Swipe >>>

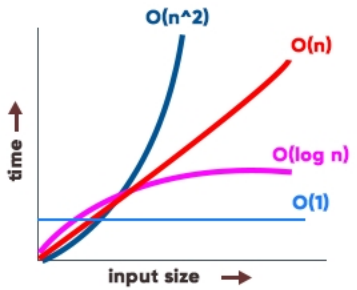
# Algorithm

# COMPUTER SCIENCE FUNDAMENTALS

## ALGORITHM #1 — Time & Space Complexities

Time and Space complexities describe the amount of time and memory an algorithm takes with respect to the amount of input.

- **O(1)** - Summing of two numbers
- **O(logn)** - Looking up people in a phone book
- **O(n)** - Finding a CD in a stack of CDs
- **O(n*logn)** - Cutting a randomly sized bread loaf into 8 pieces but by dividing them from middle each time and then rearranging them accoring to their size.
- **O(n^2)** - Asking every person his and all others health
- **O(2^n)** - Solving Tower of Hanoi problem with n discks




csmock

# COMPUTER SCIENCE FUNDAMENTALS

## ALGORITHM #2 Quick Sort

Quick sort picks an element as pivot and move the elements less than pivot to the left and greater to the right.

**Different versions of pivot logic :**

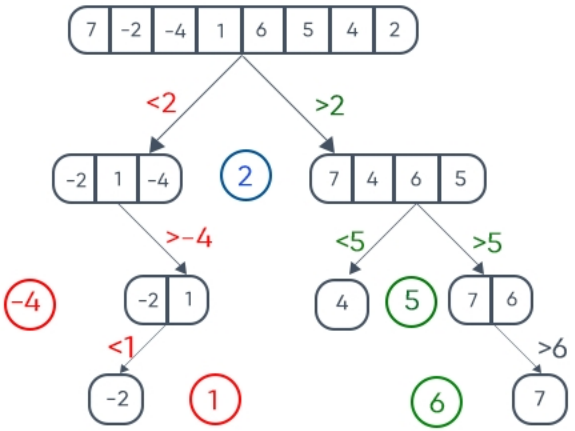
1. Pick First element
2. Pick Last element
3. Pick Random element
4. Pick Median element

**Features of Quick Sort :**

1. Divide-and-conquer algorithm
2. Not-stable
3. In-place

**Time Complexity** = O(n*logn)
**Space Complexity** = O(logn)
where, n is the size

**Last element pick implementation**




cs mock

# COMPUTER SCIENCE FUNDAMENTALS

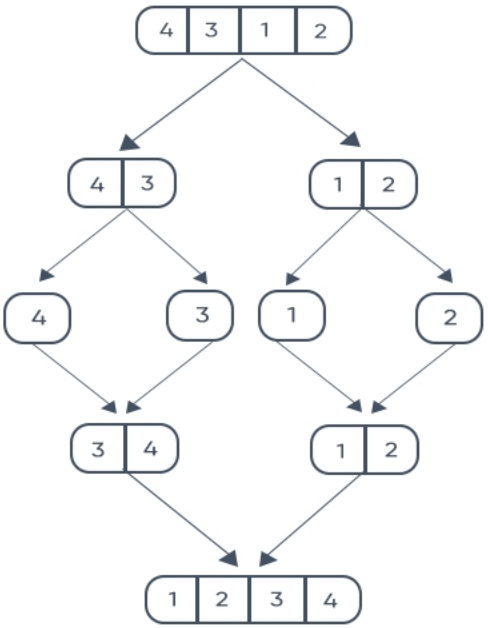
## ALGORITHM #3 Merge Sort

Merge sort divides the array into two halves recursively till the size becomes 1 and then merge the two sorted halves.

