Design & Analysis of Algorithms

Lecture 2

Asymptotic Analysis

- Asymptotic Notation is **used to describe the running time of an algorithm** - how much time an algorithm takes with a given input, n.

- Asymptotic analysis of an algorithm refers to defining the mathematical boundation/framing of its run-time performance.

- Using asymptotic analysis, we can very well conclude the **best case**, **average case**, and **worst case** scenario of an algorithm.

- Asymptotic analysis is input bound i.e., if there's no input to the algorithm, it is concluded to work in a constant time. Other than the "input" all other factors are considered constant.

- Asymptotic analysis refers to computing the running time of any operation in mathematical units of computation.

- For example, the running time of one operation is computed as ***f*(n)** and may be for another operation it is computed as ***g*(n^2)**.

- This means the first operation running time will increase linearly with the increase in **n** and the running time of the second operation will increase exponentially when **n** increases.

- Similarly, the running time of both operations will be nearly the same if **n** is significantly small.

- Usually, the time required by an algorithm falls under three types –

- **Best Case** – Minimum time required for program execution.
- **Average Case** – Average time required for program execution.
- **Worst Case** – Maximum time required for program execution.

Asymptotic Notations

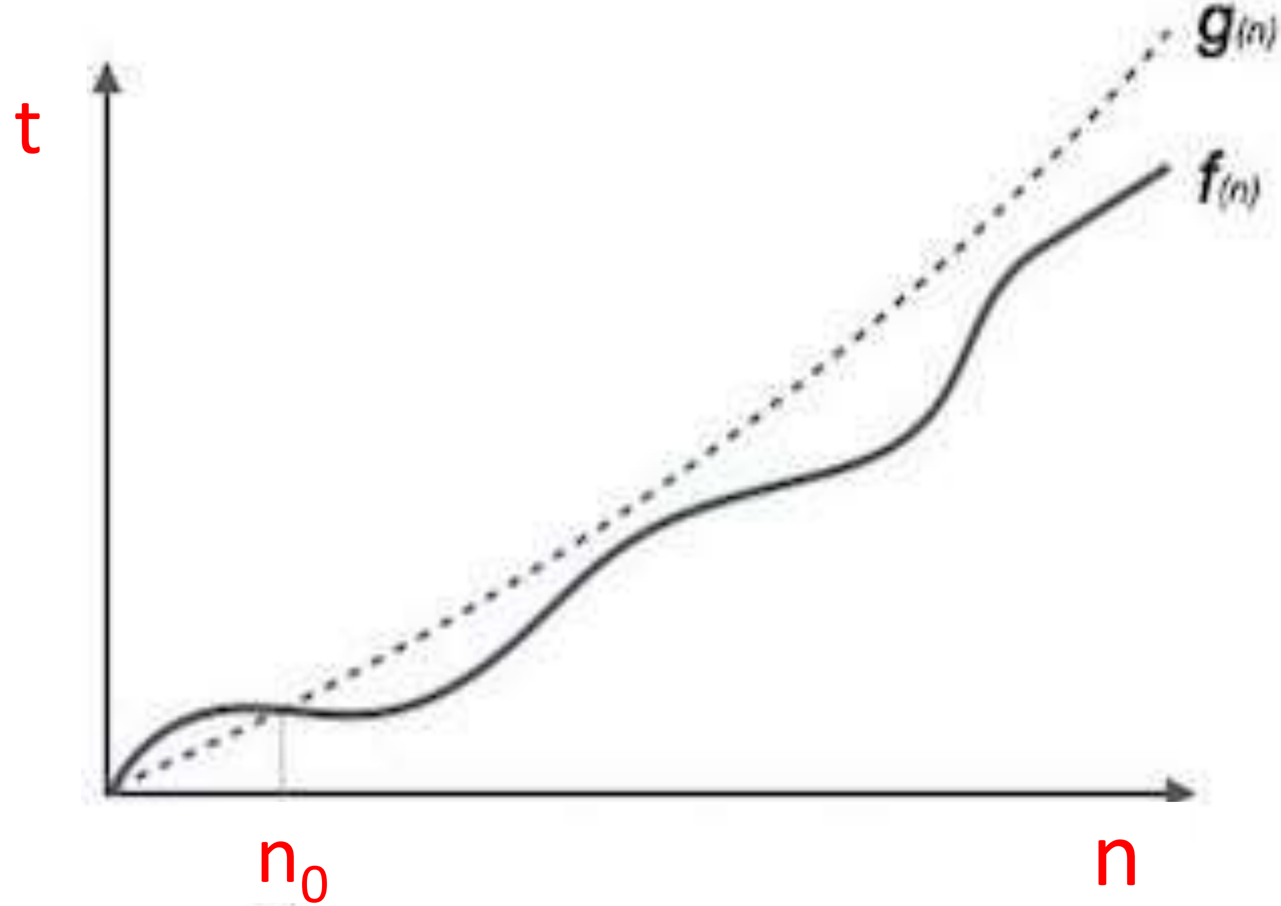
- Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

