* Slope intercept form of a line $\Rightarrow y = mx + c$ (slope, y-intercept)

$m = \tan\theta = \frac{a}{b} = a/b$

* General form to slope intercept form

$ax + by + c = 0 \Rightarrow y = -a/b\,x - c/b$

$w_1x_1 + w_2x_2 + w_0 = 0$

H.P $\rightarrow$ $w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_dx_d + w_0 = 0$ $\rightarrow$ d dimensions.

$\pi \rightarrow w^Tx + b = 0$

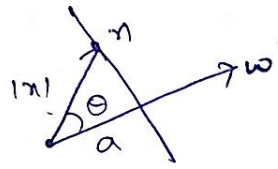
d-dim

* DOT product $\Rightarrow x_1 \cdot x_2 = |x_1||x_2|\cos\theta = x_1^T x_2 = \sqrt{x_{11}^2 + x_{12}^2} \cdot \sqrt{x_{21}^2 + x_{22}^2} \cdot \cos\theta$

$$\cos\theta = \frac{x_{11}x_{21} + x_{12}x_{22}}{\sqrt{x_{11}^2 + x_{12}^2}\ \sqrt{x_{21}^2 + x_{22}^2}} = \frac{x_1 \cdot x_2}{|x_1|\,|x_2|}$$

* Dot product of orthogonal vector is '0'.

*



$|x|\cos\theta + w_0 = 0$ $\quad a = -w_0$

$\cos\theta = \frac{-w_0}{|x|} = a/|x|$

* Distance of plane to origin $\Rightarrow \frac{-w_0}{|w|}$, $\uparrow$ $dis(0, \pi)$ $\boxed{\frac{obs\,(w_0)}{\|w\|_2}}$

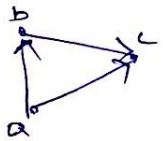
* +ve half space $\rightarrow$ $w^Tx_1 + w_0 \geq 0$ $\rightarrow$ towards direction of w.

-ve half space $\rightarrow$ $w^Tx_1 + w_0 < 0$ $\rightarrow$ opposite direction of w.

* Distance of point x_1 form plane $w^Tx + b_0 = 0$ is $\boxed{\frac{w^Tx_1 + b_0}{\|w\|_2}}$

* unit vector of any vector $w \Rightarrow \boxed{\frac{w}{\|w\|_2}} = \hat{w}$

*

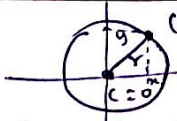


addition of vectors $\Rightarrow \vec{ab} + \vec{bc} = \vec{ac}$

Substraction of vectors

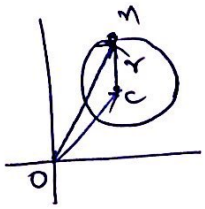
$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \end{bmatrix} \Rightarrow x - y = \begin{bmatrix} x_1 - y_1 \\ x_2 - y_2 \\ \vdots \end{bmatrix}, \quad x + y = \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ x_3 + y_3 \\ \vdots \end{bmatrix}$$

* equation of circle = $x^2+y^2=r^2$



(2) $r=\sqrt{x^2+y^2}$

* equation of circle at centre not equal to origin



$\Rightarrow \vec{Ox} = \vec{Oc} + \vec{cx} \Rightarrow \vec{cx} = \vec{Ox} - \vec{Oc} \Rightarrow$ $r=\sqrt{(x_1-c_1)^2+(x_2-c_2)^2}$

* point inside circle $\rightarrow \|x-c\| < r$, outside $\rightarrow \|x-c\| > r$ on the circle $\rightarrow \|x-c\| = r$.

hyper ellipsoid at centre = 0

* ellipse equations $\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$ $\quad \frac{x^2}{a^2}+\frac{y^2}{b^2}+\dots=1$

* square area = a^2, rectangle area = $a \times b$

CALCULUS FOR ML & AI

* Domain of $f(x)$ → set of possible values which a function can take.
* Range of $f(x)$ → values taken by function on y-axis.
* $f(x)$ is continuous at 'a' if (i) $f(a)$ is defined at 'a'.

