time cost	Insertion sort			Selection sort			Bubble sort		
	compare	move	total cost	compare	move	total cost	compare	move	total cost
Best	θ(n)	θ(n)	θ(n)	θ(n^2)	θ(n)	θ(n^2)	θ(n)	0	θ(n)
Average	θ(n^2)	θ(n^2)	θ(n^2)	θ(n^2)	θ(n)	θ(n^2)	θ(n^2)	θ(n^2)	θ(n^2)
Worst	θ(n^2)	θ(n^2)	θ(n^2)	θ(n^2)	θ(n)	θ(n^2)	θ(n^2)	θ(n^2)	θ(n^2)

SortAlgorithm	**In-place**	**stable**	**Input-sensitive**	**average**	**Worst-case**	**Best-case**	**Space**
Selection Sort	√	×	×	O(n^2)	O(n^2)	O(n^2)	O(1)
Bubble Sort	√	√	√	O(n^2)	O(n^2)	O(n)	O(1)
Insertion Sort	√	√	√	O(n^2)	O(n^2)	O(n)	O(1)
Shell Sort	√	×	√	O($n\sqrt{n}$)	O(n^2)	O(n)	O(1)
Merge Sort	×	√	×	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Quick Sort	×	×	√	O(nlogn)	O(n^2)	O(nlogn)	O(logn)
Heap Sort	√	×	×	O(nlogn)	O(nlogn)	O(nlogn)	O(1)
Counting Sort	×	√	×	O(n)	O(n)	O(n)	O(n)

SearchAlgorithm	average	Worst-case	Best-case
BruteForceStringMatching	O(n)	O(mn)	
Exhaustive Search			
Depth-First Traversal	$\theta(\|V\|^2)$ $\theta(\|V\| + \|E\|)$		
Breadth-First Traversal	$\theta(\|V\|^2)$ $\theta(\|V\| + \|E\|)$		
Binary Search	θ(log n)	θ(log(n+1))	
Quick Select (kth Smallest)	O(n)	$O(n^2)$	
Interpolation Search	O(log log n)		
Bottom-Up Heap Creation	O(n)		
ejection from a Heap	O(log n)		
Deletion, insertion and search of an AVL tree	θ(log n)	θ(log n)	
Deletion, insertion and search of a 2-3 tree	θ(log n)	θ(log n)	
Horspool's String Search	θ(n)	O(mn)	
Knapsack Problem	θ(nW).		
Warshall's Algorithm	$\theta(n^3)$	$\theta(n^3)$	$\theta(n^3)$
Floyd's Algorithm	$\theta(n^3)$	$\theta(n^3)$	$\theta(n^3)$
Prim's Algorithm	$O(\|V\| \cdot \|E\|)$ $O(\|E\| \log \|V\|)$.		
Dijkstra's algorithm	$O(\|V\| \cdot \|E\|)$ $O(\|E\| \log \|V\|)$.		

The time complexity of key-comparison based sorting has been proven to be **Ω (n log n).**

	worst access	worst search	worst insertion	worst deletion	average access	average search	average insertion	average deletion	best access	best search	best insertion	best deletion	space
unsorted array	1	n	1	1	1	n	1	1	1	1	1	1	n
sorted array	1	n	n	n	1	n	n	n	1	1	n(find position)	n(find position)	n
linked list	n	n	n	n	n	n	n	n	1	1	1	1	n
stack	n	n	1	1	n	n	1	1	1	1	1	1	n
hash table	-	n	n	n	-	1	1	1	-	1	1	1	n
BST	n(stick)	n(stick)	n	n	log n	log n	log n	log n	1	1	log n	log n(find bottom)+1 Or 1(find top)+log n(find to swap)	n
AVL tree	log n	log n	log n	log n	log n	log n	log n	log n	1	1	log n	log n(find bottom)+1 Or 1(find top)+log n(find to swap)	n

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10^1	3	10^1	$3 \cdot 10^1$	10^2	10^3	10^3	$4 \cdot 10^6$
10^2	7	10^2	$7 \cdot 10^2$	10^4	10^6	10^{30}	$9 \cdot 10^{157}$
10^3	10	10^3	$1 \cdot 10^4$	10^6	10^9	—	—

Hash table: m: large enough to allow efficient operations, without taking up excessive memory, prime.

load factor α = n/m

Separate Chaining:

Number of probes in successful search $\approx$ 1 + α /2.

Number of probes in unsuccessful search $\approx$ α

+1.reduces the number of comparisons,

+2.Good in a dynamic environment,

+3.The chains can be ordered,

-1. uses extra storage for links.

linear probing:

Successful search: 1/2+1/2(1- α)

Unsuccessful: 1/2+1/2(1- α $)^2$

+1. Space-efficient.

-1. Worst-case performance miserable
-2. Clustering is a major problem
-3. Deletion is almost impossible.

To **Avoid**:
keep track of the load factor and to rehash when it reaches, say, 0.9.
Rehashing means allocating a larger hash table, and will introduce long delays at unpredictable times

Some **drawbacks**:
1.If an application calls for traversal of all items in sorted order, a hash table is no good.
2.Also, unless we use separate chaining, deletion is virtually impossible.
3.It may be hard to predict the volume of data, and rehashing is an expensive “stop-the-world” operation.

Warshall:
computes the transitive closure (“AND”, reachable or not, 1 or 0),
matrix for directed graph,
ideal for dense graphs (DFS for sparse graphs)
$O(|V|^3)$.

Floyd:
Shortest-Paths for All-Pairs (“+”),
matrix for directed and undirected graphs.
$O(|V|^3)$.

Prim:
Minimum Spanning Trees
for undirected weighted graphs
Using weighted matrix: $O(|V| \cdot |E|)$
Using adjacency lists: $O(|E| \log |V|)$.

Dijkstra:
shortest-path from a fixed start node
for directed or undirected weighted graphs
Using weighted matrix: $O(|V| \cdot |E|)$
Using adjacency lists: $O(|E| \log |V|)$.