- **O** Notation
- **Ω** Notation
- **θ** Notation

Big Oh Notation, O

- The notation O(n) is the formal way to express the **upper bound** of an algorithm's running time.

- It measures the **worst case** time complexity or the longest amount of time an algorithm can possibly take to complete.

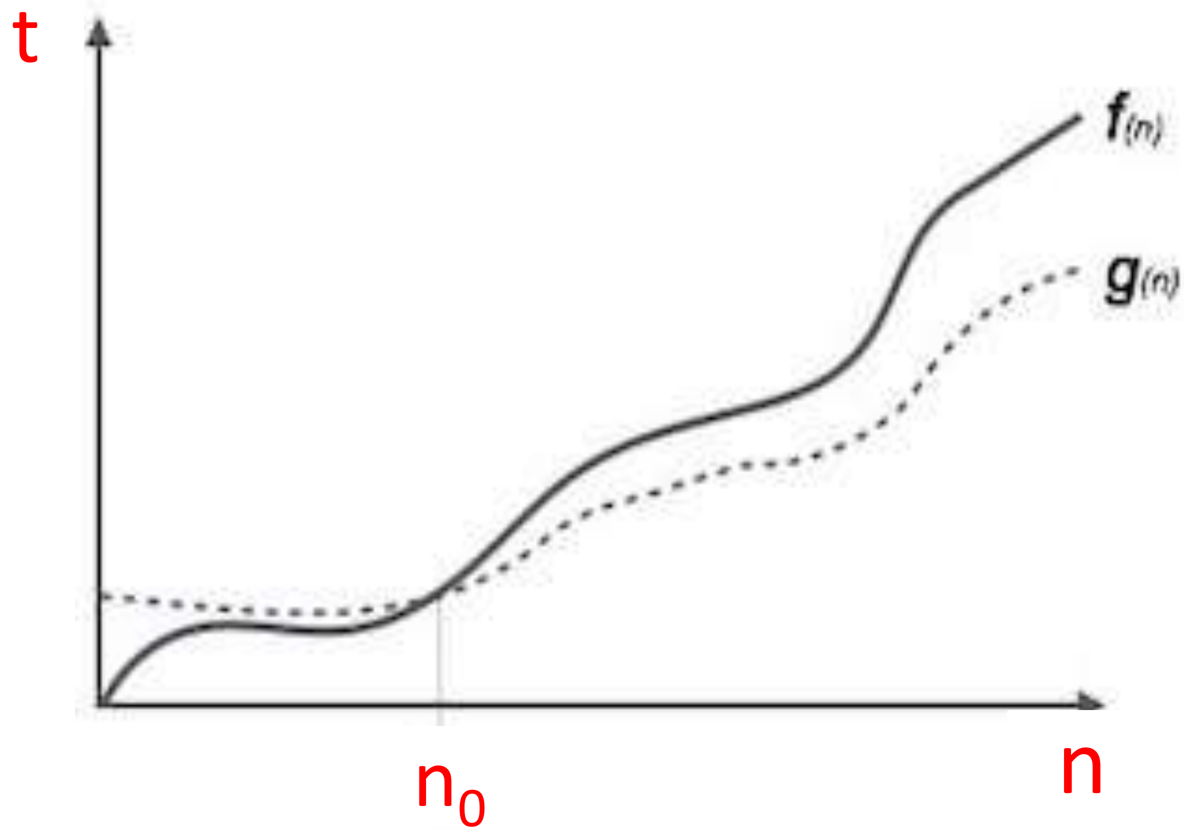


For example, for a function ***f*(n)**

***f*(n) = O(*g*(n))** : there exists $c > 0$ and $n_0 \geq 0$ such that **f (n) ≤ c.g (n)** for all $n \geq n_0$.

Omega Notation, Ω

- The notation **Ω(n)** is the formal way to express the **lower bound** of an algorithm's running time. It measures the **best case** time complexity or the best amount of time an algorithm can possibly take to complete.