(ii) $\lim_{x\to a^+} f(x) = \lim_{x\to a^-} f(x) = f(a)$ (iii) $\lim_{x\to a} f(x)$ exists.

* $\left.\frac{df(x)}{dx}\right|_{x=y}$ = slope of T_4. if slope >0 → curve is increasing
  slope <0 → curve is decreasing
* $\tan\theta = \frac{\Delta y}{\Delta x}$, $\left.\frac{dy}{dx}\right|_{x=x_1} = \lim_{(x_2-x_1)\to 0} \frac{y_2-y_1}{x_2-x_1} = \lim_{\Delta x\to 0} \frac{\Delta y}{\Delta x}$

* Derivative of common functions:-

(i) $\frac{dx^n}{dx} = n.x^{n-1}$; $n \neq 0$, n-rational number.

(ii) $\frac{d\log(x)}{dx} = 1/x$ (iii) $\frac{d(e^x)}{dx} = e^x$

(iv) $\frac{d|x|}{dx} \Rightarrow$

$x>0 \rightarrow +1$

$x<0 \rightarrow -1$

$x=0 \rightarrow$ not defined as $\lim_{x\to 0^+} \neq \lim_{x\to 0^-}$

* Function is not differentiable at
  → where ever the graph or curve is non-smooth
  → where ever the function is discontinuous.

* Rules of differentiation :-

(i) $\frac{d}{dx}(f(x)+g(x)) = \frac{df(x)}{dx} + \frac{dg(x)}{dx}$ (ii) $\frac{d}{dx}(\text{constant}) = 0$

(iii) Chain rule : $\frac{d\,f(g(x))}{dx} = f'(g(x)) \cdot g'(x)$

(iii) product rule : $\frac{d\,f(x).g(x)}{dx} = f(x)\frac{dg}{dx} + g(x)\frac{df}{dx}$

(iv) Quotient rule : $\frac{d}{dx}\left[\frac{f(x)}{g(x)}\right] = \frac{f'(x)\,g(x) - g'(x)\,f(x)}{[g(x)]^2}$

$g(x) \neq 0$

---

* MAXIMA & MINIMA

minima → −ve → 0 ← +ve ⇒ $\frac{df(x)}{dx}$ is increasing fn

maxima → +ve → 0 ← −ve ⇒ $\frac{df(x)}{dx}$ is decreasing fn

→ $\frac{d}{dx}\left(\frac{df(x)}{dx}\right) > 0$

→ $\frac{d}{dx}\left(\frac{df(x)}{dx}\right) < 0$

Partial derivatives :-

$z = x_1^2 + x_2^2 \Rightarrow \nabla_x z = \begin{bmatrix} \frac{\partial z}{\partial x_1} \\ \frac{\partial z}{\partial x_2} \end{bmatrix}$

$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

Let $z = f(x,y)$

@ ophma $\Rightarrow \nabla z = 0 \Rightarrow \nabla z = \begin{bmatrix} \frac{\partial z}{\partial x} \\ \frac{\partial z}{\partial y} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

→ we have special cases where the derivative of the function is '0' but it doesn't have either minima or maxima. Those are called 'SADDLE POINTS' ex :- $f(x) = x^3$ at $x = 0$

$z = x^2 - y^2$ at $x = 0$, $y = 0$

→ Gradient descent ⇒ $x_1 = x_0 + \alpha \left(-\frac{df}{dx}\right)_{x=x_0}$

x_0 y_0 (randomly pick) ↓ +ve constant → Learning rate.

$z = f(x,y)$ → $x_1 = x_0 + \alpha\left(-\frac{\partial f}{\partial x}\right)_{x_0,y_0}$, $y_1 = y_0 + \alpha\left(-\frac{\partial f}{\partial y}\right)_{x_0,y_0}$

→ repeat till $\left(\frac{\partial f}{\partial x}\right)_{x_t,y_t}$ & $\left(\frac{\partial f}{\partial y}\right)_{x_t,y_t}$ are close to zero.

