Practice Questions on Complexity (Week 3)

Tutorial Questions

1. The table below shows the number of times an operation needs to be performed for the corresponding value of **n** (the input size) of a particular algorithm. What is the Big O complexity of the algorithm?

n	Number of times
1	10
2	11
3	12
4	13
5	14
6	15
7	16
…	
20	29
21	30

n	Number of times
22	31
23	32
…	
500	509
501	510
502	511
…	
1000	1009
1001	1010
1002	1011

O(n)

2. The following expressions provide the number of steps taken by an algorithm to solve a problem of size **n**. For each expression, specify the Big O complexity.

	Number of steps	Big O	
a	$9 + 0.02N^2 + 0.1N$	O(N^2)	
b	$N^2 + 2N^{-3}$	O(N^2)	
c	$N! + 100N^{20}$	O(N!)	
d	$2^N + N!$	O(2^N)	O(N!)
e	$5N(\log_2 N) + N \times \text{sqrt}(N)$	O(N log N)	O(N^3/2)
f	$N^2(\log_2 N) + N(\log_2 N)^2$	O(N^2 log N)	
g	$10N^2\log(N) + 5N^3 + N^{\log(N)}$	O(N^2 log N)	O(N^logN)
h	$10^5 + 10^4(\log(N))^2 + 10^3\log(N^2)$	O(log N^2)	O((log N)^2)

3. Determine the Big O complexity of the following Python functions.

a. Two loops in a row:

```
def func1(n, m):
  for i in range(n):
    # sequence of statements
  for j in range(m):
    # sequence of statements
  return
```

O(n+m) = O(n)

b. A nested loop followed by a non-nested loop O(n^2)

```
def func2(n):
  for i in range(n):
    for j in range(n):
      # sequence of statements
  for k in range(n):
    # sequence of statements
  return
```

c. A nested loop in which the number of times the inner loop executes depends on the value of the outer loop index O(n!)

```
def func3(n):
  for i in range(n):
    j = n
    while j > i:
      # sequence of statements
      j -= 1
  return
```

O(n^2)

4. Provide the Big O complexity for each of the three functions below.

a.
```
def f(n):
  sum = 0
  for i in range(1, n):
     sum = sum + i
  return sum
```
O(n)

b.
```
def g(n):
  sum = 0
  for i in range(1, n):
    sum = sum + i + f(i)
  return sum
```
O(n^2)

c.
```
def h(n):
  return f(n) + g(n)
```
O(n^2)

5. Given the function below:

```
def func_a(n):
  for i in range(n):
    for j in range(n):
      f(j)
```

a. If the execution time of function **f(j)** is a constant **T** (independent of its parameter **j**), what is the complexity of **func_a**? O(n^2)

b. If the execution time of function **f(j)** is linear in its parameter **j** (for simplicity, assume that the execution time of **f(j)** is simply **j**), what is the complexity of **func_a**? (You should have both the count of time units and the corresponding Big O notation)

O(n^3)

6. Given that **sub_func** is a function that takes an integer **n** as input, and has the complexity of O(n).

   a. What is the Big O complexity of **func_6a**?

Hint: $\sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + (n-1) + n = \frac{n(n+1)}{2}$

Remember to add cost of sub_func

```
def func_6a(n):
  for i in range(n):
    for j in range(i):
      sub_func(n)
  return
```

O(n^2)

O(N^3)

   b. What is the Big O complexity of **func_6b**?

Hint: $\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + 3^2 + \cdots + (n-1)^2 + n^2 = \frac{n(n+1)(2n+1)}{6}$

```
def func_6b(n):
  for i in range(1, n+1):
    for j in range(1, i*i + 1):
      sub_func(n)
  return
```

O(n^3)

O(n^4)

   c. What is the Big O complexity of **func_2c**?

Hint: $\sum_{i=1}^{n} 1/i = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n-1} + \frac{1}{n} \approx O(\log n)$

```
def func_2c(n):
  for i in range(1, n+1):
    for j in range(1, n+1):
      if j % i == 0:
        for k in range(1, n+1):
          sub_func(n)
  return
```

O(n log n)

O(n^3 log n)

Extra Practice Questions

7. The table below shows the number of times an operation needs to be performed for the corresponding value of **n** (the input size) of a particular algorithm. What is the Big O complexity of this algorithm?

n	Number of times
1	1
2	2
3	3
4	4
5	5
6	6
…	
20	50
21	50
22	50

n	Number of times
23	50
…	
500	50
501	50
502	50
…	
1000	50
1001	50
1002	50
Other numbers >1002	50

O(log n) or O(1)

8. The following expressions provide the number of steps taken by an algorithm to solve a problem of size **n**. For each expression, specify the simplified Big O complexity.

