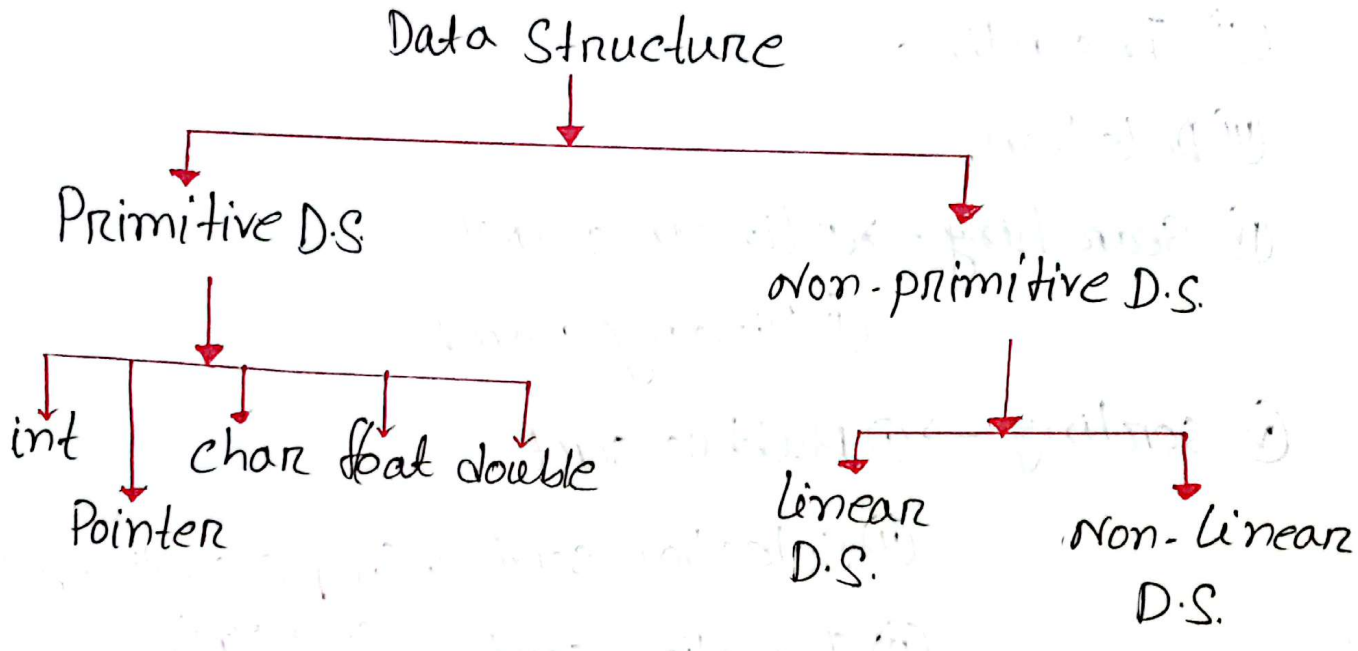
Data Structure & Algorithm

- Data Structure is about how data can be stored in different structures.
- An algorithm is a set of step-by-step instruction to solve a given problem or achieve a specific goal.



Linear Data Structure → one way

The arrangement of data in the sequential manner is known as linear data structure.

- Array, linked list, stacks, queue

Non-linear Data Structure → multiple paths

A non-linear structure is mainly used to represent data containing a hierarchical relationship between elements.

⊙ Graph, Tree & Hash Table.

⊙ operations on D.S. →

① Traversing

② Inserting

③ Deletion

④ Searching → ① linear search
② Binary search

⑤ sorting → ① Bubble sort
② Selection sort
③ Insertion sort
} Simple sorting Algo.
T.C. → $O(n^2)$
S.C. → $O(1)$

④ Quick sort
S.C → O(log n)

Efficient Sorting
T.C → O(n log n)
S.C → O(n)

⑤ Counting sort
⑥ Radix sort
⑦ Merge sort ← ⑧ Heap sort

⑨ Bucket sort } Linear Time Sort
S.C → O(1)

(VI) Merging.

