United International University (UIU)

Mid-term Preparation Session (Fall 2024)

Course: Data Structures & Algorithms – 1 (DSA 1)

Instructor:

Mahfuz Hasan Reza

Founder, Learn With Mahfuz

Head, Academic & Growth, UIU APP FORUM

Organized By:

Computronix Academy, UIU APP FORUM

Time Complexity Analysis

- Time Complexity Video (In-Depth)
- Time Complexity Previous Question Solve (.pdf)

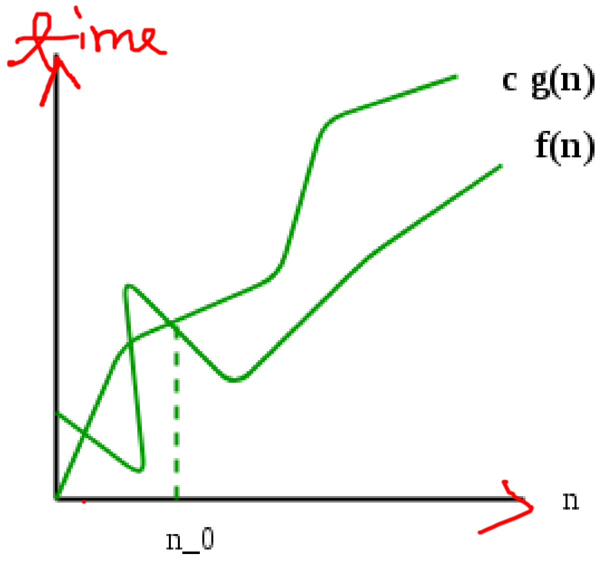
Learn With Mahfuz

Let's Code Together

Asymptotic Analysis

1. Big-Oh (O) ← Worst Case
2. Big-Omega (Ω) ← Best Case
3. Theta (θ) ← Average Case

Big-O Notation (O-notation)



$f(n) = O(g(n))$

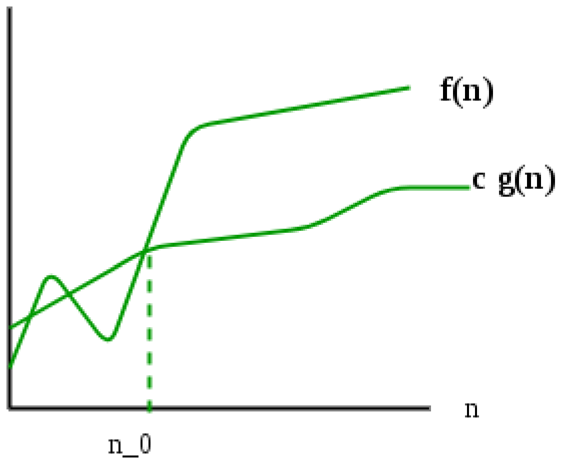
$O(g(n))$ = { $f(n)$: there exist positive constants c and n0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n0$ }

$f(n) = O(g(n))$

$0 \leq f(n) \leq c \cdot g(n)$

Learn With Mahfuz
Let's Code Together

Omega Notation (Ω-Notation):



f(n) = Omega(g(n))

$\Omega(g(n)) = \{ f(n)$: there exist positive constants c and n0 such that $0 \le cg(n) \le f(n)$ for all $n \ge n0 \}$

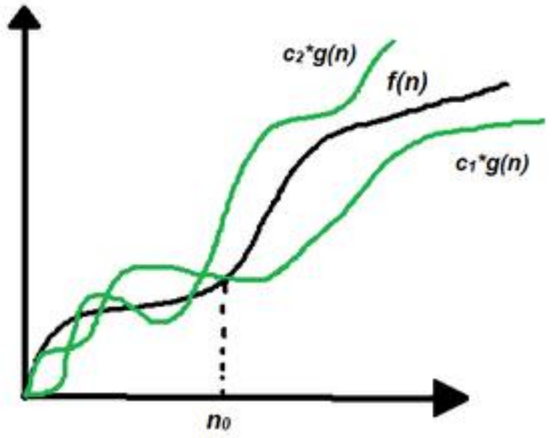
$f(n) = \Omega(g(n))$

$f(n) \geq c \cdot g(n)$; $n \geq n_0$

$n_0, c \rightarrow +ve$

Learn With Mahfuz
Let's Code Together

Theta Notation (Θ-Notation):



*Θ (g(n)) = {f(n): there exist positive constants c1, c2 and n0 such that 0 ≤ c1 * g(n) ≤ f(n) ≤ c2 * g(n) for all n ≥ n0}*