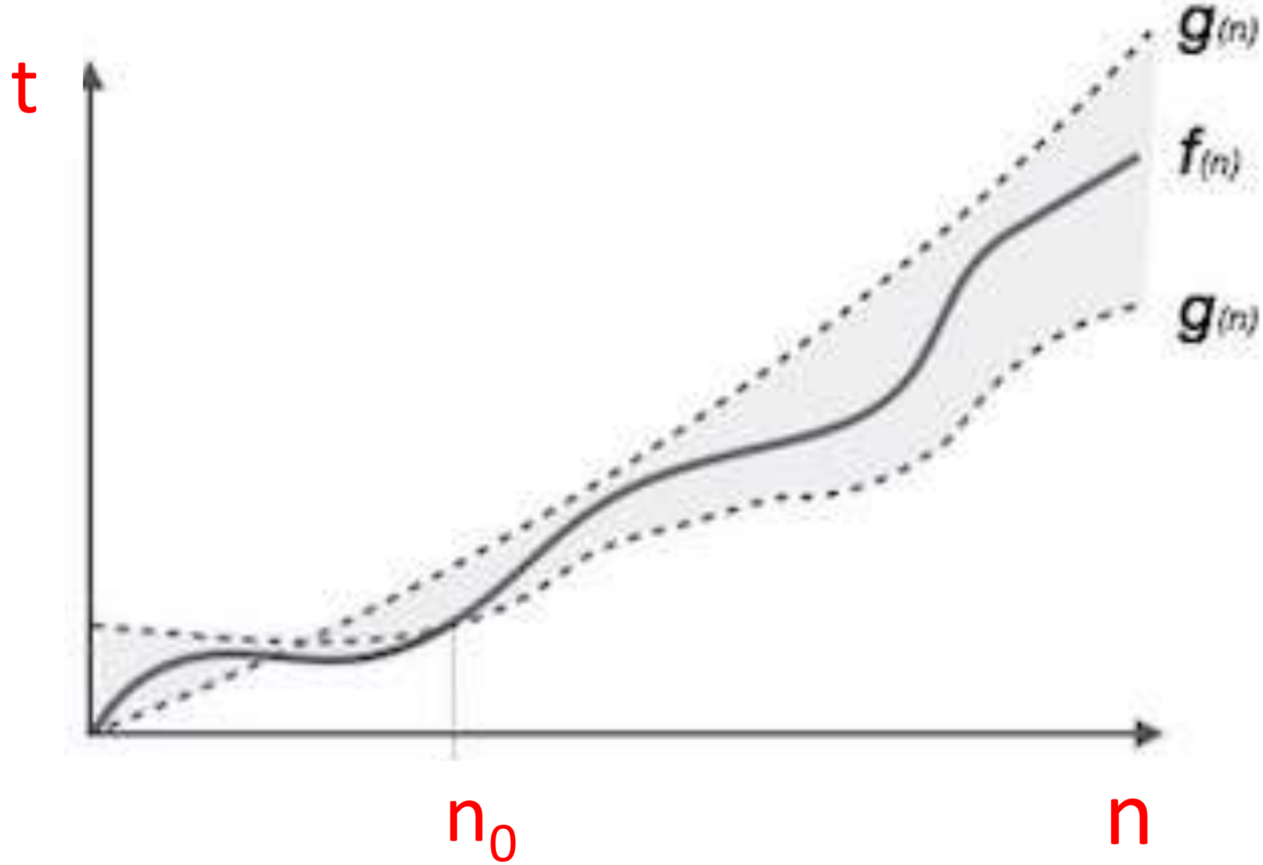
For example, for a function ***f*(n)**

f(n) = Ω (g(n)) : there exists $c > 0$ and $n_0 \geq 0$ such that **f(n) ≥ c.g(n)** for all $n \geq n0$.