**Features of Merge Sort :**

1. Divide-and-conquer algorithm
2. Stable
3. Not in-place

**Time Complexity** = O(n*logn)
**Space Complexity** = O(n)
where, n is the size




cs mock

# COMPUTER SCIENCE FUNDAMENTALS

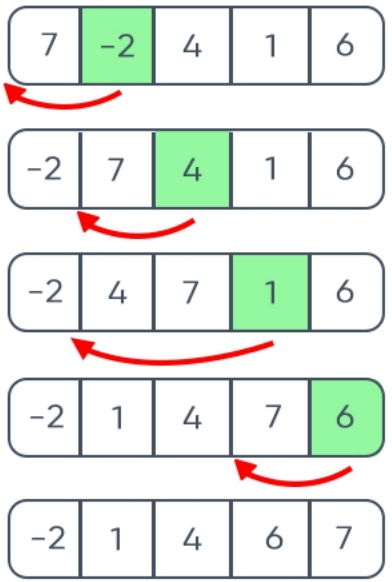
## ALGORITHM #4 Insertion Sort

Insertion sort places an unsorted element at its suitable place in each iteration.

**Features of Insertion Sort :**

1. Stable
2. In-place

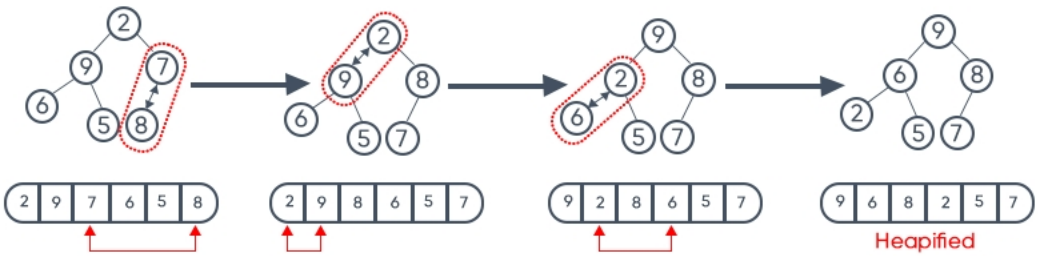
**Time Complexity** = $O(n^2)$
**Space Complexity** = $O(1)$
where, n is the size



# COMPUTER SCIENCE FUNDAMENTALS

## ALGORITHM #5 Heap Sort

Heap sort is a comparison based sorting technique where we first find the minimum/maximum element (Min/Max heap) and place at the beginning. We repeat the same process for remaining elements.




Max Heap representation

**Features of Heap Sort :**

1. Not Stable
2. In-place
3. Comparison based

**Time Complexity** = O(logn)

**Space Complexity** = O(1)
{No auxiliary space}

where, n is the size

# COMPUTER SCIENCE FUNDAMENTALS

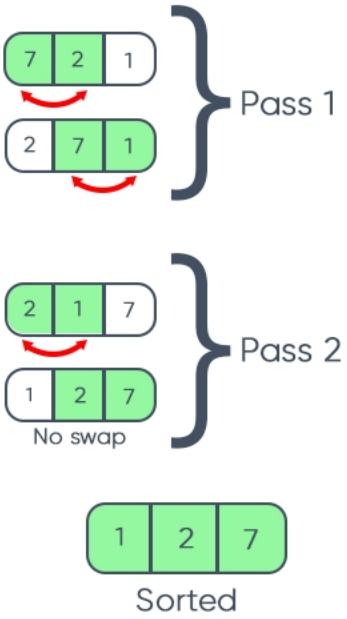
## ALGORITHM #6 Bubble Sort

Bubble sort compares adjacent elements and swaps them if they are not in order.