Note: *Θ(g) is a set*

$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$; $n \geq n_0$

$n_0, c \rightarrow +ve$

$\downarrow$

$f(n) = \Theta(g(n))$

Function	Descriptor	Big-Oh
c	Constant	$O(1)$
$\log n$	Logarithmic	$O(\log n)$
n	Linear	$O(n)$
$n \log n$	$n \log n$	$O(n \log n)$
n^2	Quadratic	$O(n^2)$
n^3	Cubic	$O(n^3)$
n^k	Polynomial	$O(n^k)$
2^n	Exponential	$O(2^n)$
$n!$	Factorial	$O(n!)$

Insertion Sort

- Insertion Sort in 1 Video (Theory, Code, Dry Run (Simulation) and Time Complexity)

Learn With Mahfuz
Let's Code Together

Insertion Sort Algorithm

```
for(int i=1; i<size; i++){
    int tmp=arr[i];
    int j=i-1;

    while(arr[j]>tmp && j>=0){
        arr[j+1]=arr[j];
        j--;
    }
    arr[j+1]=tmp;
}
```

Insertion Sort Complexity

Time Complexity	
Best	O(n)
Worst	$O(n^2)$
Average	$O(n^2)$
Space Complexity	O(1)

Arrays: Memory Mapping

$Add(a[i]) = L_0 + i \times size$

$L_0 + (i-1) \times S$

Row-Major Mapping

$$Address(A[i][j]) = L_o + [\, i * n + j \,] * w$$

Column-Major Mapping

$$Address(A[i][j]) = L_o + [\, j * m + i \,] * w$$

Linear Search

```
int search(int array[], int n, int x) {

  // Going through array sequencially
  for (int i = 0; i < n; i++)
    if (array[i] == x)
      return i;
  return -1;
}
```

Binary Search

```
do until the pointers low and high meet each other.
    mid = (low + high)/2
    if (x == arr[mid])
        return mid
    else if (x > arr[mid]) // x is on the right side
        low = mid + 1
    else                       // x is on the left side
        high = mid - 1
```

```
int binarySearch(int array[], int x, int low, int high) {

        // Repeat until the pointers low and high meet each other
  while (low <= high) {
    int mid = low + (high - low) / 2;

    if (x == array[mid])
      return mid;

    if (x > array[mid])
      low = mid + 1;

    else
      high = mid - 1;
  }

  return -1;
}
```

Binary Search

Time Complexities

- **Best case complexity:** `O(1)`
- **Average case complexity:** `O(log n)`
- **Worst case complexity:** `O(log n)`

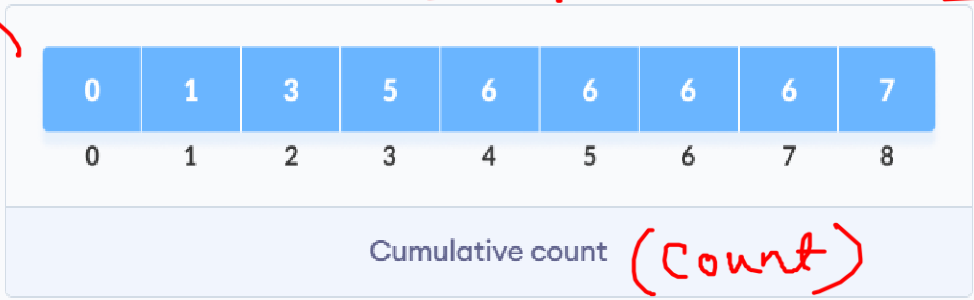
Space Complexity

The space complexity of the binary search is `O(1)`.

