is a list of size n
total = 0                       O(1) --> Assignments are constant
for j in range(n):              O(n) --> b/c the loop runs n times
    total += data[j]            n * O(1) = O(n)
big = data[0]                   O(1)
for k in range(1,n):            O(n)
    big = max(big,              n * O(1) = O(n)
data[k])
```

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

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

```
powers = 0          O(1) --> Assignments are constant
k = 1               O(1)
while k < n:        O(log n)
    k = 2*k         log n * O(1)
    powers += 1     log n * O(1)
```

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

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

```
k = 1                           O(1)
while k < n:                    O(log n)
    for j in range(k):          log n * O(n) = O(n) --> b/c linear time has a higher complexity than log time
        steps += 1              n * log n * O(1) = O(n)
    k = 2*k                     log n * O(1) = O(log n)
```

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

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

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

O(n)
n* O(1)
n * O(log n) = O(n) --> b/c linear time has a higher complexity than log time
n * log n * O(1) = O(n)
n* log n * O(1) = O(n)

The asymptotic complexity of this algorithm is: O (n) <-- Should be O(nlogn)

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