Rate of growth of functions (order of growth)

Asymptotic Notation

$$f(n) = 2n + 3$$

n : → size of input
→ size of output

$$f_1(n) = 3n^4 + 10n^3$$

Exponential Growth

$$f_2(n) = 2^n$$

Time Complexity

Space Complexity

Time :- mi~~nute~~, ~~hour~~, se~~c~~cond

Matrix Addition : $n \times n$ $\quad n \times n$

$n = 10$ $\quad 100 \quad 100$

100

$n^2 + n^2 + n^2$

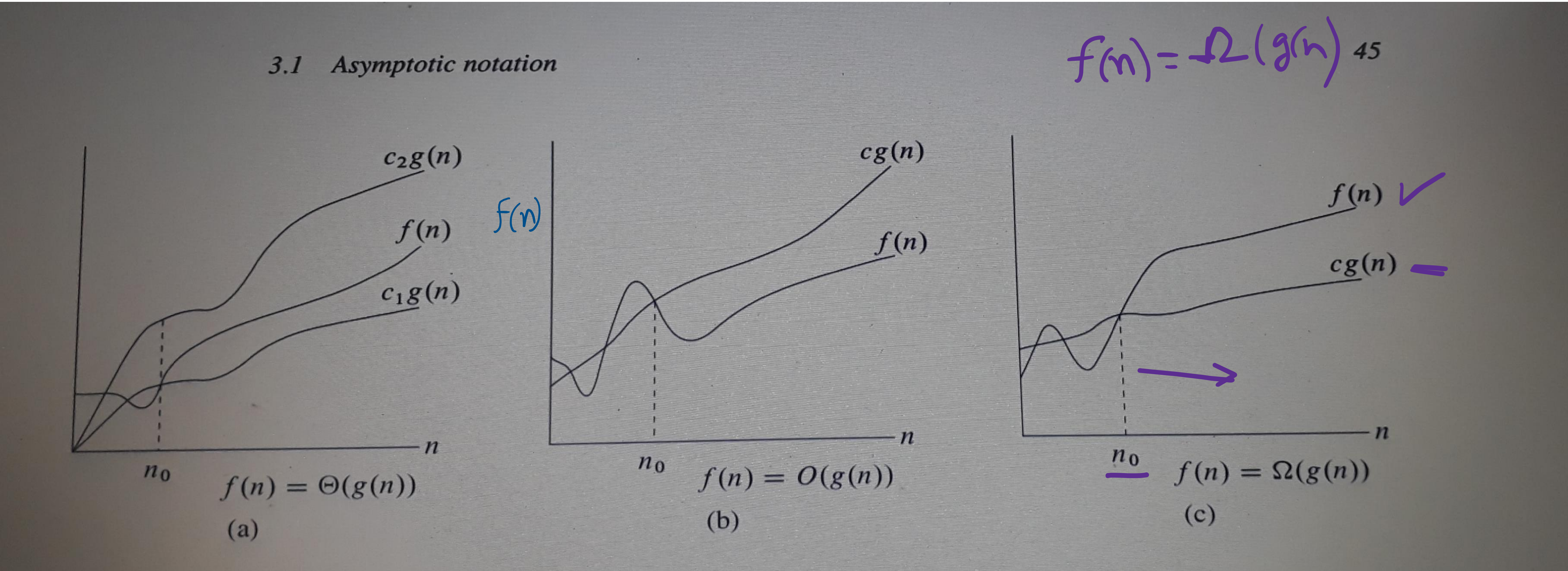
$= 3n^2$

Meaning of Asymptotic Analysis

It means the analysis is valid when the value of n [size of input & size of output] is very large

O-Notation (Big O Notation)

$O(g(n)) = \{ f(n)$: there exist positive constants c and n_0 such that $0 \le f(n) \le c\,g(n)$ for all $n \ge n_0 \}$