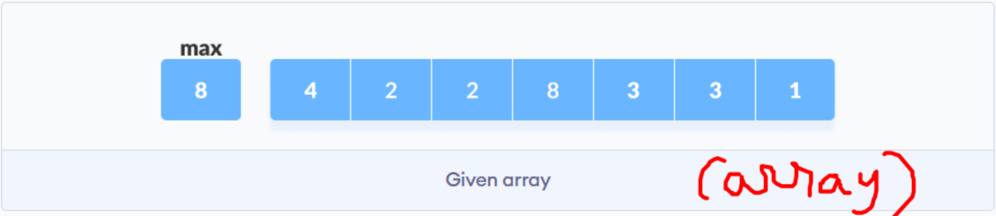
Learn With Mahfuz
Let's Code Together

Counting Sort

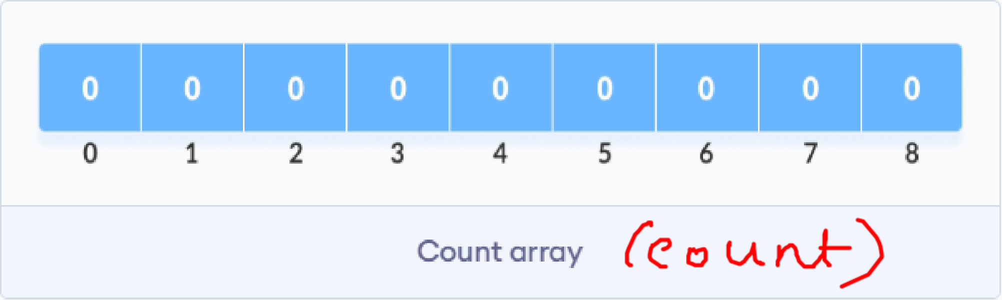
for(i=1; i<n; i++) count[i] += count[i-1]



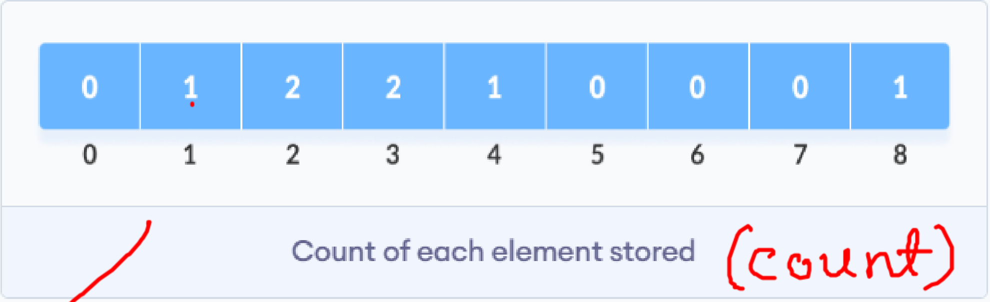
(count)



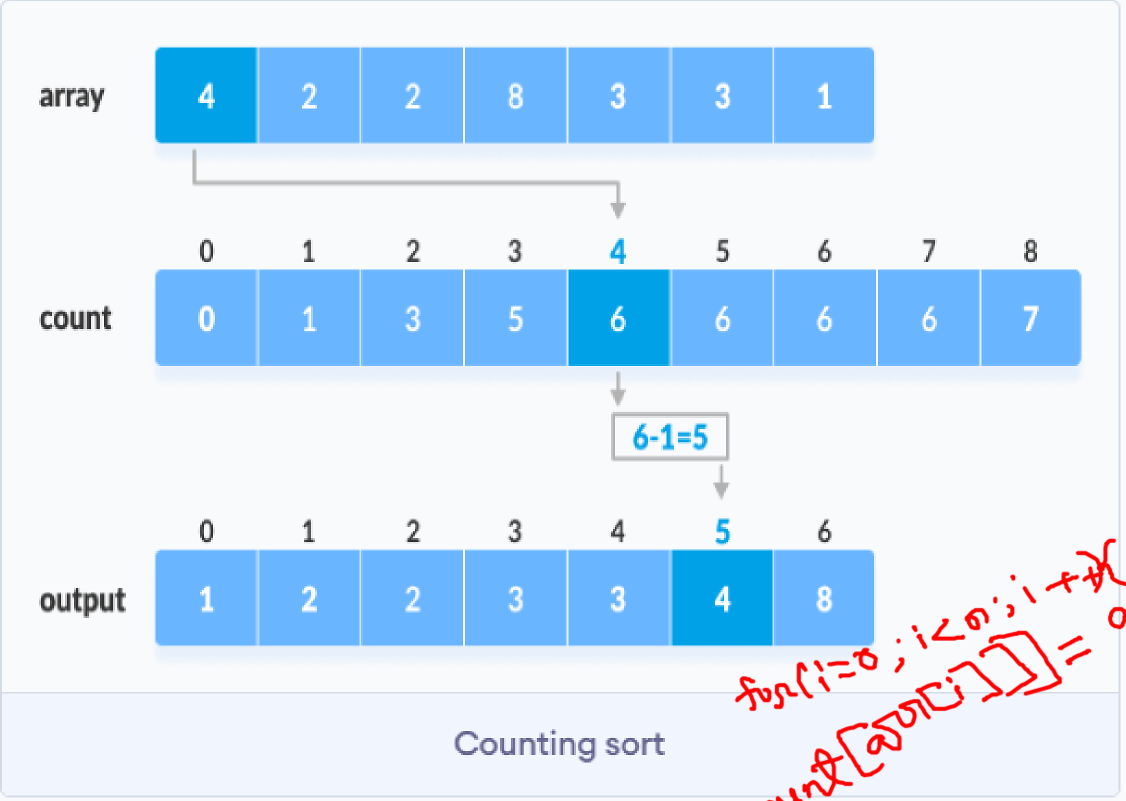
(array)