Theta Notation, θ

- The notation **θ(n)** is the formal way to express **both the lower bound and the upper bound** of an algorithm's running time. i.e. it represents the **Average** running time.

- It is represented as follows –



For example, for a function ***f*(n)**

$f(n) = \theta\ (g(n))$ if and only if

$C1.g(n) \leq f(n) \leq C2.g(n)$

for all $n \geq n_0$

Example

- Let’s consider an linear search problem

5	7	2	9	4	11	15	3	12

Task - To find an element X?

If x = 5 , it is Best case → Ω (1)
If x = 12, it is Worst case → O(n)
If x = 4, it is Average case → θ (n/2) → θ(n)

Common Asymptotic Notations

- Following is a list of some common asymptotic notations –

constant	–	$O(1)$
logarithmic	–	$O(\log n)$
linear	–	$O(n)$
n log n	–	$O(n \log n)$
quadratic	–	$O(n^2)$
cubic	–	$O(n^3)$
polynomial	–	$n^{O(1)}$
exponential	–	$2^{O(n)}$

Rates of Growth

	$\lg n$	n	$n \lg n$	n^2	n^3	2^n
1	0.0	1.0	0.0	1.0	1.0	2.0
2	1.0	2.0	2.0	4.0	8.0	4.0
5	2.3	5.0	11.6	25.0	125.0	32.0
10	3.3	10.0	33.2	100.0	1000.0	1024.0
15	3.9	15.0	58.6	225.0	3375.0	32768.0
20	4.3	20.0	86.4	400.0	8000.0	1048576.0
30	4.9	30.0	147.2	900.0	27000.0	1073741824.0
40	5.3	40.0	212.9	1600.0	64000.0	1099511627776.0
50	5.6	50.0	282.2	2500.0	125000.0	1125899906842620.0
60	5.9	60.0	354.4	3600.0	216000.0	1152921504606850000.0
70	6.1	70.0	429.0	4900.0	343000.0	1180591620717410000000.0
80	6.3	80.0	505.8	6400.0	512000.0	1208925819614630000000000.0
90	6.5	90.0	584.3	8100.0	729000.0	1237940039285380000000000000.0
100	6.6	100.0	664.4	10000.0	1000000.0	1267650600228230000000000000000.0

- In this chart, it shows the value for these classes over a wide range of input sizes.

- We can see that when the input is small, there is not a significant difference in the values, but once the input value gets large, there is a big difference.