⊙ What is Algorithms?

⇒ An algorithm is a process or a set of rules required to perform calculations or some other problem solving operations especially by a computer.

It is just a solution of problem, it can be flowchart or pseudocode.

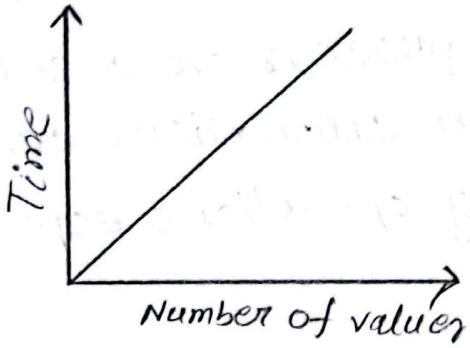
Approaches in Algorithm:

(I) Brute Force Algorithm → The general logic structure is applied to design an algorithm. Simple problem where efficiency is not a concern.

(II) Divide and Conquer → Break a problem into smaller subproblems, solve them recursively and combine the results.

(III) Greedy Algorithm → It states that a locally optimal choice at each step leads to a global optimal solution. Huffman Coding, Prim's (MST) Algo, Dijkstra's (SP)

⊙ The time complexity of an algorithm is the amount of time required to complete the execution.



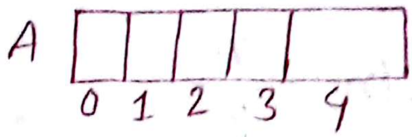
One operation

$O(1) \rightarrow 1 \longrightarrow$ Constant

one operation in an algorithm can be understood as something we do in each iteration of the algorithm, or for each piece of data that takes constant time.

(i)

```
sum(A,n)
{   S=0;                      ———→ 1
    for(i=0; i<n; i++)        ———→ n+1
    {
        S=S+A[i];             ———→ n
    }
    return S;                 ——— 1
}
```



i=0 T
i=1 T
i=2 T
i=3 T
i=4 T
i=5 F

} 6 iteration

time → $f(n) = 1 + n + 1 + n + 1$

$= 2n + 3$

degree → $O(n)$

(ii)

```
Add(A,B,n)
{ for(i=0; i<n; i++)              ———→ n+1
  {
    for(j=0; j<n; j++)            ———→ n×(n+1)
    {
      C[i,j] = A[i,j] + B[i,j];   ———→ n×n
    }
  }
}
```

time → $f(n) = 2n^2 + 2n + 1$

time comp → $O(n^2)$

for(i=1; i<n; i=i+2)

$f(n) = \frac{n}{2}$

for(i=1; i<n; i=i+20)

$f(n) = \frac{n}{20}$

Runtime function

(I) 1 → Constant runtime

(II) log N → logarithmic (cutting a big problem by few little problem & then solve)

(III) N → Linear

(IV) N log N

(V) N^2 (quadratic)

(VI) N^3 (cubic)

(VII) N^k (Polynomial)

(VIII) 2^N (exponential)

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

Asymptotic Notation

	meaning	time
O	big-oh → upper bound	(Describes worst case)
Ω	big-omega → Lower bound	(Describes best case)
Θ	~~big~~ theta → Tight bound	(average or exact behavior)