Let $f(n) = \frac{1}{2} n^2 - 3n$

$g(n) = n^2$

We want to check whether $f(n) = O(n^2)$

We need to find constant c such that $f(n) <= c\, n^2$

So, $\underline{\frac{1}{2} n^2 - 3n} <= c\, n^2$

Dividing both sides by n^2

$\frac{1}{2} - 3/n <= c$

We can make the inequality hold by taking a constant $c >= \frac{1}{2}$ and $n >= 1$

If f(n) is a polynomial of order k, then $f(n) = O(n^k)$

$f(n) = 4n^5 + 6n^2 + 3$

Example: $f(n) = 4n^3 + 3n^2 + 10$

$f(n) = O(n^3)$

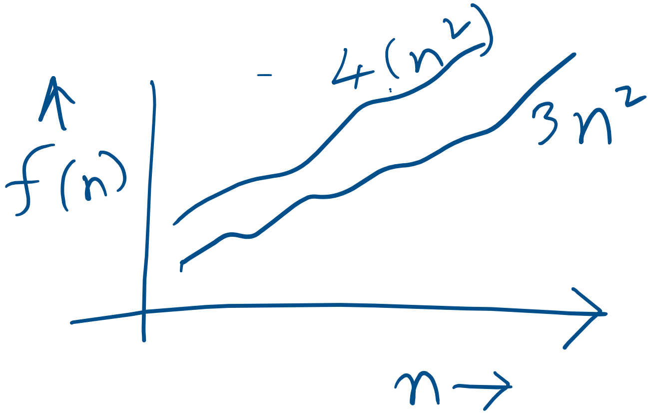
$f(n) = O(n^5)$

Big O notation is not asymptotically tight

Let $f(n) = 3n^2$

$3n^2 <= c\,n^2$

$3 <= c$



Then, $f(n) = O(n^2)$

Also, $f(n) = O(n^3)$

O(1) means constant time that is the time does not depend on size of input or size of output

$f(n) = O(1)$

Θ Notation

$\Theta(g(n))$ = { $f(n)$: there exist positive constants c_1, c_2 and n_0 such that $0 <= c_1\, g(n) <= f(n) <= c_2\, g(n)$

Let $f(n) = ½\, n^2 – 3n$

$½\, n^2 - 3n$

We want to check whether $f(n) = \Theta(n^2)$

We need to find constant c_1, c_2 such that $0 <= c_1\, n^2 <= f(n) <= c_2\, n^2$

$g(n) = n^2$

So, $½\, n^2 – 3n <= c_2\, n^2$

$f(n) <= c_2\, g(n)$

Dividing both sides by n^2

$\frac{½\, n^2 - 3n}{n^2} <= \frac{c_2\, n^2}{n^2}$

$½ – 3/n <= c_2$

We can make the inequality hold by taking a constant $c_2 >= ½$ and $n >= 1$

Now,

$$c_1 \underbrace{g(n)}_{n^2} <= \underbrace{f(n)}_{\frac{1}{2}n^2 - 3n}$$

$c_1\ n^2 <= \frac{1}{2}\ n^2 - 3n$

Dividing by n^2, we get

$c_1\ <= \frac{1}{2} - 3/n$

This inequality can be made to hold by taking $n >= 7$ and $c_1 <= 1/14$

So, the given f(n) is $\Theta\ (n^2)$

Ω Notation (Big Omega Notation):

$\Omega(g(n)) = \{ f(n)$: there exist positive constants c and n_0 such that $0 <= c\, g(n) <= f(n)$ for all $n >= n_0$

Best Case time complexity analysis

Worst Case "

Average Case "

Linear Search in Array

0	1	2	3	4	5	6	7
60	30	10	5	8	20	90	15

```
for(i=0; i<=7; i++)
  if(a[i]==15)
  { k=i;
    break;
  }
```