**Features of Bubble Sort :**

1. Stable
2. In-place

**Time Complexity** = $O(n^2)$
**Space Complexity** = $O(1)$
where, n is the size



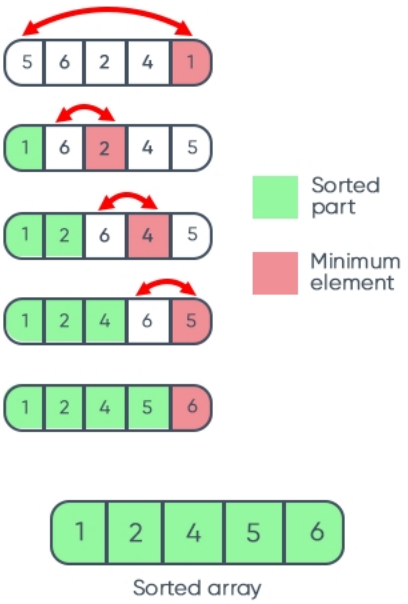
# COMPUTER SCIENCE FUNDAMENTALS

## ALGORITHM #7 Selection Sort

Selection Sort recursively finds the minimum/maximum element from unsorted part of array and swaps it at the beginning.

**Features of Selection Sort :**

1. Not Stable
2. In-place




**Time Complexity** = $O(n^2)$
**Space Complexity** = $O(1)$
where, n is the size

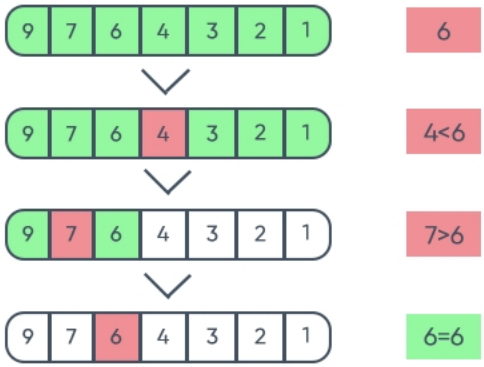
# COMPUTER SCIENCE FUNDAMENTALS

## ALGORITHM #8 Binary Search

Binary Search searches for an element in a sorted array by repeatedly dividing the search interval in half.

**Important Points :**

1. Only works on sorted arrays.
2. If value of search key is less than the middle item of interval, narrow interval to lower half, otherwise to the upper half.




**Time Complexity** = O(logn)
**Space Complexity** = O(1)
where, n is the size

csmock

# COMPUTER SCIENCE FUNDAMENTALS

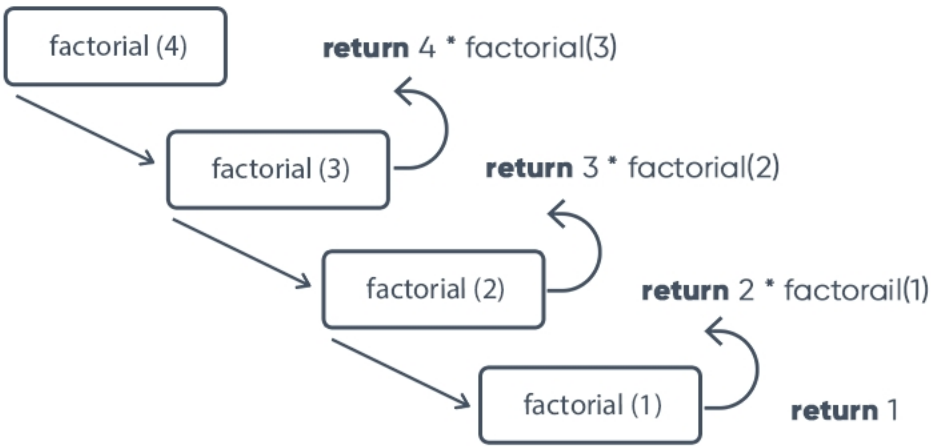
## ALGORITHM #9 Recursion

```
function factorial(n) {
    if (n == 1)
        return 1;
    return n * factorial(n-1);
}
```

Time Complexity = O(n)

Space Complexity = O(n)

where, n is the size




cs mock

# COMPUTER SCIENCE FUNDAMENTALS

## ALGORITHM #10 Sieve of Eratosthenes

The Sieve of Eratosthenes is one of the most efficient ways to find all primes smaller than n when n is smaller than $10^7$.