KEY words and identifiers

→ reserved words in python, cannot be used as variables, functions, classes and other identifiers.

→ 35 keywords. * import keyword, keyword.kwlist.

→ identifier identifies classes, functions, variables etc;

→ identifier → allows ((a-z), (A-Z), (0-9), (_))
→ cannot begin with → digit
→ keyword cannot be used as identifier.

→ Comments → #, multiline comments, ''' _ ''', """ _ """.
↳ do not come under the category of statements.

→ syntax to access doc string → (classname.__doc__).

→ '\' → used for multiline statements.

→ pr(id(x)) → prints the storage location of x.

→ Intervals [-5, 256] → all variables with similar value direct to same location, outside [-5, 256] → variables with similar values direct to different locations.

* type() - to know which class a variable/value belong to
→ type(a)
* isinstance() - to check if a given object belong to particular class. → (isinstance(1+2j, complex))

Mutable → lists, dictionaries, sets

immutable → integers, floats, Booleans, strings, tuples

→ set is unordered collection of unique elements.

→ sets donot allow repetition, slicing.

Output formating:-

print("The value is a is {} and b is {}".format(a,b))

print("The value is a is {0} and b is {1}".format(a,b))

print("The value is a is {a} and b is {b}".format(a=2,b=3))

print("The value is a is {0} and b is {b}".format(0,b=3))

Operators:-

Arithmetic:- addition(+), substraction(-), multiplication(*), division(/), modulo division(%), floor division(//), exponent(**)

Comparision:- >, <, ==, !=, >=, <=

logical:- and, or, not Bitwise:- &(and), |(or), NOT(~), XOR(^), Bitwise right shift(>>), Bitwise left shift(<<) assignment:- =, +=, -=, *=, /=, %=, **= identity operators:- is, is not (pointing to same location)

membership:- in & not in (dict → it checks key values)

* The 'ELSE' block of while loop runs only if the test expression is returning 'False' and if there is no 'Break' statement.

* Iterating over a sequence is called Traversal.

* range(start, stop, step) → not store values in memory, just remember, start, stop, step.

* For loop can have a optional 'else' - executed if the items in the 'sequence used in for loop is exhausted". when "break is used", "else is ignored".

Lists:- (i) len(A) → length of list

(ii) list insert:- lst.insert(2, "three") → will add element 4, at location x.

(ii) list remove:- lst.remove("two") → it will remove first occurrance of 'two'.
→ Error when "two" not present in lst.

(iii) list append:- lst.append("five") → will append at the last.
lst.append(lst2) ⇒ [lst1, ~~lst2~~, [lst2]] → appends the list lst2 as is to lst1

(iii) list extend: lst1.append(list2) → [list1, [list2]] → adds lst2 directly to list1
lst1.extend(list2) → [list1, list2] → adds elements to list1.

(iv) list delete:- lst.pop(i) → index value / lst.remove(value)

(v) list reverse:- lst.reverse() → print(lst) → it will be reversed

(v) list sort() :- sorts the original list itself. → default - ASC
list.sort() → reverse = true for desc

(vi) list sorted() :- returns the copy of sorted list.
sorted(list)

NOTE:- whenever we want to sort a list make sure all elements belong to same datatype.

→ we can have multiple reference to list and when reference object change, main object also changes.

→ str.split(',') → used to split a string using ',' - result in list.

Slicing → list [start(included) : stop(excluded) : step]

NOTE:- '+' is like 'extend' → [1,27,3] + [3,4] ⇒ [1,27,3,3,4]

(vii) count:- list.count(element) → counts # the element in list.

→ list comprehension examples:-

* newlist = [i**2 for i in list] * list2 = [i for i in list1 if i>=0]

* lst3 = [(i, i**2) for i in range(10)]

→ nested list comprehensions:-

* list = [[row[i] for row in matrix] for i in range(4)]

Tuples:- mutable object in side tuple (immutable object) can be changed.

