Big O Notation

Practice Questions

Question 1

Complete the table below to indicate the notation used to describe each of the following BigO notations:

Notation	Name
$O(1)$	Constant
$O(2^n)$	Exponential
$O(n)$	Linear
$O(\log n)$	Logarithmic
$O(n^2)$	Quadratic (or polynomial)

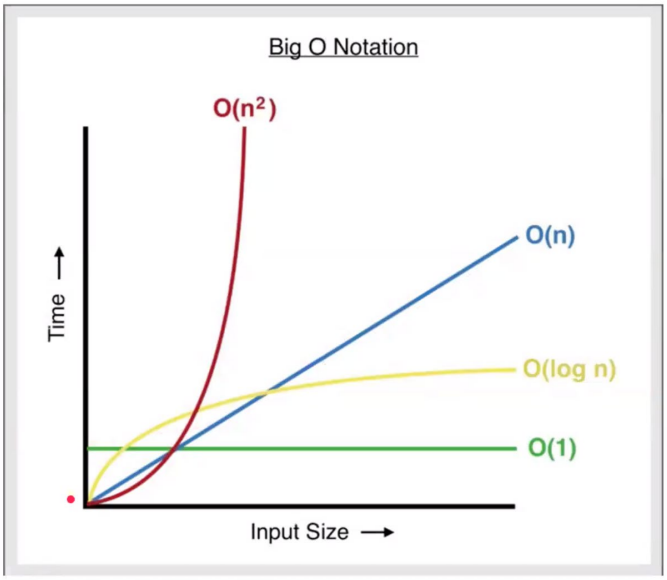
Question 2:

Put the list from above in order from least complex/efficient to most complex/efficient.

Notation	Name
$O(1)$	**Constant**
$O(\log n)$	**Logarithmic**
$O(n)$	**Linear**
$O(n^2)$	**Quadratic (or polynomial)**
$O(2^n)$	**Exponential**

Question 3:

Label the graph below with a key to indicate what each coloured line represents



Question 4:

Which case scenario does Big O use:

a. Best case
b. Worst case
c. Average case
d. All cases must be considered

Question 5:

What is the complexity of the following functions:

a. Linear

```
def example_function(list1, letter):
    count = 0
    for i in list1:
        if i[0]== letter:
            count += 1
    return count
```

b. Constant

main.py > example_function

```
def example_function(list1):
    if list1 and len(list1) > 0:
        return list1[-1]
    else:
        return None
```

c. Constant

```
def example_function2(x, y):
    return x + y
```

d. Linear

```
def example_function(list1):
    count = 0
    for i in list1:
        if i < 5:
            count += 1
    return count
```

e. **Logarithmic**

```
def binarySearch(listData, value)
  low = 0
  high = len(listData) - 1
  while (low <= high)
    mid = (low + high) / 2
    if (listData[mid] == value):
      return mid
    elif (listData[mid] < value)
      low = mid + 1
    else:
      high = mid - 1
  return -1
```

f. **Quadratic (or polynomial)**

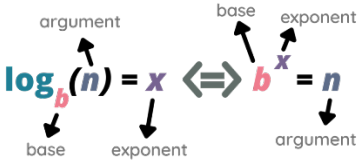
```
def prime_numbers(lower_number, higher_number):
    for num in range(lower_number, higher_number + 1):
        if num > 1:
            for i in range(2, num):
                if (num % i) == 0:
                    break
            else:
                print(num)
```

g. **Exponential**

```
def example_function(n):
    if n < 0:
        print("Incorrect input")
    elif n == 0:
        return 0
    elif n == 1 or n == 2:
        return 1
    else:
        return example_function(n-1) + example_function(n-2)
```

Question 6:

Rewrite the following logarithms so that they are expressed as a quadradic:



$\log_3 81 = 4 \quad \mathbf{3^4 = 81}$
$\log_2 32 = 5 \; \mathbf{2^5 = 32}$
$\log_2 128 = 7 \; \mathbf{2^7 = 128}$
$\log_3 243 = 5 \; \mathbf{3^5 = 243}$

Question 6

Calculate the following logarithms:

$\log_2 8$= **3**
$\log_2 64$= **6**
$\log_2 128$=**7**
$\log_2 1$= **0**
$\log_2 32$= **5**
$\log_3 125$= **4.395 (I ment to set this as $\log_5 125$)=3**

Question 7

Given an array of n, how long will a binary search take to find a specific value assuming the data is sorted. In the following cases:

a) N= 15 **(4 times ($\log_2 50$) is 3.9)**
b) N= 25 **(5 times ($\log_2 50$) is 4.64)**
c) N= 50 **(6 times ($\log_2 50$) is 5.64)**
d) N=100 **(7 times ($\log_2 100$) is 6.64)**
e) N= 1000 **(10 times ($\log_2 1000$) is 9.97)**

Question 8

The code snippet below relates to the questions which follow:

```
def example_funciton(list1, list2):

    matching = []
    for i in list1:
        for j in list2:
            if i == j:
                matching.append(i)
    return matching

list1 = [19, 2, 3, 4, 7, 18]
list2= [4, 16, 2, 3, 15, 1]
print(example_funciton(list1, list2))
```

a) What is the complexity of the function? **Quadratic (or polynomial)**
b) How many times does the outer loop of the function run? **6**
c) How many times does the inner loops of the function run on each iteration in the worst case scenario? **6**
d) In the worst-case scenario how many steps will this take? **36**
e) What could the complexity of the function be reduced to if the items in list2 were sorted in order from smallest to largest? **N log n**