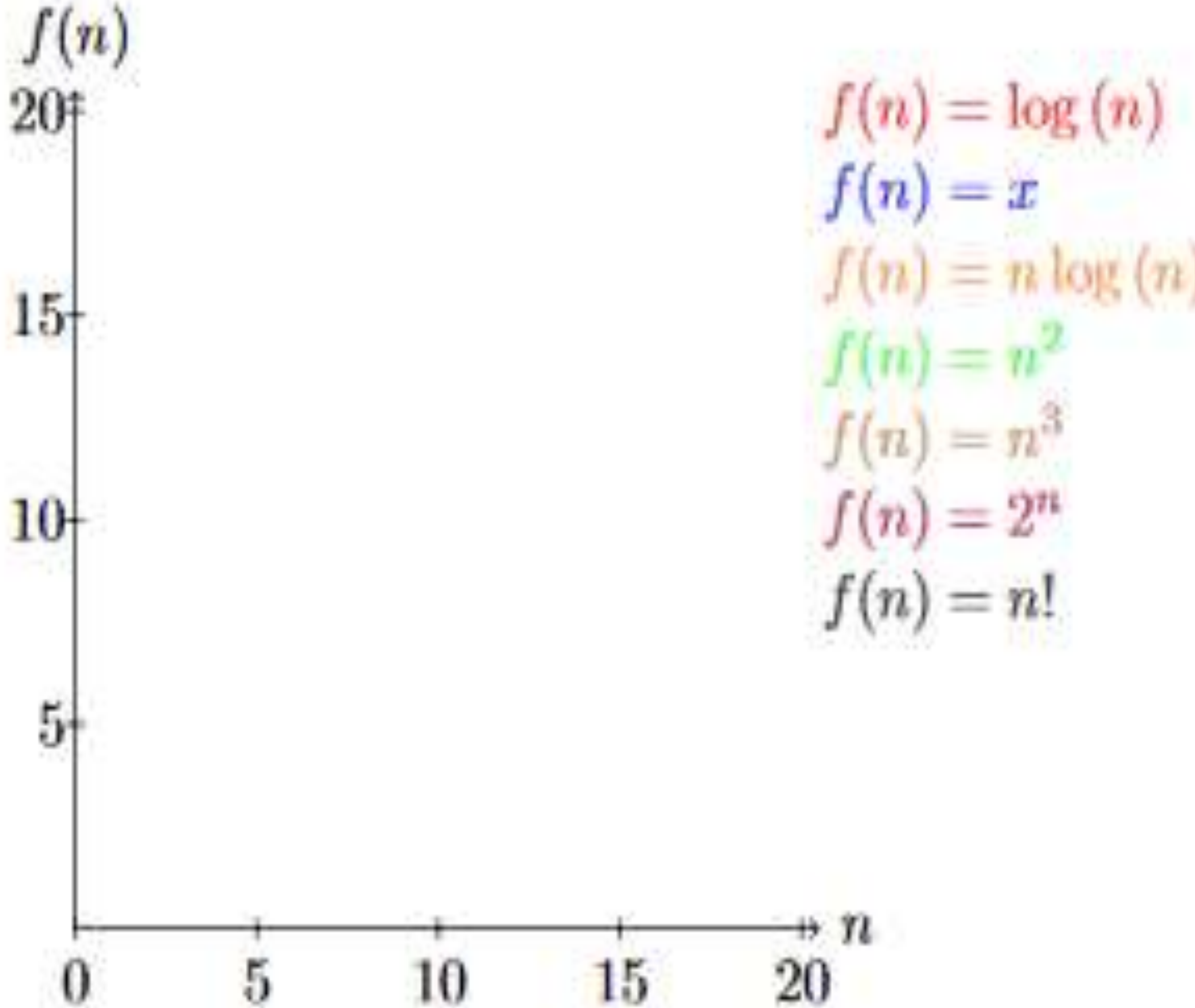
- Because of this, we will always consider what happens when the size of the input is large, because small input sets can hide rather dramatic differences.

- Because the faster growing functions increase at such a significant rate, they quickly dominate the slower-growing functions.

- This means that if we determine that an algorithm’s complexity is a combination of two of these classes, we will frequently ignore all but the fastest growing of these terms.

- For example, if we analyze an algorithm and find that it does $x^3 + 30x$ comparisons, we will just refer to this algorithm as growing at the rate of x^3.

  This is because even at an input size of just 100 the difference between x^3 and $x^3 + 30x$ is only 0.3%.


$f(n)$
20
15
10
5
0
5
10
15
20
n
$f(n) = \log(n)$
$f(n) = x$
$f(n) = n\log(n)$
$f(n) = n^2$
$f(n) = n^3$
$f(n) = 2^n$
$f(n) = n!$

The Scale of Complexity

$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < n^4$

$------------< 2^n < 3^n < 4^n$ ------- n^n

Math Formulae – refer slide

Examples

- Here are the g(n), expressions with their constants dropped and lower order terms removed:

$g(x)$	Simplified
$n^2 + 3n$	n^2
n^4	n^4
$3n - 1$	n
$\log_2 2x$	$\log_2 x$

- There is one interesting simplification there, where $\log_2 2x$, becomes $\log_2 x$. That is because

$$\log_2 2x = \log_2 x + \log_2 2$$

and since $\log_2 2$ is a lower order, we can drop it.