(count)



(count)



for(i=0; i<n; i++){
output[--count[arr[i]]] = arr[i];

for(i=0; i<n; i++)
count[arr[i]]++;

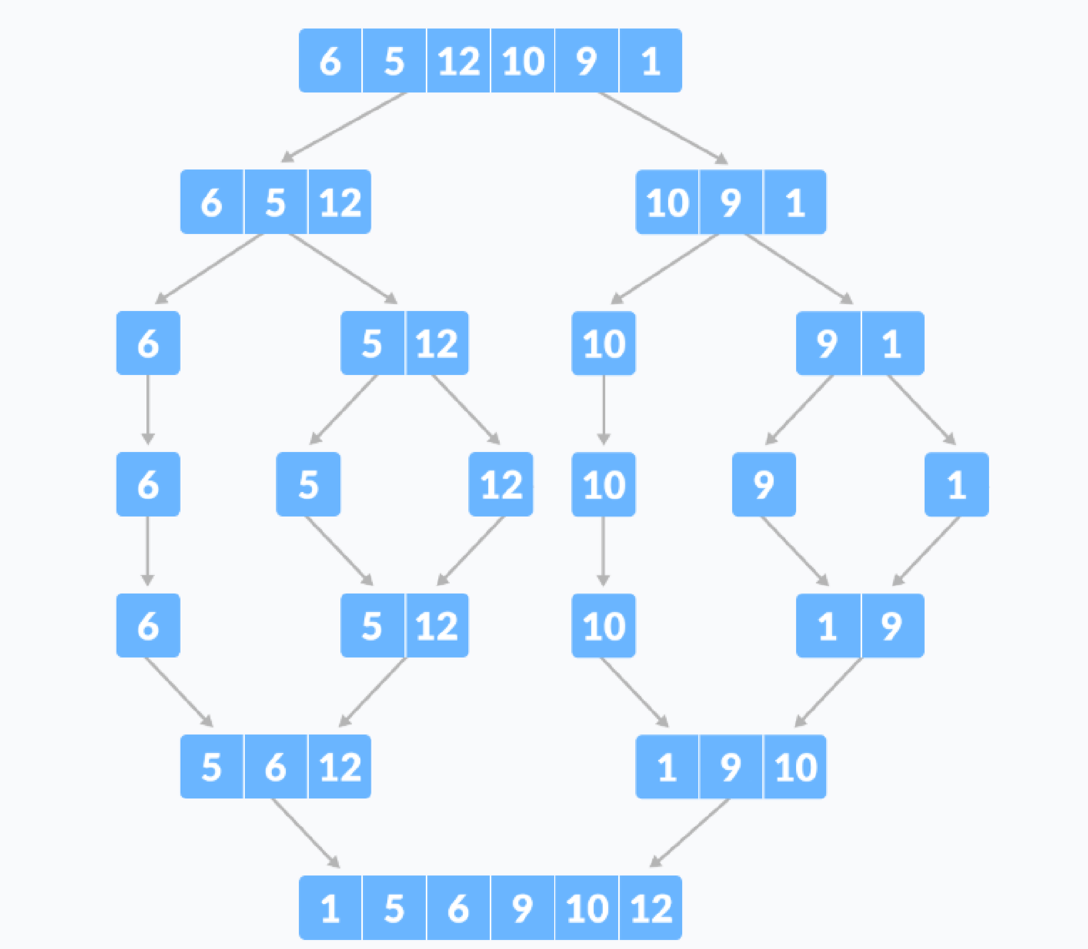
Learn With Mahfuz
Let's Code Together

Counting Sort

Complexity

Time Complexity	
Best	O(n+max)
Worst	O(n+max)
Average	O(n+max)
Space Complexity	O(max)
Stability	Yes

