InsertionSort: left shift until correct -> O(n^2)$

Mergesort: two sub-array to sort(divide&conquer) -> O(n)$

BinarySearch: start search from the middle, then middle again -> $T(n) \leq T(n/2) + O(1) = O(\log n)$

Quicksort: use pivot to sort, move pivot in the middle at end. -> Best: $T(n) = 2T(n/2) + \Theta(n) -> O(n \log n)$, Worst: $O(n^2)$

BFS: G=(V,E): G for graph, V for nodes, E for edges. Search all nodes in a layer before going to the next. -> $O(n + E)$, n for V

DFS: Search according to depth first, touch the button then traceback. -> $O(n + E)$

Dijkstra: greedy alg for calculate the shortest path using graph. -> $T(n) = O(n \log n + m \log n) = O(m \log n)$, $m = V + E$, $n = V$

Huffman: compress document using Binary Tree, most frequent word put in forward to get less bit. -> $O(n \log n)$

SegmentedLeastSquares: fit a curve containing several pieces of line (dynamic programming). -> $O(n^2)$

Sequence Alignment: find the most similarity of two sentence with least error(dynamic programming). -> Two for loops: $O(nm)$

Upper: Big O $\leq$, **Lower**: Big Ω $\geq$, **Tight**: Big Θ $=$, little o $<$, little w $>$

$\log n < n < n \log n < n^2 < 2^n < 3^n < n^n$

Simple path for distinct nodes; **Simple cycle** for distinct paths; **Strongly Connect Components(SCC)** for bidirectional node.

Recursive Fibonacci: $T(n) = O(1) + T(n-1) + T(n-2) = \Omega(2^n/2)$

Non-recursive Fib: $T(n) = O(n)$

Fib mul add: $T(n) = O(\log n)$

BFS_CutNode: find node v between s and t which will destroy all s-t path if v is deleted. -> $O(n + E)$

Graph_isOdd: find if a directed graph G has an odd-length cycle. -> $O(V^2)$

WaterPouringBFS: two bottles with capacity X and Y with initial water x and y. Find possible or not that A liters water should in any bottle. -> $O(n^2)$

2SAT: find if there has the specific node and its negation in a same SCC, if yes, it is not Boolean satisfiable. -> $O(n + m)$, for n numbers of variables and m number of clauses.

Recursive: call function repeatly; **Iterative**: Use (for) loop

Master Theorem: For a $T(n) = aT(n/b) + f(n)$, where $f(n) = \Theta/O/\Omega(n^c)$. If c<2, T(n)=θ(n^2); if c=2, T(n)=θ(n^2 * log(n)); if c>2, T(n) = θ(f(n))