LEGEND

TIME Complexity VS. **SPACE** Complexity

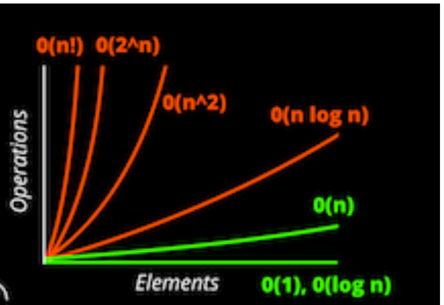
Good Fair Bad

Good Fair Bad

<BIG-O-CHEATSHEET>

</>

www.bigocheatsheet.com

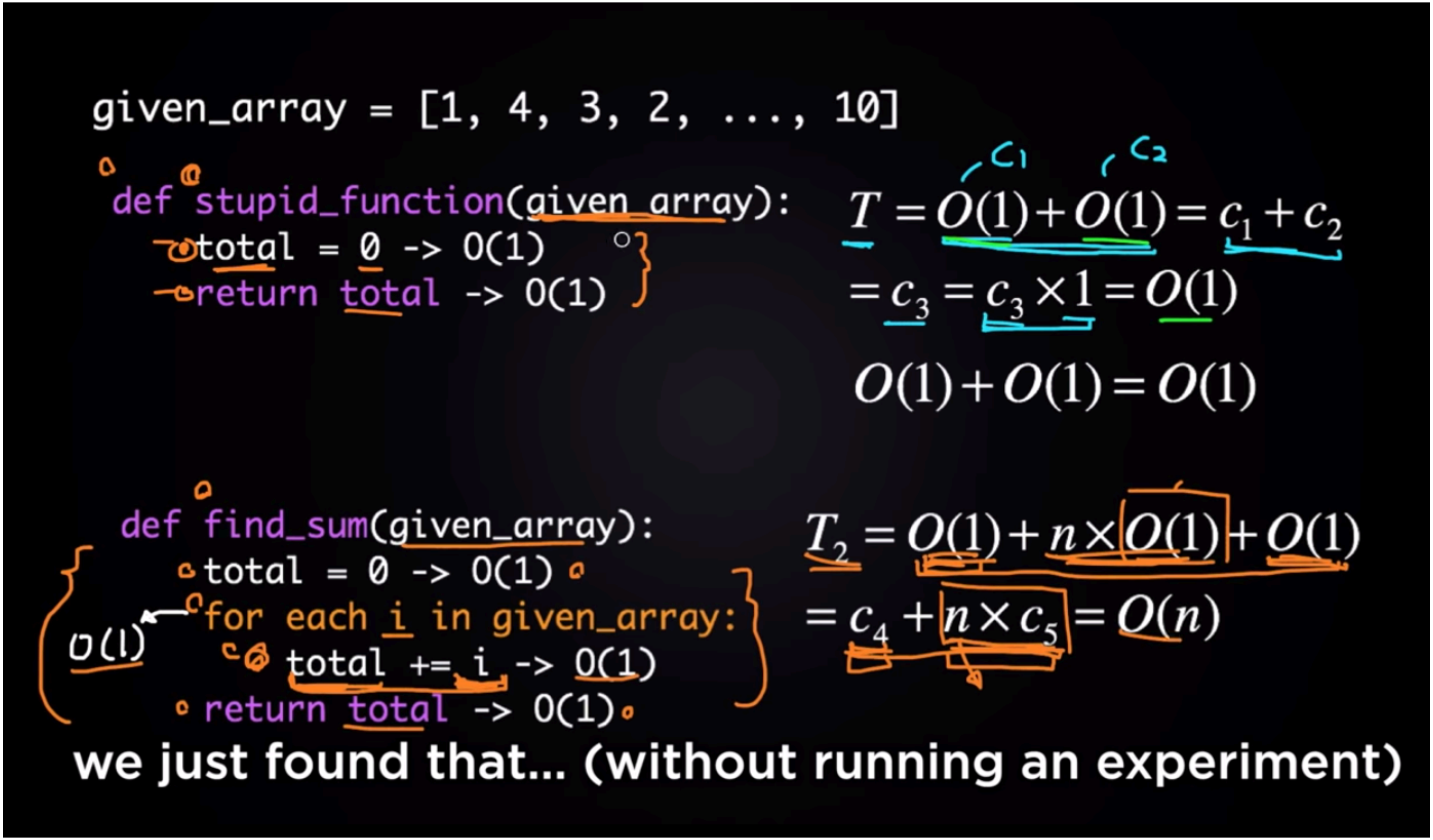


DATA STRUCTURE Operations

DATA Structure	TIME Complexity								SPACE Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

ARRAY SORTING Algorithms

ARRAY Algorithms	TIME Complexity			SPACE Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(1)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)t	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)



```
given_array = [1, 4, 3, 2, ..., 10]

def stupid_function(given_array):
    total = 0 -> O(1)
    return total -> O(1)
```