Search 60

Search 15

$f(n) = n$

n elements

$1/n$

Best case $O(1)$

Worst case $O(n)$

$$1 \times \frac{1}{n} + 2 \times \frac{1}{n} + \cdots + n \times \frac{1}{n}$$

$$= \frac{1}{n}\left(1 + 2 + 3 + \cdots n\right)$$

$$= \frac{1}{\cancel{n}} \frac{\cancel{n}(n+1)}{2}$$

$$= \frac{n+1}{2}$$

$$f(n) = O(n)$$

An example: (Multiplication of matrix A[n x n] and matrix B [n x n]

- Input: matrices *A* and *B*
- Let *C* be a new matrix of the appropriate size
- For *i* from 1 to *n*: → n
    - For *j* from 1 to n: → n^2
        - Let sum = 0
        - For *k* from 1 to n:
            - Set sum ← sum + $A_{ik} \times B_{kj}$ n^3
        - Set C_{ij} ← sum
- Return *C*

$n + n^2 + n^2 + n^2$
$+ n^3 + n^2 + 1$

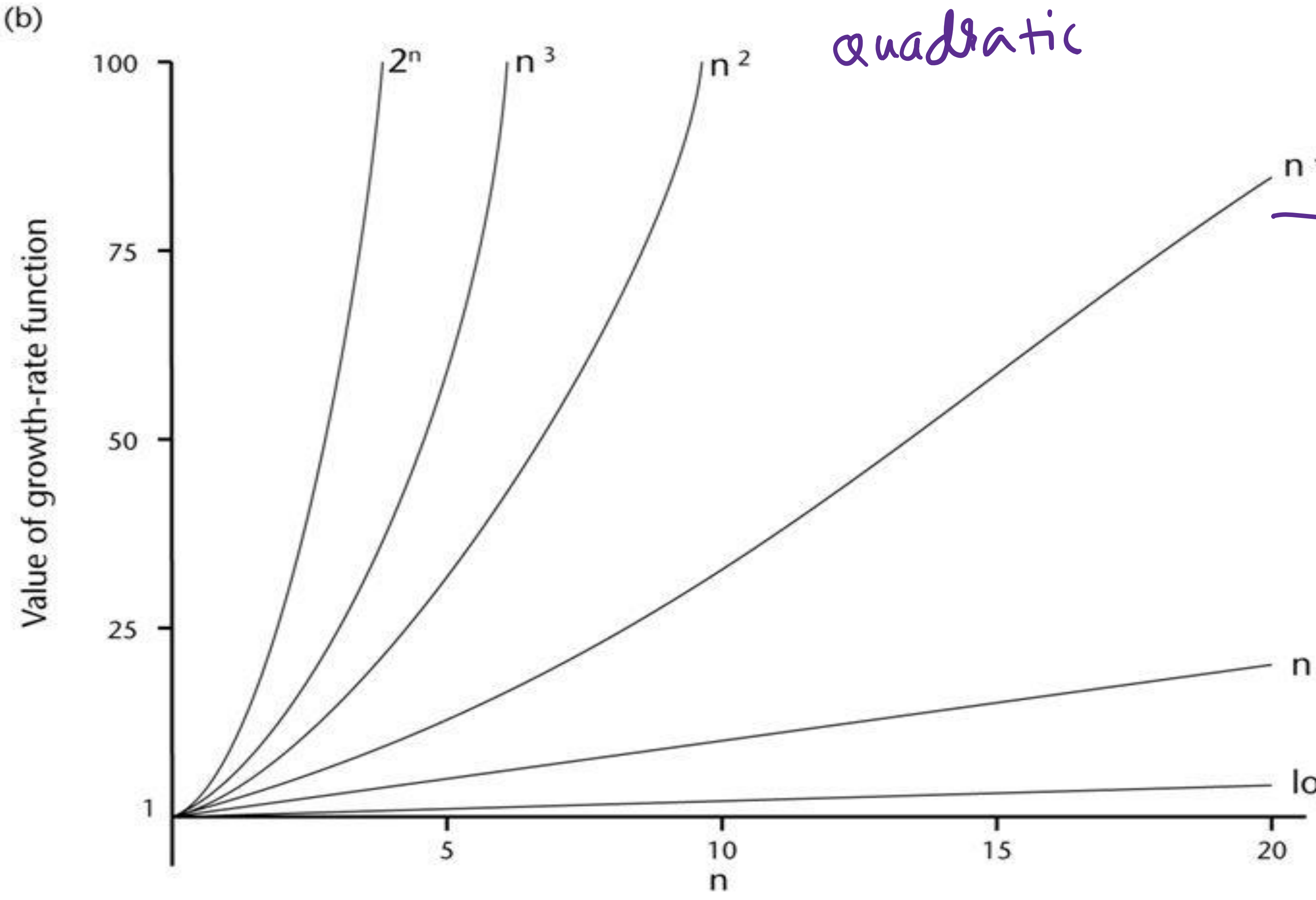
$f(n) = n^3 + 4n^2 + n + 1$

$i = ; i = n; i{+}{+})$

$i = 1 \quad j = 1 \cdots n$
$i = 2 \quad j = 1 \cdots n$

$j = 1 \quad k = 1 \cdots n$
$j = 2 \quad k = 1 \cdots n$
$\vdots$

A Comparison of Growth-Rate Functions (cont.)