	Number of steps	Big O	
a	$2N^2 + 2N^3 + 3N^4$	O(N^4)	
b	$N^2 \times 2N^{-3}$	O(N^2)	O(n^-1)
c	$N! \times 100N^{20}$	O(N!N^20)	
d	$5 \times (2N)!$	O(10N!)	O((2N!))
e	$N(\log_2 N) + N(\log_3 N) + N(\log_4 N)$	O(N log N)	
f	$N(\log_2 (2N))$	O(N log N)	
g	$1000 + 5\log(N) + 2N + N^2 + 2^{2N}$	O(2^2N)	
h	$\log(N^5) + \log_5(N) + 5N$	O(N)	

9. Determine the Big O complexity of the following Python functions.

a.
```
def q1(n):
  for j in range(n):
    f(j) # f takes constant time
  return
```
O(n)

b.
```
def q2(n):
  for j in range(n):
    g(j) # g takes time linear in the value of its parameter
  return
```
O(n^2)

c.
```
def q3(n, k):
  for j in range(n):
    g(k) # g takes time linear in the value of its parameter
  return
```
O(nk)

10. Provide the complexity of **my_func** for the following algorithms.

   a. Given that **sub_func** is a function that takes an integer `n` as input and has a complexity of O(`n`), what is the Big O complexity of **my_func**?

```
def my_func(n):
  for i in range(n):
    if i%2==0:
      sub_func(n)
  return
```

O(n log n)

function calls n/2
times: n*n/2 = (n^2)/2
= O(n^2)

   b. Given that **sub_func** is a function that takes an integer `n` as input and has a complexity of O(`log n`), what is the Big O complexity of **my_func**?

```
def my_func(n):
  i = 2
  while i < n:
    i += 1
    sub_func(n)
  return
```

O(n log n)

   c. Given that **sub_func** is a method that takes an integer `n` as input and has a complexity of O(`n`), what is the Big O complexity of **my_func**?

```
def my_func(n):
  for i in range(n):
    sub_func(i)
  return
```

O(n^2)

11. You work for a company that manufactures cups. To start cup manufacturing you need to set four switches [switch1, switch2, switch3, switch4] to a certain position. Every switch only has two possible positions, "ON" or "OFF". (For example, this is one of the valid positions: ["ON", "OFF", "ON", "OFF"]). Unfortunately, the lead engineer is on leave, and only he knows the right combination of switches to get the machines started.

    Suppose you have a function **start(combination_array)** that returns **True** if the specified input switch positions can get the machines started, and **False** otherwise.

   a. Provide an algorithm in pseudocode that prints out the right combination of switch positions.
   b. If only a call to the **start** function is considered as one operation, what is the worst-case complexity of an algorithm that takes as input the number of switches N, and prints out the right combination of N switches? O(N^2)
   c. After trying some positions of switches, you realize that two (or more) consecutive "ON", or two (or more) consecutive "OFF" would cause the system to overheat. For example, the following switch positions would cause overheating:

      - ["ON", "ON", "ON",..., "OFF"]
      - ["ON", "OFF", "OFF",..., "ON"]

      O(N)

      What is the complexity of an algorithm that calls the **start** function only for those combinations of N switches that would not cause overheating? Note that only a call to the **start** function is considered as one operation.

12. An array **a** has **n** distinct numbers (i.e. there are no duplicate numbers). The following algorithm finds the **k**th smallest number of an array (1≤k≤n). For example, given array **a**=[14,12,26,19] and **k**=3, the algorithm returns the 3rd smallest number which is 19.

```
01  def find_minimum(a,k):
02    min = a[0]
03    max = a[0]
04
05    # part (a)
06    for j in range(1, len(a)):
07      if min > a[j]:
08        min = a[j] # find minimum number
09      if max < a[j]:
10        max = a[j] # find maximum number
11
12    # part (b)
13    if k <= 1:
14      return min
15    if k >= len(a):
16      return max
17
18    # part (c)
19    for i in range(2, k+1):  # find ith minimum no. of a
20      ith_min = max
21
22      # part (c1)
23      for j in range(len(a)):
24        if a[j] < ith_min and a[j] > min:
25          ith_min = a[j]
26
27      min = ith_min
28      print(i)           # print the value of i
29      print(ith_min)     # print the value of ith_min
30      print("---")
31
32    return ith_min
```

a. What do lines 20-25 do? (i.e. what is the value of **ith_min** when the program reaches line 26?) Hint:
Trace through these lines assuming that **a** = [1, 2, 3, 4, 5], **max** = 5 and **min** = 1.
Trace through these lines again for the same **a** and **max**, but when **min** = 3.

b. Given array **a**=[3, 5, 1, 10, 7, 22, 15, 19, 6, 8, 16] and **k**=6, what is the value of **ith_min** that will be printed by **line 29** when the value of **i** printed by **line 28** is 4?

c. What is the worst-case complexity of the **find_minimum** algorithm?

d. Describe a more efficient algorithm to find the kth smallest number of an array of **n** distinct numbers. Show that your proposed algorithm will have a lower (better) worst-case complexity than the algorithm **find_minimum**.

~End