$$T = \overset{c_1}{O(1)} + \overset{c_2}{O(1)} = c_1 + c_2$$

$$= c_3 = c_3 \times 1 = O(1)$$

$$O(1) + O(1) = O(1)$$

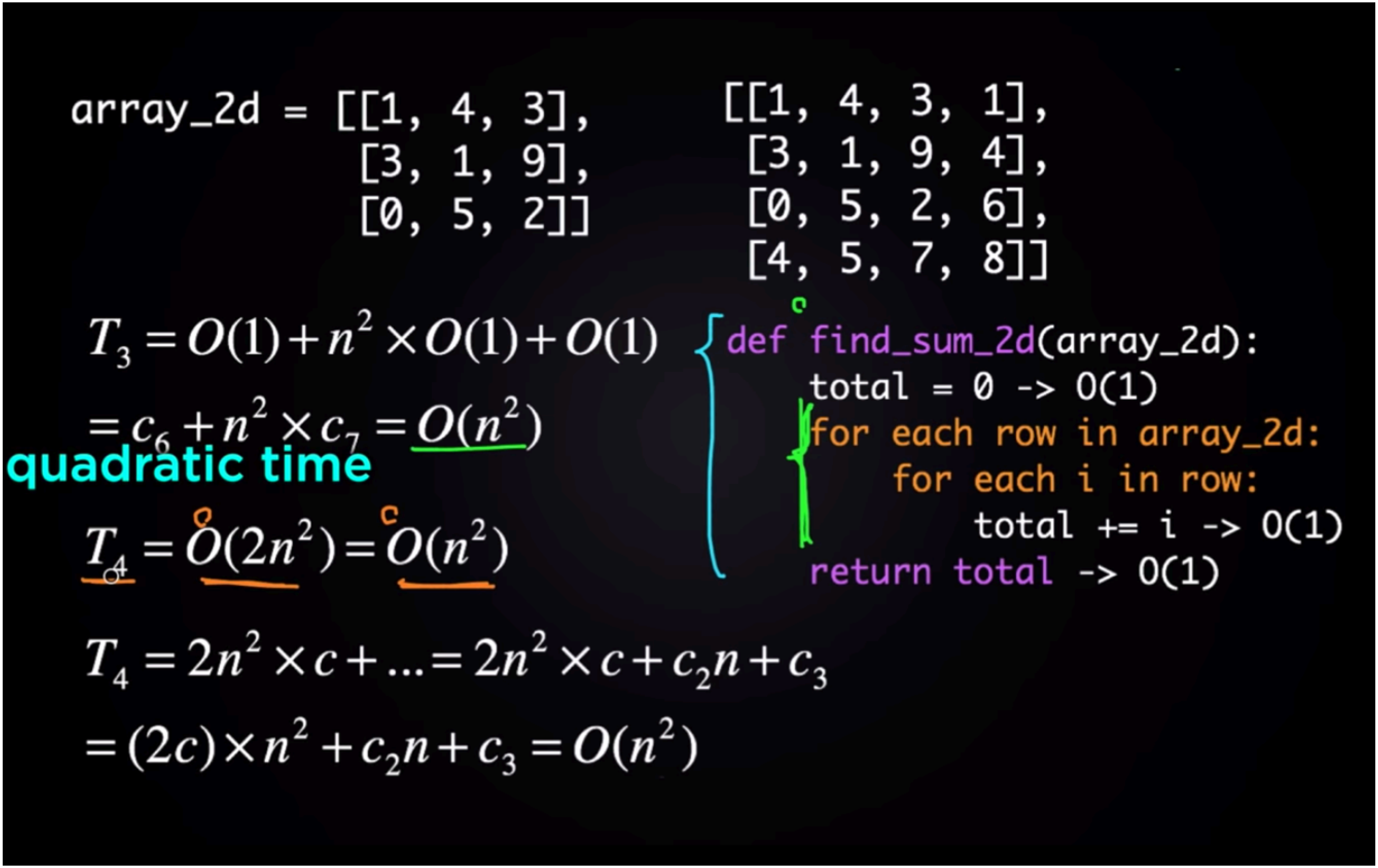
```
def find_sum(given_array):
    total = 0 -> O(1)
    for each i in given_array:
        total += i -> O(1)
    return total -> O(1)
```

O(1)

$$T_2 = O(1) + n \times \boxed{O(1)} + O(1)$$

$$= c_4 + \boxed{n \times c_5} = O(n)$$

we just found that... (without running an experiment)



```
array_2d = [[1, 4, 3],
            [3, 1, 9],
            [0, 5, 2]]
```

```
[[1, 4, 3, 1],
 [3, 1, 9, 4],
 [0, 5, 2, 6],
 [4, 5, 7, 8]]
```