* when we have only one element in tuple we need to use (element,) → else the datatype of that object will be of element.

*'+' → concatenation of tuples → (1,2,3) + (4,5,6) = (1,2,3,4,5,6)

* del → we can use "del tuple" to delete tuple.

* count() → tuple.count(1) → gives count of '1' in tuple.

* index() → tuple.index(3) → gives index of first occurance of '3' if '3' not present then error.

* sorted() → Here sort() → doesn't work, only sorted(tuple) works.

* max(), min() → gives max & min, but same datatype elements.

Sets:- unorded collection of unique items.

set.add(): adds single element → set.add(1)

set.update(): adds multiple elements → set.update([1,2,3,4]), set.update((1,2,3,4)), set.update({1,2,3})

→ NOTE:- cannot use dict in set.

Removal of elements from set:-

set.discard(3) → removes '3' - when element not present no error

set.remove(3) → removes '3' - when element not present error (unlike list & tuple)

set.pop() → takes no argument, and removes random element. throws error when set is empty

set.clear() → removes all elements.

Set operations:

Union: set1 | set2 (or) set1.union(set2)

intersection: set1.intersection(set2) (or) set1 & set2

difference: set1.difference(set2) (or) set1 - set2 a=[1,2,3], b=[3,4] a-b=[1,2]

symmetric diff: set1.symmetric_difference(set2) (or) set1 ^ set2 a^b=[1,2,4]

issubset(): set1.issubset(set2)

Frozenset() :- as tuples are the immutable lists, Frozensets are the immutable sets. → Created by frozenset().

* frozenset() → supports copy(), difference(), intersection(), isdisjoint(), issubset(), issuperset(), symmetric_difference() and union().
→ does not support → add or remove elements.

Dictionary :- Dictionary key should be hashable. (list & sets cannot be used as keys in Dictionary)

→ **access :-** dict['key'] or dict.get('key') → NO 'key' then error.

→ **add / modify :-** dict['key'] = value

→ **Deletion / removal :-** dict.pop('key') / dict.popitem() (arbitrary item) / dict.clear() (removes all)
del dict['key']

Methods() :-

→ copy() → dict.copy() → returns copy of dict.

→ fromkeys() → a = {}.fromkeys(['a','b'], 0) creates a dict.

→ keys() → a.keys() → returns key of a. as list

→ values() → a.values() → returns all the values as list

→ items() → a.items() → returns all key-value pairs as a list of tuples.

→ dir() → to display a list of available methods.

Dictionary comprehensions :-

(i) a = {k:v for k,v in d.items() if v>2} → creating new dict

(ii) d = {k+'c' : v*2 for k,v in d.items() if v>2} → perform operation while creating.

Strings:- * String is a sequence of unicode characters.

* Note * Strings are immutable - that is making changes to the existing values is not possible.

* + → concatenation * '*' - repetation

String methods()

(i) upper() → upper values (converts) (iii) split() → str.split(',') (split with ',')

(ii) lower() → lower values (converts) (iv) join() → ' '.join(['a', 'b']) = 'a b'

(v) find() → 'Hing'.find('H') = 0 (returns '-1' if 'H' is not present) (vi) replace() → 'Hing'.replace("H", "D")

(vii) reversed() → reversed(str) → reverses the string

(viii) ~~sort()~~ → ~~str.sort()~~ →

Functions():-

* avoids the repetation of code and make it reusable.
* When return is not done, function returns 'None'.
* Scope of variable is the portion of the program upto which a variable gets recognised.
* The lifetime of the variable inside a function is as long as the function executes.

built in functions:-

* abs() - absolute value, all() - takes an iterable and returns true if all elements are 'true' else returns 'false'. NOTE:- empty list always true.
* divmod() → divmod(9, 2) = (4, 1) (4 ↑ Quotient, 1 → remainder)
* enumerate() → enumerate(list, number) → returns tuple with index starting from 'number' and list elements.
* filter():- The filter() method constructs an iterator from elements of an iterable for which a function returns 'true'.

