Ainsley McLaughlin

* In future, state assumptions

UVA DATA SCIENCE

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]
```

$O(1)$, $O(1)$, $O(n)$, $O(1)$, $O(1)$, $O(n)$, $O(1)$

$n \cdot O(5) = n \cdot O(1) = O(n)$

from nested for loop

$n \cdot O(n) = O(n^2)$

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

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

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

$O(1)$, $O(1)$, $O(1)$

*print and append are $O(1)$ (constant run time)

The asymptotic complexity of this algorithm is: O (____1______)

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

```
► num = 10   O(1)

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

    return testList  O(1)                → returns are O(1)
```

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

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

```
► testList = [1, 43, 31, 21, 6, 96, 48, 13, 25, 5]  O(1)   ↙consider
                                                          list changeable
                                                          (not fixed, will be
                                                           modified)
  def someMethod(testList):
    for i in range(len(testList)):  O(n)
        for j in range(i+1, len(testList)):  O(n-(i+1)) = O(n)  → n·O(n) = O(n^2)
           if testList[j] < testList[i]:  O(1)        → O(1)·n^2 = O(n^2)
             testList[j], testList[i] = testList[i], testList[j]  O(1)
           print(testList)  O(1)
```

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

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

* *

```
► def searchTarget(target_word):
  # Assume range variables are unrelated to size of aList
                        ↙constant
    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          O(1)
     right = len(sortedList)- 1      O(1)

     while (left <= right):     O(log n)
        mid = (left + right)/2      O(1)
        if (sortedList(mid)==target):   O(1)
            return mid      O(1)
        elif(sortedList(mid) < target):   O(1)
            left = mid + 1    O(1)
        else:

             right = mid - 1      O(1)

     return -1     O(1)
```

→ $O(1) \cdot \log n = O(\log n)$

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

$T(n) = O(1) + O(1) + O(\log n) + \underline{O(\log n)} + O(\log n) + O(\log n) + \ldots + O(\log n) = O(\log n)$

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

→ * $O(1) \cdot n = O(n)$

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

$T(n) = O(1) + O(n) + \underset{*}{\underline{O(n)}} + O(1) + O(n) + O(1) = O(n)$

```
►   powers = 0    O(1)
    k = 1     O(1)
    while k < n:    O(log n)
        k = 2*k     O(1)
        powers += 1    O(1)
```

→ (*) $O(1) \cdot \log n = O(\log(n))$

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

$T(n) = O(1) + O(1) + O(\log n) + \underset{(*)}{\underline{O(\log n)}} + O(\log n)$

*Don't understand this one

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

→ not $O(\log n)$ since K does not get to that limit

→ $O(n) \cdot O(n) = O(n^2)$

→ $O(1) \cdot n^2 = O(n^2)$

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

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

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

O(n)
O(1)
O(logn) *
O(1) ✩
O(1)

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

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

$* \; n \cdot O(\log n)$

$✩ \; O(1) \cdot n\log n = O(n\log n)$

For second to last :

while $\rightarrow$ $O(n)$

for $\rightarrow \sum_{i=1}^{\log(n)} O(k) = O\left(2^{\log_2(n)}\right) = O(n)$