Big-oh

The function $f(n) = O(g(n))$ if $\exists$ +ve constant c_1, c_2 and n_0

such that $f(n) \leq c * g(n) \; \forall \; n \geq n_0$

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

$2n+3 \leq 10n$

$f(n) \quad c \quad g(n)$

$\therefore f(n) = O(n)$

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

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

lower bound — Aveg. bound — upper bound

Omega

The function $f(n) = \Omega(g(n))$

such that $f(n) \geq c * g(n)$

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

$2n+3 \geq 1 \times n$

$f(n) \quad c \cdot g(n)$

$f(n) = \Omega(n)$

$f(n) = \Omega(\log n)$

Theta

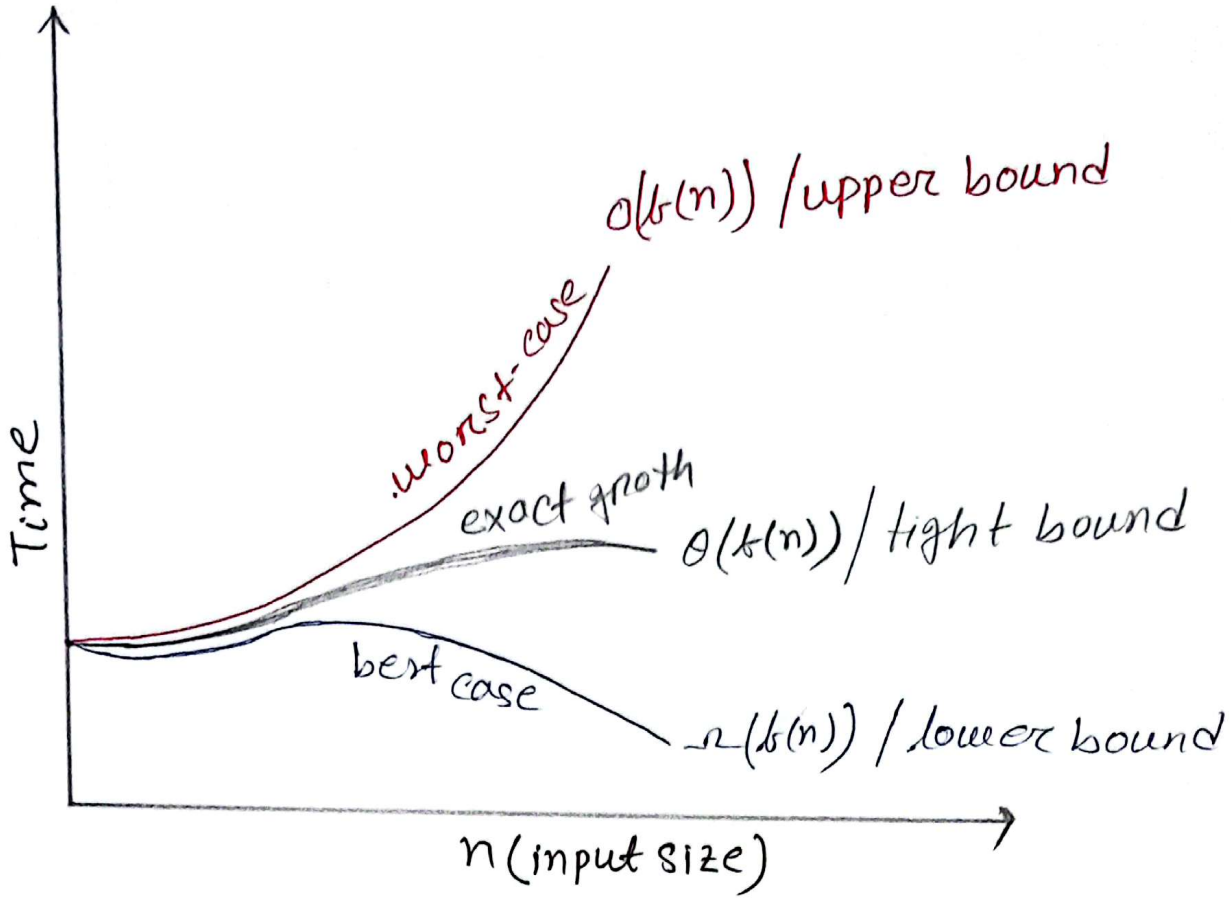
The function $f(n) = \theta(g(n))$

$f(n) = 2n+3$

$f(n) = \theta(n)$

$1 \times n \leq 2n+3 < 5 * n$

$c_1 g(n) \quad f(n) \quad c_2 g(n)$



- big-oh → $O(b(n))$ → maximum amount of run-time
- big-omega → $\Omega(b(n))$ → minimum amount of run time
- big-theta → $\Theta(b(n))$ → exact growth rate of an algorithm