$$T_3 = O(1) + n^2 \times O(1) + O(1)$$

$$= c_6 + n^2 \times c_7 = \underline{O(n^2)}$$

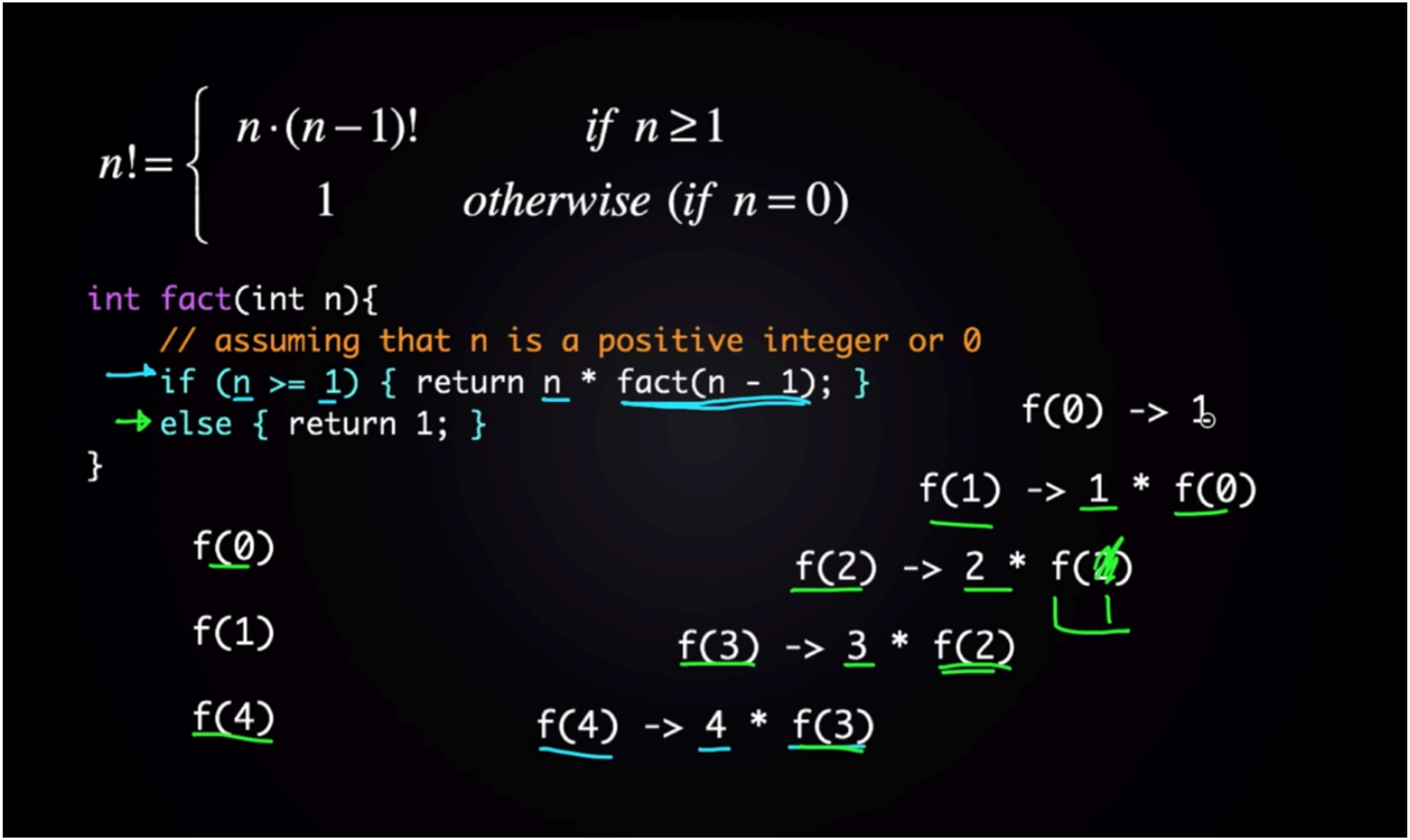
quadratic time

$$\underline{T_4} = \underline{O(2n^2)} = \underline{O(n^2)}$$

```
def find_sum_2d(array_2d):
    total = 0 -> O(1)
    for each row in array_2d:
        for each i in row:
            total += i -> O(1)
    return total -> O(1)
```

$$T_4 = 2n^2 \times c + \ldots = 2n^2 \times c + c_2 n + c_3$$

$$= (2c) \times n^2 + c_2 n + c_3 = O(n^2)$$



$$n! = \begin{cases} n \cdot (n-1)! & \textit{if } n \geq 1 \\ 1 & \textit{otherwise (if } n = 0) \end{cases}$$

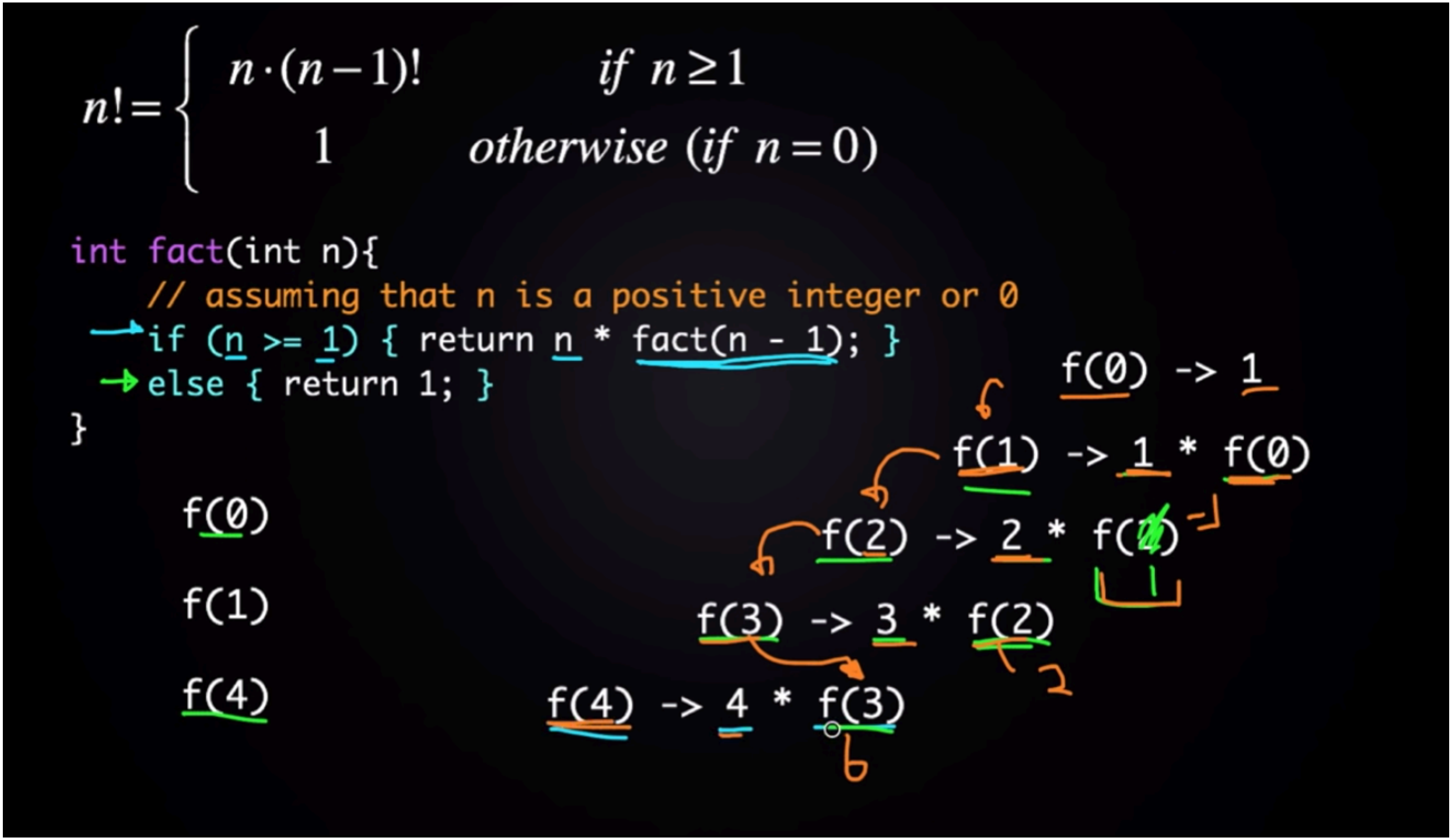
```
int fact(int n){
    // assuming that n is a positive integer or 0
   →if (n >= 1) { return n * fact(n - 1); }
   →else { return 1; }
}
```

```
      f(0)                                          f(0) -> 1

                                                f(1) -> 1 * f(0)

      f(1)                                  f(2) -> 2 * f(1)
                                                        1
                                      f(3) -> 3 * f(2)

      f(4)                      f(4) -> 4 * f(3)
```



$$n! = \begin{cases} n\cdot(n-1)! & \textit{if } n \geq 1 \\ 1 & \textit{otherwise } (\textit{if } n = 0) \end{cases}$$

