-: TIME COMPLEXITY :-

Sort	Time		
	Average	Best	worst
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Modified Bubble Sort	$O(n^2)$	$O(n)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n^2)$	$O(n)$	$O(n^2)$
Heap Sort	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$
merge Sort	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$
Quick Sort	$O(n\log(n))$	$O(n\log(n))$	$O(n^2)$.

Operation	1D Array complexity	Singly linked list complexity
Insert at beginning	$O(N)$	$O(1)$
Insert at End	$O(1)$	$O(N)$
Insert at middle	$O(N)$	$O(N)$
Delete at Beginning	$O(N)$	$O(1)$
Delete at End	$O(1)$	$O(N)$
Delete at middle	$O(N)$	$O(N)$
Search	$O(N)$ linear Search $O(\log n)$ Binary Search	$O(N)$
Indexing	$O(1)$	$O(N)$

(i) Reverse the singly linked list – $O(n)$
(ii) Reverse the doubly linked list – $O(n)$
(iii) Reverse the circular linked list – $O(n)$
(iv) Reverse a doubly circular linked list – $O(n)$
(v) Reverse an array – $O(n)$.

Data Structure	Access	Search	Insertion	Delete
Array	O(1)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)
Queue	O(n)	O(n)	O(1)	O(1)
SLL	O(n)	O(n)	Begin: O(1) End: O(n)	Begin: O(1) End: O(n)
DLL	O(n)	O(n)	Begin: O(1) End: O(n)	Begin: O(1) End: O(n)
BST	O(n)	O(n)	O(n)	O(n)
B-TREE	O(log(n))	O(log(n))	O(log(n))	O(log(n))
AVL-TREE	O(log(n))	O(log(n))	O(log(n))	O(log(n))

Algorithm	Average	Best	worst
linear Search	O(n)	O(1)	O(n)
Binary Search	O(logn)	O(1)	O(logn)
Bucket Sort	O(n+K)	O(n+K)	$O(n^2)$
Radix Sort	O(nK)	O(nK)	O(n+K)
Tim Sort	O(nlogn)	O(n)	O(nlogn)
Shell Sort	$O((nlogn)^2)$	O(n)	$O((nlogn)^2)$
counting Sort	O(n+K)	O(n+K)	O(n+K)
Randomized Quick Sort	O(nlogn)	O(nlogn)	O(nlogn)

Scanned with CamScanner

Djikstra	$O(V^2)$ matrix Representation $O(E\log V) = O(V\log V)$ adjacency list representation (Fibonacci Heap)
Bellman Ford	$O(VE)$
Floyd warshall	$O(V^3)$ There are three nested loops that run through the vertices of the graph
Kruskal's	$O(E\log E) = O(E\log V)$ → sparse Graphs.
prim's	$O(E\log V)$ → Dense Graphs prim's algorithm can be improved using Fibonacci Heaps to $O(E+\log V)$.
Job Sequencing	1. sort Job according to decreasing order of deadline $= O(n\log n)$ 2. for each Job find slot in array of size $n = O(n^2)$ total time $= O(n\log n) + O(n^2) = O(n^2)$.
ASP	$O(n\log n)$, $O(n)$ (I/p Always sorted)
Huffman	$O(n\log n)$
mcm	Time complexity: $O(n^3)$, Auxiliary space: $O(n^2)$
LCS	$O(mn)$.

SRP (COMPLEXITIES)

Algorithm	Time complexity
Breadth First Traversal for a Graph	$O(V+E)$ for adjacency list representation. $O(V^2)$ for adjacency matrix representation.
Deapth First Traversal for a Graph	$O(V+E)$ for adjacency list representation $O(V^2)$ for adjacency matrix representation.
Dijkstra's Shortest path Algorithm	Adjacency matrix – $O(V^2)$ Adjacency list – $O(E\log V)$
Topological sorting:- Shortest path in Directed Acyclic Graph	$O(V+E)$.

Note:- $O(\log\log\log n) < O(\log\log n) < O(\log n) < O(n\log n) < O(n^\wedge P) < O(n^\wedge \log n) < O(2^\wedge n) <$ O(n!

Commonly used complexity:-

for(i=0; i<n; i++) $\longrightarrow O(n)$

for(i=0; i<n; i=i+2) $\longrightarrow O(n)$

for(i=n; i>1; i--) $\longrightarrow O(n)$

for(i=1; i<n; i=i*2) $\longrightarrow O(\log_2 n)$

for(i=1; i<n; i=i*3) $\longrightarrow O(\log_3 n)$

for(i=n; i>1; i=i/2) $\longrightarrow O(\log_2 n)$

for(i=n; i>1; i=i/3) $\longrightarrow O(\log_3 n)$.

In-place sorting:-
Ex:- Bubble sort, Insertion & Selection

Not-in-place sorting:-
Ex:- merge sort.

Stable sorting:-
Ex:- Insertion, Bubble & merge sort

Adaptive sorting algorithms:-
1. Bubble sort
2. Insertion sort
3. Quick sort

Non-adaptive sorting algorithms:-
1. Selection sort
2. merge sort
3. Heap sort

Master's Theorem:-

$$T(n) = aT(n/b) + \theta(n^K \log^P n).$$

we compare 'a' with 'b^K' and then following cases-

case-1:-

If $a > b^K$, then $T(n) = \theta(n^{\log_b^a})$

case-2:-

If $a = b^K$ then,

If $P < -1$, then $T(n) = \theta(n^{\log_b^a})$

If $P = -1$, then $T(n) = \theta(n^{\log_b^a} \cdot \log^2 n)$

If $P > -1$, then $T(n) = \theta(n^{\log_b^a} \cdot \log^{P+1} n)$.

case-3:-

If $a < b^K$ then,

If $P < 0$, then $T(n) = O(n^K)$

If $P >= 0$, then $T(n) = O(n^K \log^P n)$.