**Pseudo code :**

```
Let A be boolean array with default
value as true,
for i = 2, 3, 4, ... √n
    if A[i] is true
        for j = i^2, i^2 + i, ... n
            A[j] = false
return all i such that A[i] is true.
```


**Time Complexity** = O(n log (log n))
**Space Complexity** = O(√n)
where, n is the size

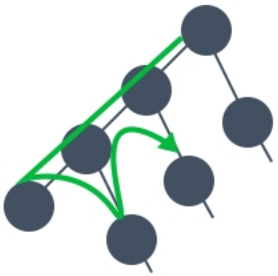
# COMPUTER SCIENCE FUNDAMENTALS

## ALGORITHM #11 Backtracking

Backtracking traces the route back upto a checkpoint from where you can again go ahead and take another route to your desired destination.

**Pseudo code :**

```
boolean solve(Node n)
    if n is a leaf node
        if the leaf is a goal node,
            return true
        else return false
    else
        for each child c of n
        if solve(c) succeeds,
            return true
        return false
```



**Applications :**

1. To find all Hamiltonian Paths.
2. To solve the N Queen problem.
3. Maze solving problem.
4. The Knight's tour problem.

# COMPUTER SCIENCE FUNDAMENTALS

## ALGORITHM #12 Greedy

Greedy algorithm makes optimal choice at each step as it attempts to find overall optimal way to solve the problem.

**Applications :**

1. Finding an optimal solution (Activity selection, Fractional Knapsack, etc).
2. Finding close to the optimal solutionfor NP-Hard problems like TSP.


**Advantages :**

1. Easy to implement.
2. Typically have less time complexities.

**Disadvantages :**

1. The local optimal solution may not always be global optimal.

# COMPUTER SCIENCE FUNDAMENTALS

## ALGORITHM #13 Kadane

Kadane's algorithm looks for all positive contiguous segments of the array and keep track of maximum sum contiguous segment among all positive segments.

**Pseudo Code :**

```
max_so_far = 0
max_ending_here = 0

loop i= 0 to n
   max_ending_here =
         max_ending_here + a[i]
   if(max_ending_here < 0)
         max_ending_here = 0
   if(max_so_far < max_ending_here)
         max_so_far = max_ending_here
return max_so_far
```


Largest Sum = 7

**Applications :**

1. Solving Maximum subarray problems.
2. Solving real-life problems involving genomic sequence analysis and computer vision.

# COMPUTER SCIENCE FUNDAMENTALS

## ALGORITHM #14 Hashing

Hashing is a one-way mathematical algorithm that maps data of arbitrary size to a hash of a fixed size.


**Popular Hashing Algorithms :**

1. Message Direct 5 (MD5).
2. Secure Hash Function (SHA).
3. RIPEMD.
4. Whirlpool.

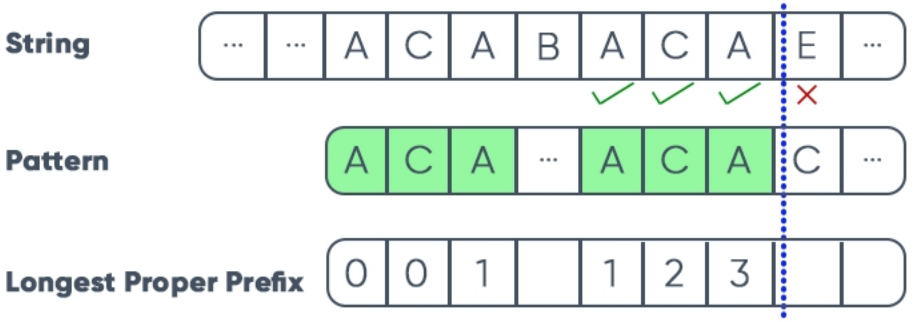
**Applications :**

1. Data Structures using Hash tables.
2. Message Digest.
3. Password Verification.
4. Compiler Operation.

# COMPUTER SCIENCE FUNDAMENTALS

## ALGORITHM #15 KMP

Knuth-Morris and Pratt (KMP) is a linear time algorithm which recognises patterns efficiently in a given string.