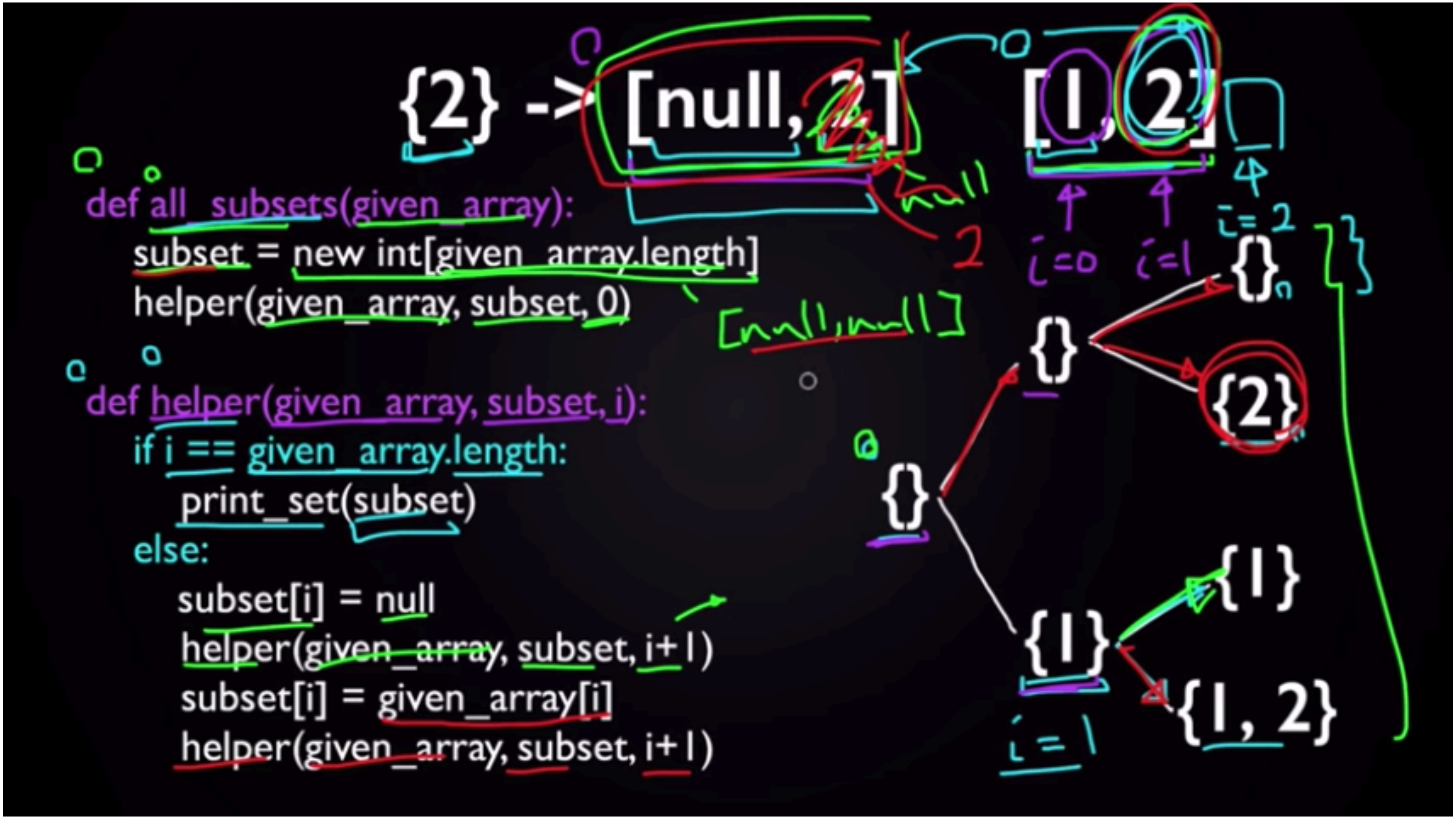
```
int fact(int n){
    // assuming that n is a positive integer or 0
    if (n >= 1) { return n * fact(n - 1); }
    else { return 1; }
}
```

f(0)

f(1)

f(4)

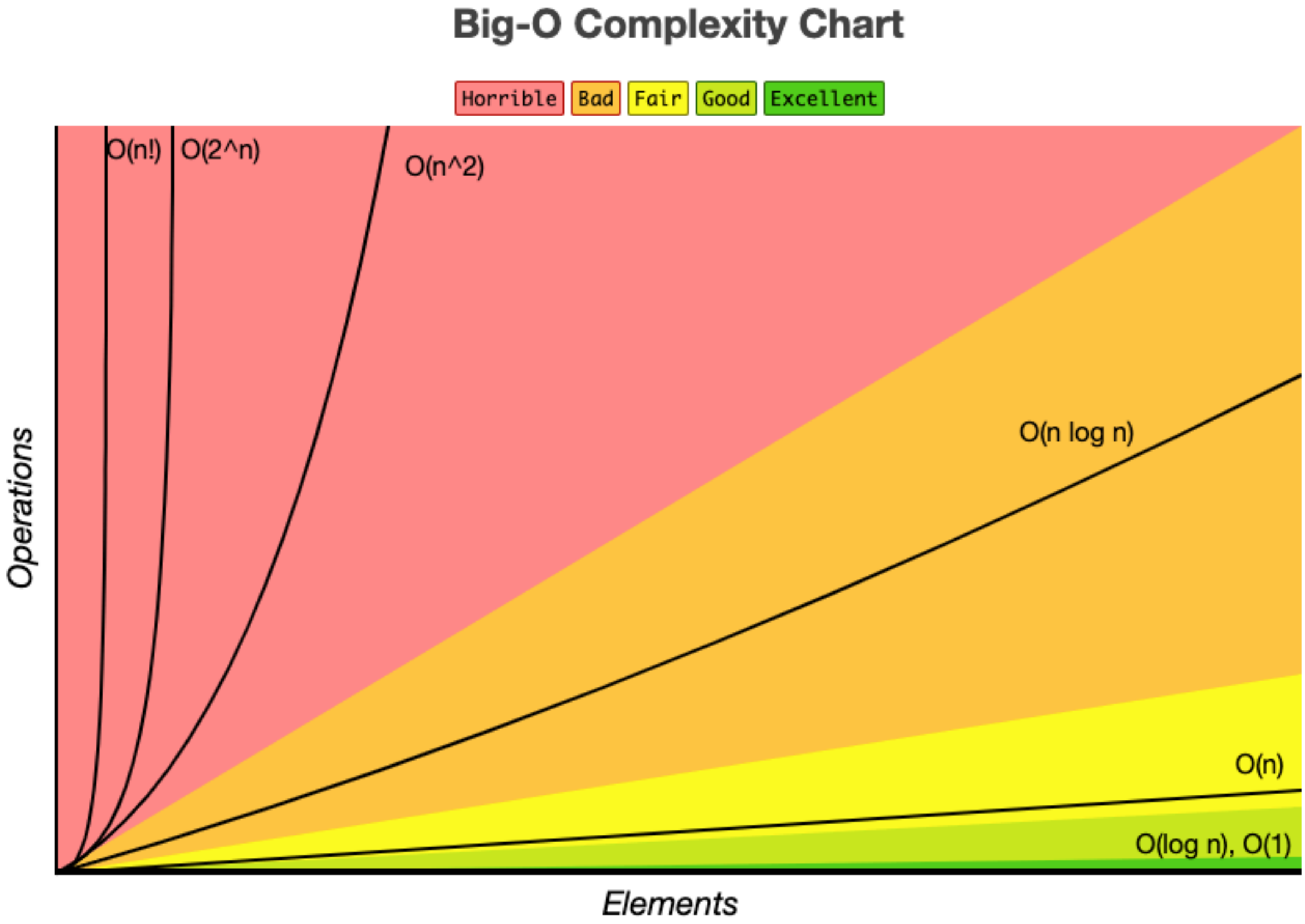
```
                                f(0) -> 1
                         f(1) -> 1 * f(0)
                   f(2) -> 2 * f(1)
            f(3) -> 3 * f(2)
     f(4) -> 4 * f(3)
```