filter(function, iterable)

* Filter function will loop the input iterator and passes each element of the iterator through the function and creates an iterator with only those ~~functions~~ elements that do not return None, when passed through the function.

isinstance():- isinstance() function checks if the object (first argument) is an instance or subclass of class_name class (second argument). ∴ isinstance (object, class_name)

map():- map (function_name, iterable_name)

→ map will apply the function 'function_name' to each and every element in the iterable 'iterable_name's. The result would be of map class. If we want to get the result in the form of list (or) a tuple (or) a set. then we have to apply the list() (or tuple () (or) set() functions on the map function output.

reduce():-

* reduce() function is for performing some computation on a list and returning the result.

* it applies a rolling computation to sequential values pairs in a list.

```
From functools import reduce
def multiply (x, y):
    return x*y.
product: reduce (multiply, lst)
```

Function arguments:-

(i) Default arguments : assigning a default value to the function argument, when a value is passed in the function call, default argument value is replaced else ~~else~~ default is used.

ex:- def a(b='k'): return b → a( ) → o/p - k a('j') o/p 'j':

NOTE:- At the time of function defination and function call we must make sure that non-default arguments come first and then the default arguments.

(ii) Key word arguments:-

* we should use **kwargs - if we want to handle named arguments.
* **kwargs is considered a dictionary and access is by using keys. access by kwargs[key].

(iii) Arbitrary arguments:-

* *args → represent arbitrary arguments. we ~~dont know~~ can pass any number of arguments and can be access them by index or looping. tuple (args).

Rules of arguments:-

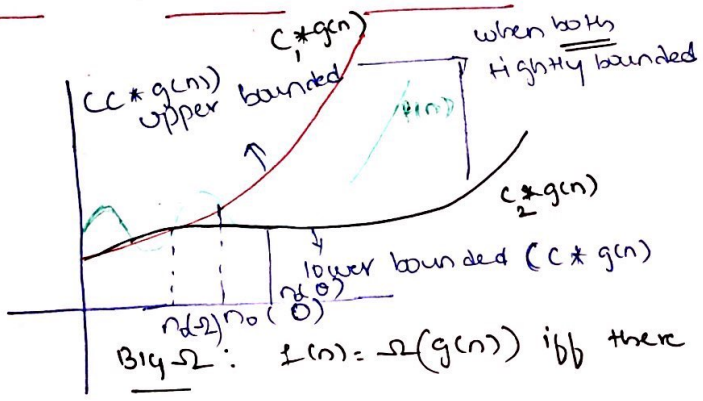
* Keyword arguments must follow positional arguments if any.

INSERTION SORT:-

(i) two subarrays unsorted & sorted (ii) each element in unsorted subarrays compared with sorted subarray and inserted in approriate position.

* Insertion sort is an "inplace" algorithm. as it consumes the constant amount of additional memory irrespective of the i/p size.
* Insertion sort is an stable sorting algorithms as the initial order of the same key value is retained even after sorting using an sorting algorithm.
* Insertion sort is an online sorting algorithms as it can sort the data that is given to us over time.
  → $\Theta(n+m)$ — m → number of inversions; n → is the size of i/p array
* Insertion sort time complexity: Best case: $O(n)$ worst case: $O(n^2)$
  space complexity: $O(1)$

BIG O, Θ (Theta), Ω (omega) :-



BIG O: $f(n) = O(g(n))$ iff and only if there exists some n_0 and c such that $0 \le f(n) \le c * g(n)$ for all $n \ge n_0$.

BIG Θ: $f(n) = \Theta(g(n))$ iff there exists some n_0, c_1 & c_2 s.t $0 \le c_1 * g(n) \le f(n) \le c_2 * g(n)$ for all $n \ge n_0$.