**Applications :**

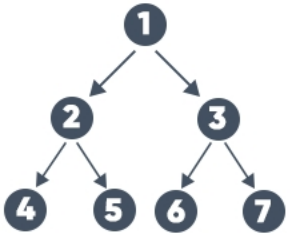
1. Pattern recognition
2. Network systems
3. Database schema

**Time Complexity** = O(n+m)
**Space Complexity** = O(m)
where, n is length of the string
m is length of pattern

# COMPUTER SCIENCE FUNDAMENTALS

## ALGORITHM #16 PreOrder, InOrder, PostOrder

Tree Traversal is a process to visit all the nodes of a tree and may print their values too.




**PreOrder :** 1 2 4 5 3 6 7

**InOrder :** 4 2 5 1 6 3 7

**PostOrder :** 4 5 2 6 7 3 1

**PreOrder** : root -> left -> right

**InOrder** : left -> root -> right

**PostOrder**: left -> right -> root

**Time Complexity** = O(n+m)

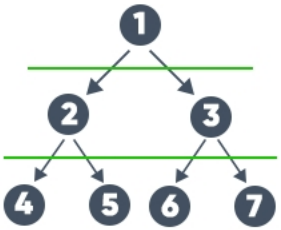
**Space Complexity** = O(h)

where, n is the number of nodes
m is the number of edges
h is the height of the tree

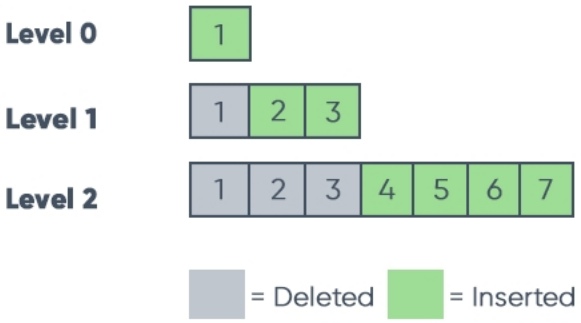
# COMPUTER SCIENCE FUNDAMENTALS

## ALGORITHM #17 Level Order Traversal

Level Order Traversal is the process where we traverse every node of a level before going to the lower level.




**Level Order :** 1 2 3 4 5 6 7




**Time Complexity** = O(n)
**Space Complexity** = O(n)

where, n is the number of nodes

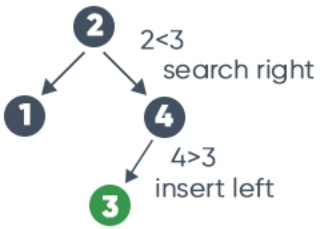
# COMPUTER SCIENCE FUNDAMENTALS

## ALGORITHM #18 Binary Search Tree

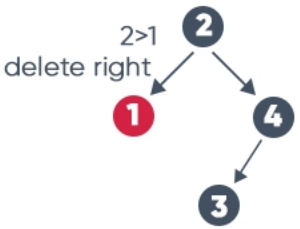
A binary search tree (BST) is a rooted binary tree where left subtree is less and right subtree is greater than the root node.

**Search(4)**


**Insert(3)**




**Delete(1)**




**Time Complexity** = O(h)
**Space Complexity** = O(n)

where, n is the number of nodes
h is the height of tree

csmock

# COMPUTER SCIENCE FUNDAMENTALS

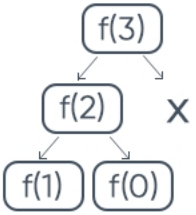
## ALGORITHM #19 Dynamic Programming

Dynamic Programming solves all possible small problems and then combine to obtain solutions for bigger problems.

**Dynamic Programming Methods :**

**1. Top-down with Memoization**

Creates a memo or a "note to self" of the values returned from solving each problem.