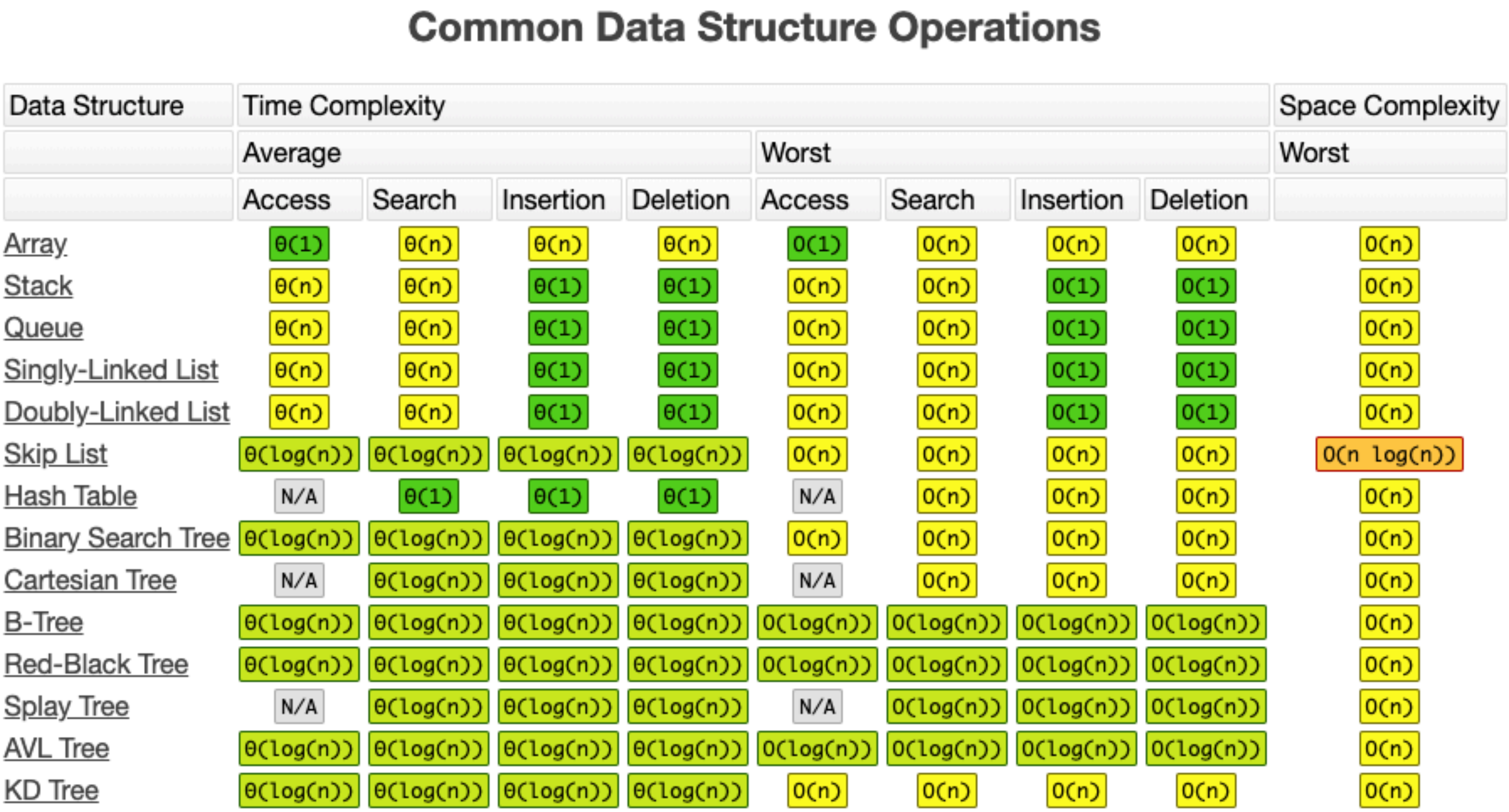


{2} -> [null, 2]   [1, 2]

```
def all_subsets(given_array):
    subset = new int[given_array.length]
    helper(given_array, subset, 0)

def helper(given_array, subset, i):
    if i == given_array.length:
        print_set(subset)
    else:
        subset[i] = null
        helper(given_array, subset, i+1)
        subset[i] = given_array[i]
        helper(given_array, subset, i+1)
```