quadratic

polynomial time

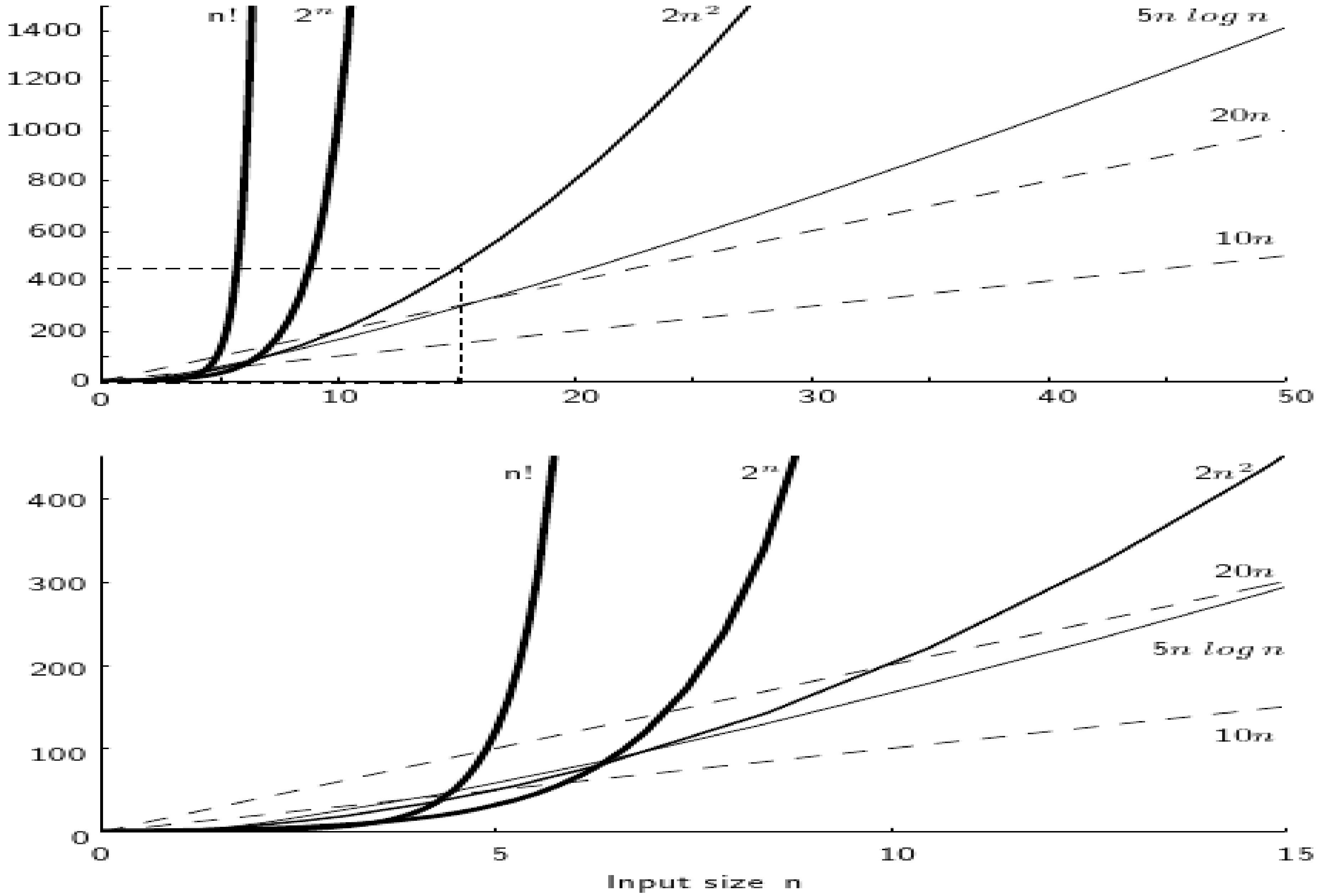
$n^2, n^3, n^4 \ldots$

$f(n) = \log_2 n$

linear growth

logarithmic

n!


n!
2^n
2n^2
5n log n
20n
10n
1400
1200
1000
800
600
400
200
0
0
10
20
30
40
50
n!
2^n
2n^2
20n
5n log n
10n
400
300
200
100
0
0
5
10
15
Input size n

	constant	*logarithmic*	*linear*	*N-log-N*	*quadratic*	*cubic*	*exponential*
n	**O(1)**	**O(log *n*)**	**O(*n*)**	**O(*n* log *n*)**	**O(n^2)**	**O(n^3)**	**O(2^n)**
1	1	1	1	1	1	1	2
2	1	1	2	2	4	8	4
4	1	2	4	8	16	64	16
8	1	3	8	24	64	512	256
16	1	4	16	64	256	4,096	65536
32	1	5	32	160	1,024	32,768	4,294,967,296
64	1	6	64	384	4,069	262,144	1.84×10^{19}

Y=m

	Also called	$n = 100$	$n = 10{,}000$	$n = 1{,}000{,}000$
$O(1)$	Constant time	0.000001 sec.	0.000001 sec.	0.000001 sec.
$O(\lg n)$	Logarithmic time	0.000007 sec.	0.000013 sec.	0.00002 sec.
$O(n)$	Linear time	0.0001 sec.	0.01 sec.	1 sec.
$O(n \lg n)$		0.00066 sec.	0.13 sec.	20 sec.
$O(n^2)$	Quadratic time	0.01 sec.	100 sec.	278 hours
$O(n^3)$	Cubic time	1 sec.	278 hours	317 centuries
$O(2^n)$	Exponential time	10^{14} centuries	10^{2995} centuries	10^{30087} centuries
$O(n!)$	Factorial time	10^{143} centuries	10^{35645} centuries	N/A

Linear List: An ordered list of elements

( )

(10)

predecessor
successor

(10 8)

(10 8 20 30 15)

Operations? — scan the list from left to right
& from right to left

- Insert a new element

(10 8 20 30 15)

Insert 50 at 2nd position

(10 50 8 20 30 15)

- Store an element

e.g. Store <u>70</u> at <u>3rd</u> position

(10 50 70 20 30 15)

Delete: (10 15 70 ~~20~~ 30 15)

e.g. delete 20 from the list

(10 15 70 30 15)

Finding the <u>length</u> Of the list

Retrieving an element at a particular position.
e.g. Retrieve the 3rd element

Create: Empty list

Stack: A stack is a linear list where all insertions and deletions are made at one end of the list. This end is called the 'top' of the stack.