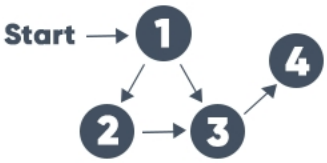

**2. Bottom-up with Tabulation** - Starts solving the sub-problems and moves up by filling up a table of n-dimensions.

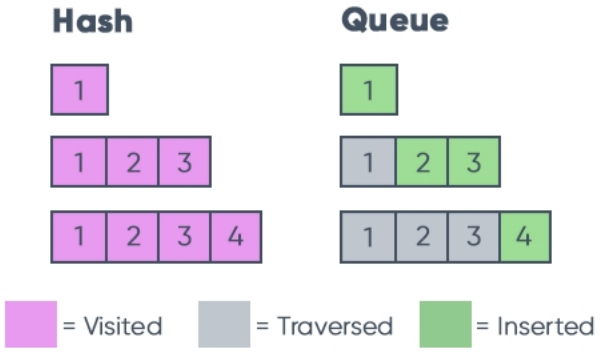

cs mock

# COMPUTER SCIENCE FUNDAMENTALS

## ALGORITHM #20 Breadth First Search

Breadth first search is a traversal algorithm that starts traversing the DS from root node and explores all the neighbouring nodes.




**Applications :**

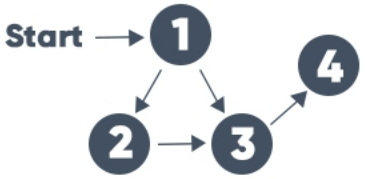
1. Crawlers in Search Engines
2. GPS Navigation systems
3. Broadcasting

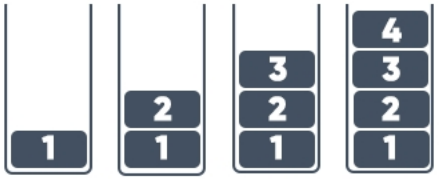
**Time Complexity** = O(|V|)
**Space Complexity** = O(|V|)
where, |V| is number of nodes

# COMPUTER SCIENCE FUNDAMENTALS

## ALGORITHM #21 Depth First Search

Depth first search starts traversing the DS from root node and explores as far as possible along each branch before backtracking.




**Search Order : 4 3 2 1**

**DFS Implementation using Stack**

**Applications :**

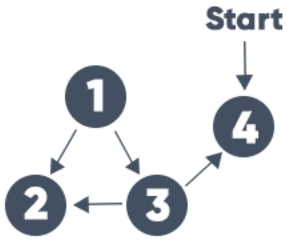
1. Detecting cycle in a graph
2. Solving puzzles like mazes
3. Path Finding

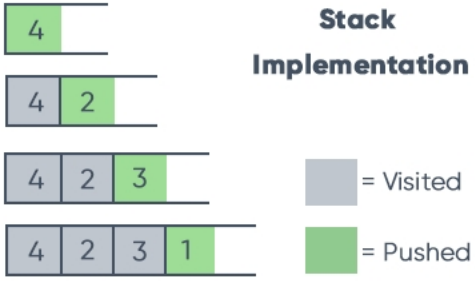
**Time Complexity** = $O(b^m)$
**Space Complexity** = $O(bm)$
where, b is branching factor
m is max depth

# COMPUTER SCIENCE FUNDAMENTALS

## ALGORITHM #22 Topological Sorting

Topological sorting takes a directed graph and returns an array of nodes where each node appears before all the nodes it points to.




**Sorting Order :** 1 3 2 4

**Applications :**

1. Maven dependency resolutions
2. Instruction Scheduling
3. Scheduling jobs

**Time Complexity** = O(V+E)
**Space Complexity** = O(V)
where, V is number of vertices
E is number of edges

cs mock

# About us :

CS Mock aims to smoothen the placement journey of college graduates by polishing their skills, highlighting the points of improvement and boosting their confidence. We provide 1-on-1 live mock interview and mentorship with industry professionals who are part of hiring team.


+91 9956127183

support@csmock.com

www.csmock.com