Step 5

- Now that we've simplified the expressions, it should be easy to match them:

$f(x)$	Simplified	$g(x)$	Simplified
$n^2 + 2n - 10$	n^2	$n^2 + 3n$	n^2
$n^3 * 3n$	n^4	n^4	n^4
$n + 30$	n	$3n - 1$	n
$\log_2 x$	$\log_2 x$	$\log_2 2x$	$\log_2 x$

Problems

INFORMAL summary

- $f(n) = O(g(n))$ roughly means $f(n) \leq g(n)$
- $f(n) = \Omega(g(n))$ roughly means $f(n) \geq g(n)$
- $f(n) = \Theta(g(n))$ roughly means $f(n) = g(n)$
- $f(n) = o(g(n))$ roughly means $f(n) < g(n)$
- $f(n) = w(g(n))$ roughly means $f(n) > g(n)$

We use these to classify algorithms into classes, e.g. n, n^2, $n \log n$, 2^n.

Some Logarithmic Properties

$$\log_a 1 = 0$$

$$\log_a a = 1$$

$$\log_a(mn) = \log_a m + \log_a n$$

$$\log_a \frac{m}{n} = \log_a m - \log_a n$$

$$\log_a m^n = n \cdot \log_a m$$

$$\log_a m = \log_b m \cdot \log_a b$$

$$\log_a m = \frac{\log_b m}{\log_b a}$$

$$\log_a b = \frac{a}{\log_b a}$$

$$\log_a x = \frac{\ln a}{\ln x}$$

1. Assume that each of the expressions below gives the processing time T(n) spent by an algorithm for solving a problem of size n. Select the dominant term(s) having the steepest increase in n and specify the lowest Big-Oh complexity of each algorithm.

Expression	Dominant term(s)	$O(\ldots)$
$5 + 0.001n^3 + 0.025n$		
$500n + 100n^{1.5} + 50n\log_{10} n$		
$0.3n + 5n^{1.5} + 2.5 \cdot n^{1.75}$		
$n^2 \log_2 n + n(\log_2 n)^2$		
$n \log_3 n + n \log_2 n$		
$3 \log_8 n + \log_2 \log_2 \log_2 n$		
$100n + 0.01n^2$		
$0.01n + 100n^2$		
$2n + n^{0.5} + 0.5n^{1.25}$		
$0.01n \log_2 n + n(\log_2 n)^2$		
$100n \log_3 n + n^3 + 100n$		
$0.003 \log_4 n + \log_2 \log_2 n$		

Answer

Expression	Dominant term(s)	$O(\dots)$
$5 + 0.001n^3 + 0.025n$	$0.001n^3$	$O(n^3)$
$500n + 100n^{1.5} + 50n\log_{10} n$	$100n^{1.5}$	$O(n^{1.5})$
$0.3n + 5n^{1.5} + 2.5 \cdot n^{1.75}$	$2.5n^{1.75}$	$O(n^{1.75})$
$n^2 \log_2 n + n(\log_2 n)^2$	$n^2 \log_2 n$	$O(n^2 \log n)$
$n \log_3 n + n \log_2 n$	$n \log_3 n,\ n \log_2 n$	$O(n \log n)$
$3 \log_8 n + \log_2 \log_2 \log_2 n$	$3 \log_8 n$	$O(\log n)$
$100n + 0.01n^2$	$0.01n^2$	$O(n^2)$
$0.01n + 100n^2$	$100n^2$	$O(n^2)$
$2n + n^{0.5} + 0.5n^{1.25}$	$0.5n^{1.25}$	$O(n^{1.25})$
$0.01n \log_2 n + n(\log_2 n)^2$	$n(\log_2 n)^2$	$O(n(\log n)^2)$
$100n \log_3 n + n^3 + 100n$	n^3	$O(n^3)$
$0.003 \log_4 n + \log_2 \log_2 n$	$0.003 \log_4 n$	$O(\log n)$

1. For each of the following pairs of functions state whether f(n) is O(g(n)), Ω(g(n)), or Θ(g(n)) Choose the most specific answer

(a) $f(n) = 3n^2 + 7n$ and $g(n) = \frac{5n^3+3}{n}$

Solution:

- f(n) is Θ(g(n)).

- The dominant term in both functions is a square.

(b) $f(n) = \lceil n \rceil^2$ and $g(n) = \lfloor n \rfloor^2$.

Solution:

- f(n) is Θ(g(n)).
- The two functions are equal for integer inputs. For non-integer inputs, floor and ceiling can differ by at most 1.