Big Ω: $f(n) = \Omega(g(n))$ iff there exists a n_0, c s.t $0 \le c * g(n) \le f(n)$ for all $n \ge n_0$

NOTE:- if $f(n) = \Theta(g(n))$ then $f(n) = O(g(n))$ & $f(n) = \Omega(g(n))$

Small O & ω (small omega) :- Difference b/w O & o → O → for some c. o → for all c.

Small o :- $f(n) = o(g(n))$ iff $\forall$ (for all) $c > 0$ s.t $0 \le f(n) < c * g(n)$ $\exists$ (there exists) $n_0 > 0$

alternate def:- $f(n) = o(g(n))$ iff $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

small ω :- defination -1:- $f(n) = \omega(g(n))$ iff $g(n) = o(f(n))$

defination 2: $f(n) = \omega(g(n))$ iff $\forall c > 0$ $\exists$ $n_0 > 0$ s.t $0 \le c \cdot g(n) < f(n)$ $\forall n \ge n_0$

defination 3: $f(n) = \omega(g(n))$ iff $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$

NOTE:- O, Θ, Ω, o, ω → are called asymptotic notations.

* mathematical operators to understand the relationship b/w O, Θ, Ω, o, ω
  (i) $f(n) = O(g(n)) \rightarrow f \le g$
  (ii) $f(n) = \Omega(g(n)) \rightarrow f \ge g$
  (iii) $f(n) = \Theta(g(n)) \rightarrow f = g$
  (iv) $f(n) = o(g(n)) \rightarrow f < g$
  (v) $f(n) = \omega(g(n)) \rightarrow f > g$

transitive property :- If $f(n) = \Theta/\Omega/O(g(n))$ & $g(n) = \Theta/\Omega/O(h(n))$ then $f(n) = \Theta/\Omega/O(h(n))$ - transitive property holds for all asymptotic notations

Reflexive relationship :- $f(n) = O/\Omega/\Theta(f(n)) \Rightarrow f(n) \overset{\leq}{\underset{\geq}{=}} f(n)$ - holds for $(O\,\Theta\,\Omega)$

Symmetric relationship :- $f(n) = \Theta(g(n))$ iff $g(n) = \Theta(f(n))$ | $f = g$ iff $g = f$.

Transpose symmetry :- $f(n) = O(g(n))$ iff $g(n) = \Omega(f(n))$ | $f \leq g$ iff $g \geq f$.

$f(n) = o(g(n))$ iff $g(n) = \omega(f(n))$ | $f < g$ iff $g > f$.

NOTE :- TRICHOTOMY DOES NOT HOLD FOR ASYMPTOTIC NOTATIONS. (if $a \nmid b$ are # then any of these $a = b$, $a < b$, $a > b$ are true)

ORDERS OF COMMON FUNCTIONS :-

$O(1) < O(\log\log n) < O(\log n) < O((\log n)^c) < O(n^c) < O(n) < O(n \log^* n) <$

constant $\alpha(\log n)$ double log; log; $c > 1$ poly log; $0 < c < 1$ fraction pow; linear; n log star n

$O(n \log n) = O(\log n!) < O(n^2) < O(n^c) < L_n[\alpha, c] = e^{(c+o(1))(\ln n)^{\alpha}(\ln\ln n)^{1-\alpha}}$

log linear, quasilinear; quadratic; $c > 2$ polynomial; $0 < \alpha < 1$ LNotation

$< O(c^n) < O(n!)$

$c > 1$ exponential; factorial

NOTE :- if $f(n) = a_0 + a_1 n^1 + a_2 n^2 + \dots + a_m n^m$, $f(n)$ is polynomial of degree 'm'.

then $f(n) = O(n^m)$; $O(n^{m+1})$; $O(n^{m+k})$ where $k \geq 0$

$f(n) = \Omega(n^m)$; $\Omega(n^{m-1})$; $\Omega(n^{m-k})$ where $k \geq 0$

$f(n) = \Theta(n^m)$.