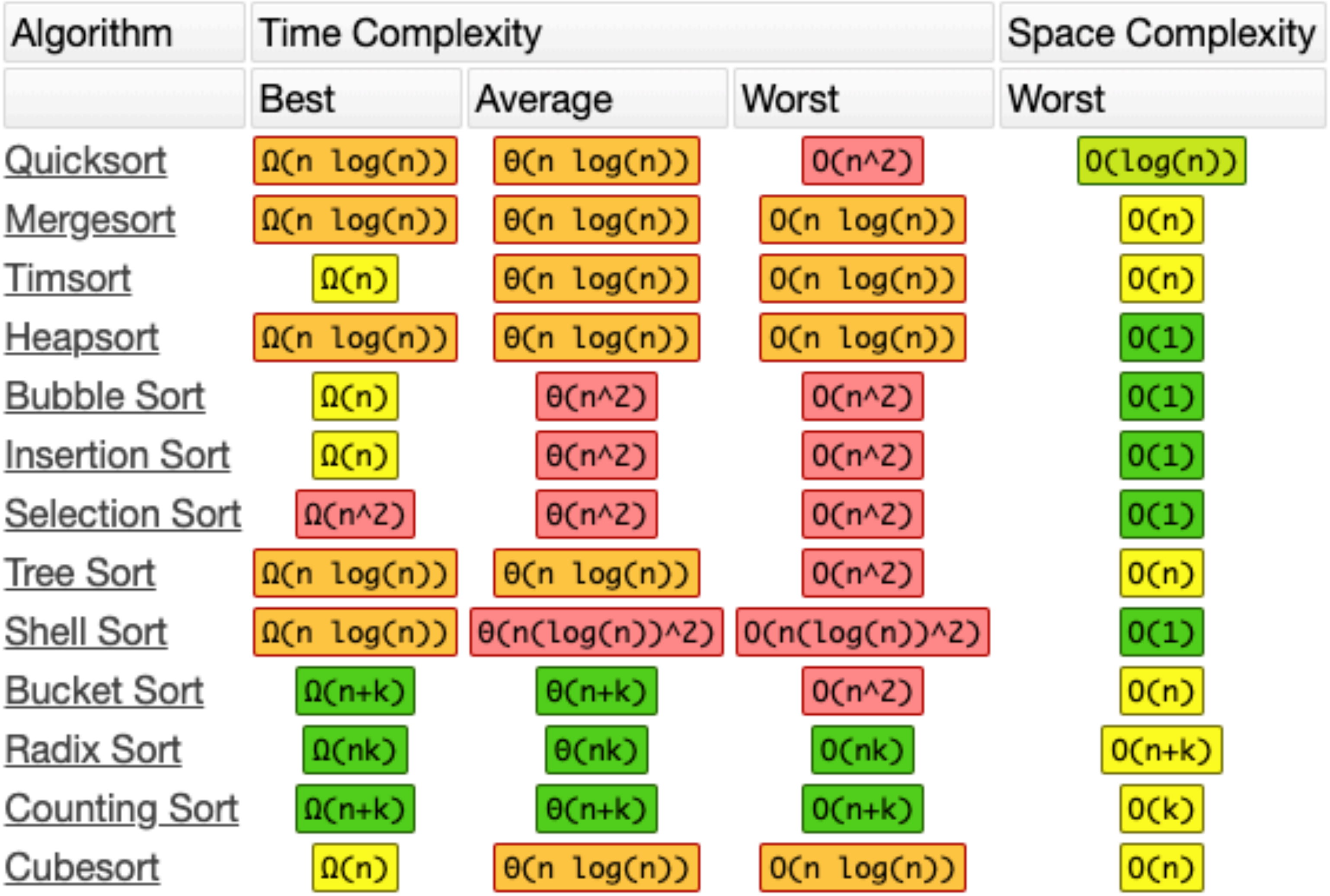

Big-O Complexity Chart
Horrible
Bad
Fair
Good
Excellent
O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n), O(1)
Operations
Elements



Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

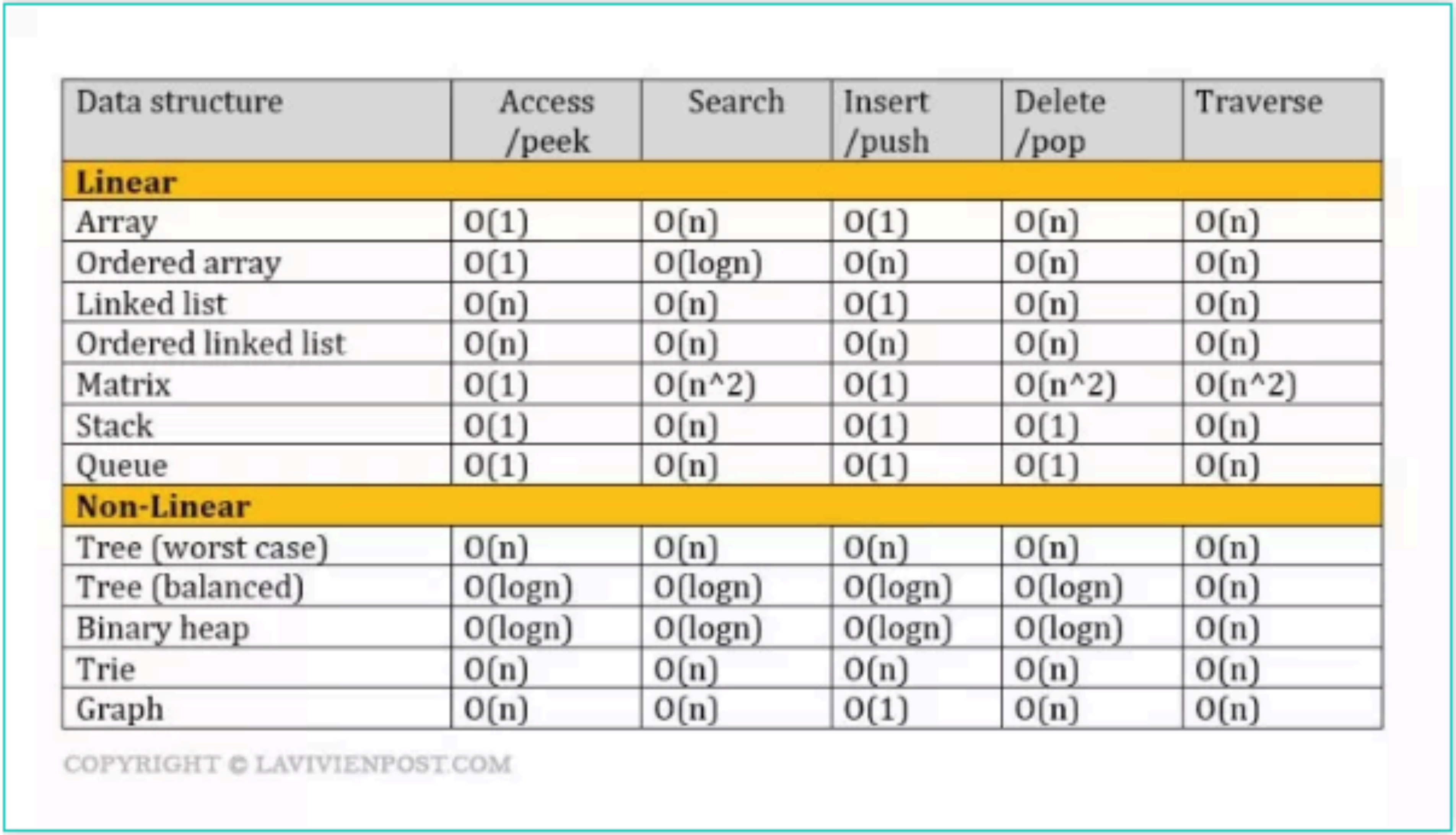
Array Sorting Algorithms



Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)

Big-O Notation of Data Structures

Complete list of Data Structure

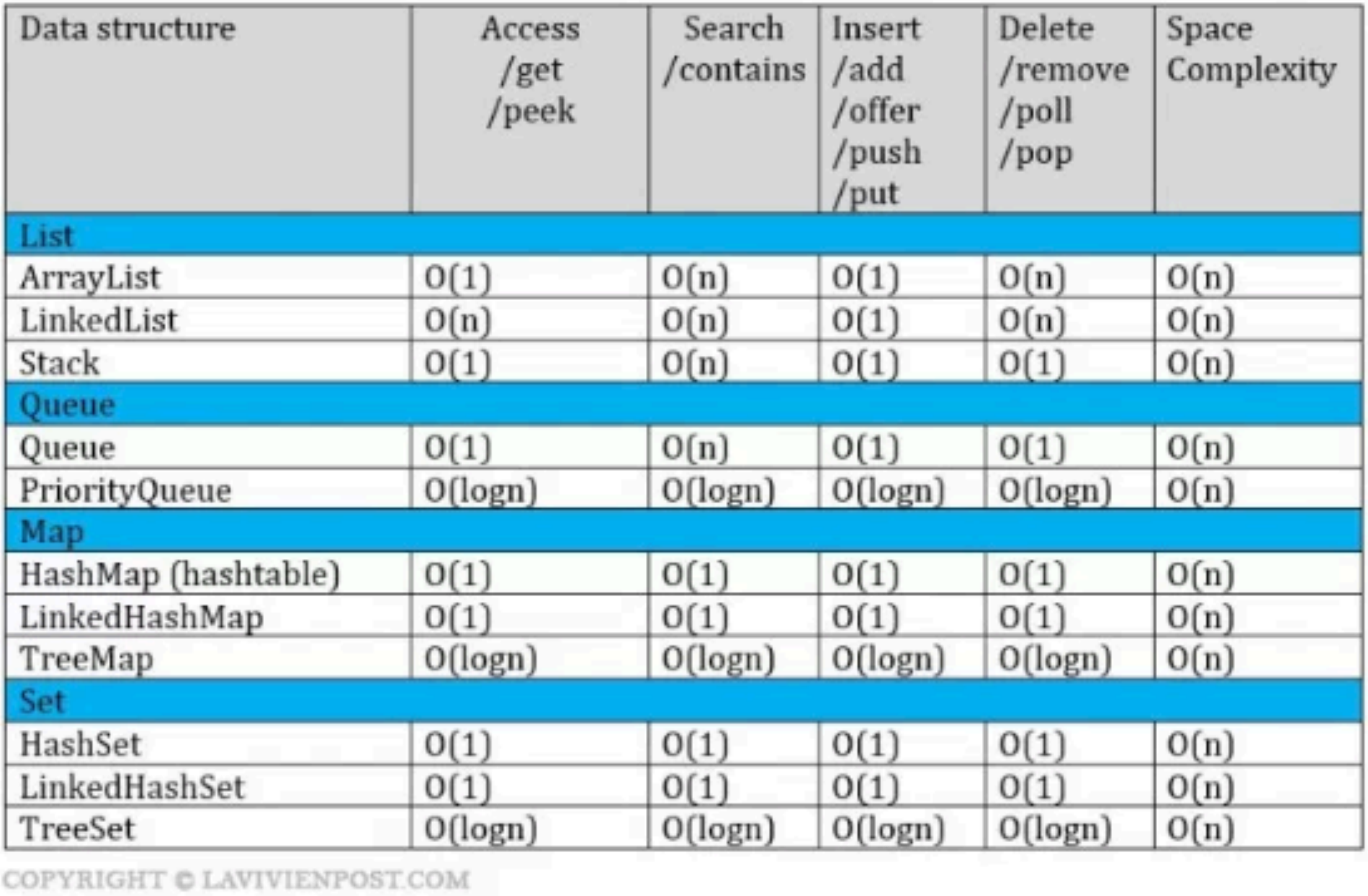


Data structure	Access /peek	Search	Insert /push	Delete /pop	Traverse
Linear					
Array	O(1)	O(n)	O(1)	O(n)	O(n)
Ordered array	O(1)	O(logn)	O(n)	O(n)	O(n)
Linked list	O(n)	O(n)	O(1)	O(n)	O(n)
Ordered linked list	O(n)	O(n)	O(n)	O(n)	O(n)
Matrix	O(1)	O(n^2)	O(1)	O(n^2)	O(n^2)
Stack	O(1)	O(n)	O(1)	O(1)	O(n)
Queue	O(1)	O(n)	O(1)	O(1)	O(n)
Non-Linear					
Tree (worst case)	O(n)	O(n)	O(n)	O(n)	O(n)
Tree (balanced)	O(logn)	O(logn)	O(logn)	O(logn)	O(n)
Binary heap	O(logn)	O(logn)	O(logn)	O(logn)	O(n)
Trie	O(n)	O(n)	O(n)	O(n)	O(n)
Graph	O(n)	O(n)	O(1)	O(n)	O(n)

