**Insertion Sort**: 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 ->

**Quicksort**: use pivot to sort, move pivot in the middle at end. -> Best: , Worst:

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

**DFS**: Search according to depth first, touch the button then traceback. ->

**Dijkstra**: greedy alg for calculate the shortest path using graph. -> ,

**Huffman**: compress document using Binary Tree, most frequent word put in forward to get less bit. ->

**SegmentedLeastSquares**: fit a curve containing several pieces of line (dynamic programming). ->

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

**Insertion Sort**: 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 ->

**Quicksort**: use pivot to sort, move pivot in the middle at end. -> Best: , Worst:

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

**DFS**: Search according to depth first, touch the button then traceback. ->

**Dijkstra**: greedy alg for calculate the shortest path using graph. -> ,

**Huffman**: compress document using Binary Tree, most frequent word put in forward to get less bit. ->

**SegmentedLeastSquares**: fit a curve containing several pieces of line (dynamic programming). ->

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

**Insertion Sort**: 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 ->

**Quicksort**: use pivot to sort, move pivot in the middle at end. -> Best: , Worst:

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

**DFS**: Search according to depth first, touch the button then traceback. ->

**Dijkstra**: greedy alg for calculate the shortest path using graph. -> ,

**Huffman**: compress document using Binary Tree, most frequent word put in forward to get less bit. ->

**SegmentedLeastSquares**: fit a curve containing several pieces of line (dynamic programming). ->

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

**Insertion Sort**: 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 ->

**Quicksort**: use pivot to sort, move pivot in the middle at end. -> Best: , Worst:

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

**DFS**: Search according to depth first, touch the button then traceback. ->

**Dijkstra**: greedy alg for calculate the shortest path using graph. -> ,

**Huffman**: compress document using Binary Tree, most frequent word put in forward to get less bit. ->

**SegmentedLeastSquares**: fit a curve containing several pieces of line (dynamic programming). ->

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

**Upper**: Big O , **Lower**: Big , **Tight**: Big , little o , little w

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

**Recursive Fibonacci**:

**Non-recursive Fib**:

**Fib mul add**:

**BFS\_CutNode**: find node v between s and t which will destroy all s-t path if v is deleted. ->

**Graph\_isOdd**: find if a directed graph G has an odd-length cycle. ->

**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. ->

**Path\_Num**: Find the number of path of two nodes in a graph: use topo\_order() find topology order of the node, then use two for loops to sum each topo order elements’ neighbor to output the path number. -> T(n) = O(V+E)

**Upper**: Big O , **Lower**: Big , **Tight**: Big , little o , little w

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

**Recursive Fibonacci**:

**Non-recursive Fib**:

**Fib mul add**:

**BFS\_CutNode**: find node v between s and t which will destroy all s-t path if v is deleted. ->

**Graph\_isOdd**: find if a directed graph G has an odd-length cycle. ->

**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. ->

**Path\_Num**: Find the number of path of two nodes in a graph: use topo\_order() find topology order of the node, then use two for loops to sum each topo order elements’ neighbor to output the path number. -> T(n) = O(V+E)

**Upper**: Big O , **Lower**: Big , **Tight**: Big , little o , little w

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

**Recursive Fibonacci**:

**Non-recursive Fib**:

**Fib mul add**:

**BFS\_CutNode**: find node v between s and t which will destroy all s-t path if v is deleted. ->

**Graph\_isOdd**: find if a directed graph G has an odd-length cycle. ->

**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. ->

**Path\_Num**: Find the number of path of two nodes in a graph: use topo\_order() find topology order of the node, then use two for loops to sum each topo order elements’ neighbor to output the path number. -> T(n) = O(V+E)

**Upper**: Big O , **Lower**: Big , **Tight**: Big , little o , little w

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

**Recursive Fibonacci**:

**Non-recursive Fib**:

**Fib mul add**:

**BFS\_CutNode**: find node v between s and t which will destroy all s-t path if v is deleted. ->

**Graph\_isOdd**: find if a directed graph G has an odd-length cycle. ->

**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. ->

**Path\_Num**: Find the number of path of two nodes in a graph: use topo\_order() find topology order of the node, then use two for loops to sum each topo order elements’ neighbor to output the path number. -> T(n) = O(V+E)