Geometric series :- $S_n = \frac{a_1(1-r^n)}{(1-r)}$, $r \neq 1$ where n is # of terms, a_1 first term, r is the common ratio.

finite geometric series $\rightarrow$ $r = \frac{a_2}{a_1}$ n^{th} term $= ar^{n-1}$

→ Infinite geometric series having ratios with an absolute value less than one", → else no sum. use the formula $S = \frac{a_1}{1-r}$

arithmetic progression : terms in the sequence are said to increase by a common difference 'd'. nterm → $a + (n-1)d$, Sum = $n/2\,[2a + (n-1)d]$

sum of n natural numbers → $\frac{n(n+1)}{2}$

sum of squares of n # → $\frac{n(n+1)(2n+1)}{6}$

cubes of n# → $\left[\frac{n(n+1)}{2}\right]^2$

* Divide & conquer → dividing large problem to small subproblems (divide) and combine them in some form to solve the larger problem (conquer).

→ recursion is popular choice.

* a^b →		iterative	recursive
	time complexity	$O(b)$	$O(\lg b)$
	space complexity	$O(1)$	$O(\lg b)$

→ two types of dynamic programing — call stack issue

(i) Top down DP → recursion + DSP (memoization) → top-down

(ii) Bottom up DP → iteration + DSP (tabulation) Bottom up better to use

→ Fibonacci –

	iter	recur
time complexity	$O(n)$	$O(2^n)$
space complexity	$O(n)$	$O(1)$ + call stack

→ Longest common subsequence — iter: $O(m \times n)$; recur: 2^b $b \in \max(m,n)$ — $m \to len(a)$, $n \to len(b)$

→ Merge sort → Best case time complexity } $O(n \log n)$
worst case time complexity }
Best case space complexity } $O(n)$.
worst case space complexity }

Note:- merge sort is the external sorting algorithm

→ Linear search:- worst case time complexity – $O(n)$
Best case time complexity – $O(1)$
worst & best case space complex – $O(1)$

→ Binary search:- ↳ works only on sorted arrays
worst case time complexity – $O(\log n)$ or $\Theta(\log n)$
Best case time complexity – $O(1)$
worst & best case space complex – $O(1)$ or $\Theta(1)$

* matrix multiplication →	recursive	Brute force
time complexity	$\Omega(2^n)$	$O(n^3)$

* subset sum problem → Brute force – 2^n
DP – $O(n * sum)$. if sum = 2^n the DP is waste

* matrix multiplication → optimal sub structure

$$m[i,j] = \begin{cases} 0 & \text{if } i \geq j \\ \min\limits_{i \leq k < j} m[i,k] + m[k+1,j] + p_{i-1} \times p_k \times p_j & \text{if } i < j. \end{cases}$$

* subset sum problem → False if $n \geq 0$ & sum > 0
True if sum $= 0$
$S(n-1, sum)$ (or) $S(n-1, sum - set[n])$.

Lambda functions:- * lamb = lambda x: x**2 ; print (lamb(5))

Use with filter:- evenlist = list(filter(lambda x: (x%2==0), [1,2,3,4,5]))

Use with map:- maplist = list(map(lambda x: x**2, lst))

Use with reduce:- reduces = reduce(lambda x,y: x*y, lst)

* IRRespective of whether an exception occurs or not, the 'finally' block gets executed.
* When no exception occur in try then only else will be executed.

Multithreading & Multiprocessing:-

multiprocessing:- from multiprocessing import process, queue.

Start() → Starts the process, join() → combine the process

→ thread: light weight process with faster context switching

→ from threading import thread.

→ GIL - Global interpreter lock, Python's memory management is not thread safe.

JobLib:- simple parallel computing library in python: from joblib import parallel, delayed

SQL:-

DML: insert, update, delete, call, explain call, lock

DDL: create, alter, truncate, ~~add~~, modify, drop. comment, rename

DQL: select

DCL: Grant, revoke, TCL - commit, savepoint, Roll Back, set transaction, set, contraint