COPYRIGHT © LAVIVIENPOST.COM

Big-O Notation of Java Collections

Complete list of Java Collections

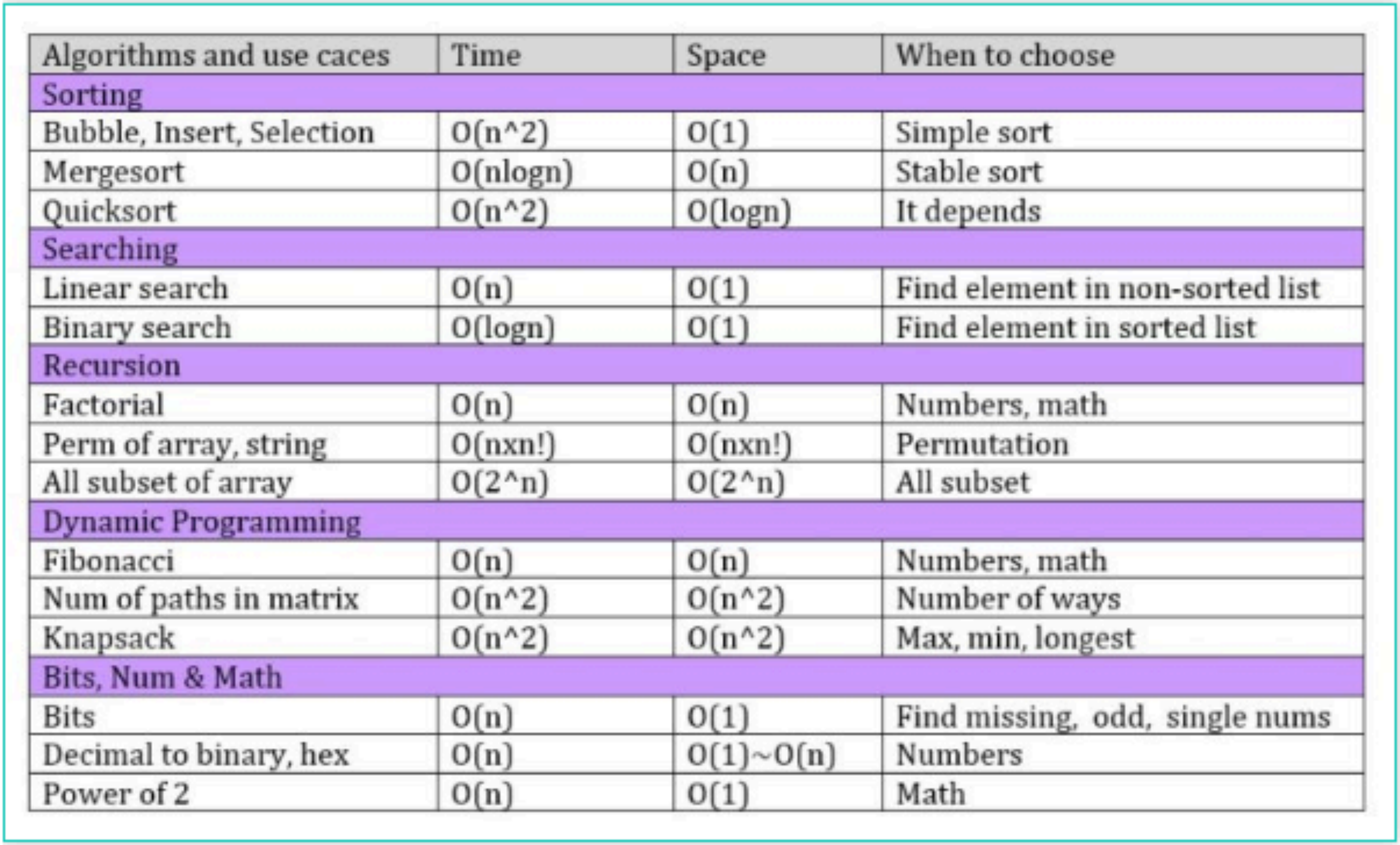


Data structure	Access /get /peek	Search /contains	Insert /add /offer /push /put	Delete /remove /poll /pop	Space Complexity
List					
ArrayList	O(1)	O(n)	O(1)	O(n)	O(n)
LinkedList	O(n)	O(n)	O(1)	O(n)	O(n)
Stack	O(1)	O(n)	O(1)	O(1)	O(n)
Queue					
Queue	O(1)	O(n)	O(1)	O(1)	O(n)
PriorityQueue	O(logn)	O(logn)	O(logn)	O(logn)	O(n)
Map					
HashMap (hashtable)	O(1)	O(1)	O(1)	O(1)	O(n)
LinkedHashMap	O(1)	O(1)	O(1)	O(1)	O(n)
TreeMap	O(logn)	O(logn)	O(logn)	O(logn)	O(n)
Set					
HashSet	O(1)	O(1)	O(1)	O(1)	O(n)
LinkedHashSet	O(1)	O(1)	O(1)	O(1)	O(n)
TreeSet	O(logn)	O(logn)	O(logn)	O(logn)	O(n)

COPYRIGHT © LAVIVIENPOST.COM

Big-O Notation of Algorithms

Complete list of Algorithms



Algorithms and use caces	Time	Space	When to choose
Sorting			
Bubble, Insert, Selection	O(n^2)	O(1)	Simple sort
Mergesort	O(nlogn)	O(n)	Stable sort
Quicksort	O(n^2)	O(logn)	It depends
Searching			
Linear search	O(n)	O(1)	Find element in non-sorted list
Binary search	O(logn)	O(1)	Find element in sorted list
Recursion			
Factorial	O(n)	O(n)	Numbers, math
Perm of array, string	O(nxn!)	O(nxn!)	Permutation
All subset of array	O(2^n)	O(2^n)	All subset
Dynamic Programming			
Fibonacci	O(n)	O(n)	Numbers, math
Num of paths in matrix	O(n^2)	O(n^2)	Number of ways
Knapsack	O(n^2)	O(n^2)	Max, min, longest
Bits, Num & Math			
Bits	O(n)	O(1)	Find missing, odd, single nums
Decimal to binary, hex	O(n)	O(1)~O(n)	Numbers
Power of 2	O(n)	O(1)	Math