# Worksheet 2: Big-O notation Answers

**Task 1**

1. (a) Complete the following table to show that as n becomes large, in a function

f(n) = 5n2 + 10n + 2

only the n2 term has a significant effect on the size of f(n)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **n** | **n2** | **5n2** | **10n** | **2** | **f(n)= 5n2+ 10n + 2** |
| **10** | 100 | 500 | 100 | 2 | 602 |
| **100** | 10,000 | 50,000 | 1,000 | 2 | 51,002 |
| **1000** | 1,000,000 | 5,000,000 | 10,000 | 2 | 5,010,002 |
| **10,000** | 100,000,000 | 500,000,000 | 100,000 | 2 | 500,100,002 |

(b) What is the time complexity of an algorithm that has 5n2 + 10n + 2 steps, expressed in the Big-O notation?

O(n2)

2. Calculate the number of assignment statements in each of the following pseudocode fragments. Hence calculate the Big-O time complexity of each algorithm.

(a) a = 1000

FOR i = 1 TO n

x = a + i

ENDFOR

Number of assignment statements = 1 + n Big-O = O(n)

(b) FOR i = 0 TO n

FOR j = 0 TO n

k = n \* 2 + i \* j

ENDFOR

ENDFOR

Number of assignment statements = (n + 1)(n + 1) = (n + 1)2 = n2+ 2n + 1. The term in n2 dominates so Big-O = O(n2)

(c) FOR i = 1 TO n

x = 5 + i \* i

ENDFOR

FOR j = 1 TO n

y = 10 – j

ENDFOR

There are 2n statements. Ignore the coefficient 2, to give O(n)

**Task 2**

1. A “brute force” method is being used to crack a password. The password is exactly 4 uppercase alphabetic characters.

(a) In a worst case scenario, how many permutations would have to be checked to find the correct password?

264 = 456,976

(b) On average, how many would need to be checked?

264/2= 456,976 / 2 = 228,488

(c) What is the Big-O time complexity of this algorithm?

O(26n) which is exponential.

2. Look at the following graphs. A graph which has an edge connecting every vertex is said to be “complete”.

How many edges are there in each graph? 3, 6, 10, 15

Verify that for a graph of n vertices, there are n(n-1)/2 edges.

3(3-1)/2 = 3; 4(4-1)/2=6; 5(5-1)/2=10; 6(6-1)/2=15

This is the same problem as the following:

“There are n people at a party. Each person shakes hands with every other person. How many handshakes take place?”

(a) Assuming that each traversal of an edge is just one operation, what is the time complexity of traversing each edge in a “complete” graph just once?

O(n2), ignoring the less significant terms and the constant.

n(n-1)/2 = (n2 – n)/2 = ½n2 – ½n

(b) A computer game has 5 doors which have to be opened in a particular sequence in order to progress to the next step. In how many different orders can the doors be opened?

5! = 120

(c) What is the order of complexity of the problem if there are n doors? O(n!)

(d) Suppose there were 7 doors. How many doors would need to be opened, on average, to find the correct door using a “Brute Force” method? 7! / 2 = 2,520.

3. In the Fibonacci sequence 0, 1, 1, 2, 3, 5, 8 … , each number after the first two is the sum of the two previous numbers.

What will be the next two numbers in the sequence? 13, 21

Here are two subroutines written in Python each designed to find the first n numbers in the Fibonacci series.

**Subroutine 1: (a recursive routine)**

def fibonacci(n):

if n == 0:

return 0

if n == 1:

return 1

return fibonacci(n-1) + fibonacci(n-2)

**Subroutine 2: (an iterative routine)**

def fibonacci2(n):

fibNumbers = [0,1] *#list of first two Fibonacci numbers*

*# now append the sum of the two previous numbers to the list*

for i in range(2, n+1):

fibNumbers.append(fibNumbers[i-1] +fibNumbers[i-2])

return fib[n]

Which subroutine do you think has the lower time complexity?

If you can run the Python program *Fibonacci.py*, or write your own code in a language you are familiar with, fill in the times in the following table:

Values on one particular computer are shown below. They will be different on every computer.

|  |  |  |
| --- | --- | --- |
|  | **Time to execute recursive subroutine (milliseconds)** | **Time to execute iterative subroutine (milliseconds)** |
| **n = 10** | 0.3093 | 0.1771 |
| **n = 20** | 16.2403 | 0.3766 |
| **n = 25** | 154.2345 | 0.5158 |
| **n = 30** | 1777.2807 | 0.6621 |
| **n = 35** | 20,109.4839 | 0.9156 |
| **n = 40 (estimate)** | 245,256.0795 | 1.0208 |
| **N = 100 (estimate)** | Several million years | About 2 milliseconds |

Which of O(n) O(n2) O(2n) O (log n) is the approximate time complexity of:

(a) the recursive subroutine? O(2n) (More accurately, 1.618n) where 1.618 is the golden ratio – explanation is beyond the scope of this course!)

(b) the iterative subroutine? O(n)

**Quick quiz on Big-O notation**

Tick the statements which are correct:

(a) The aim of the Big-O notation is to give a rough idea of how time and/or memory requirements will grow as the problem gets bigger

True. It measure time complexity or space complexity

(b) The statement *“The worst case run-time complexity of algorithm A is O(n2)”* means that *“Algorithm A takes at most c x n2 steps (where c is a constant) to solve a problem of size n (for large n)”*

True (approximately) – it may not be true for small values of n, when a term in n and a constant may be significant

(c) Problems of complexity O(1) have only one statement, however large the problem

False – there may be any number of statements but the number stays constant however large the problem becomes

(d) A problem with complexity O(100n) is of a different order of magnitude from one of complexity O(n)

False – the constant is irrelevant and is ignored

(e) An algorithm of complexity O(n3) is useless for any practical purpose

False – problems that grow in polynomial time are not considered insoluble or “intractable”

(f) Hashing is an example of a problem of time complexity O(1)

True – however many values are to be hashed, the time taken to execute the hashing algorithm remains constant

(g) Some problems may be O(n) for small values of n but O(n2) for large values of n

False – the time complexity does not change

(h) “Divide and conquer” algorithms typically have time complexity O(log n) and are very efficient

True