Merge Sort



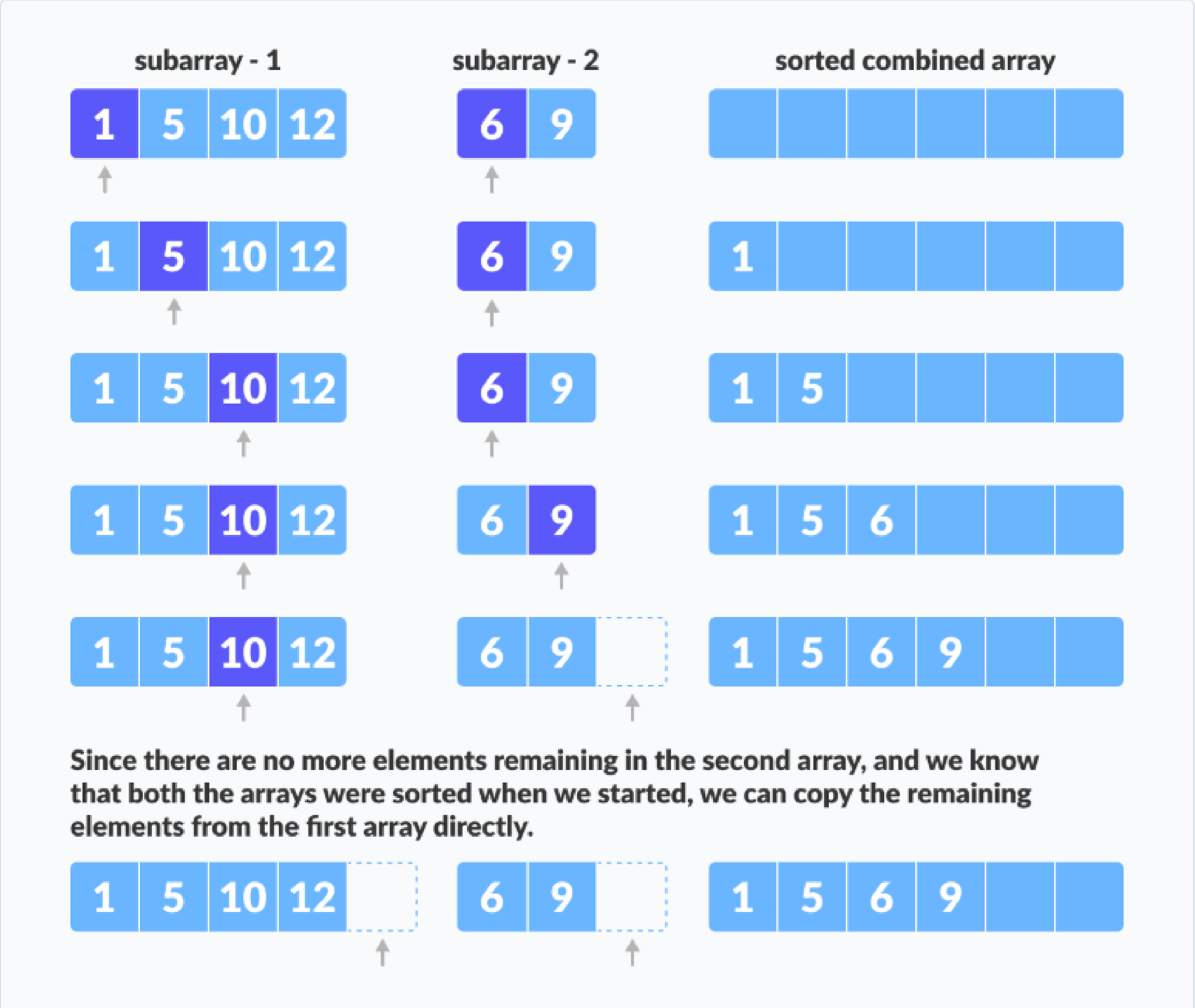
Merge Sort example

Learn With Mahfuz
Let's Code Together

Merge Sort

```
MergeSort(A, p, r):
    if p > r
        return
    q = (p+r)/2
    mergeSort(A, p, q)
    mergeSort(A, q+1, r)
    merge(A, p, q, r)
```

```
Have we reached the end of any of the arrays?
    No:
        Compare current elements of both arrays
        Copy smaller element into sorted array
        Move pointer of element containing smaller element
    Yes:
        Copy all remaining elements of non-empty array
```



Merge step

Merge Sort Complexity

Time Complexity	
Best	O(n*log n)
Worst	O(n*log n)
Average	O(n*log n)
Space Complexity	O(n)
Stability	Yes

quicksort(arr, pi, high)

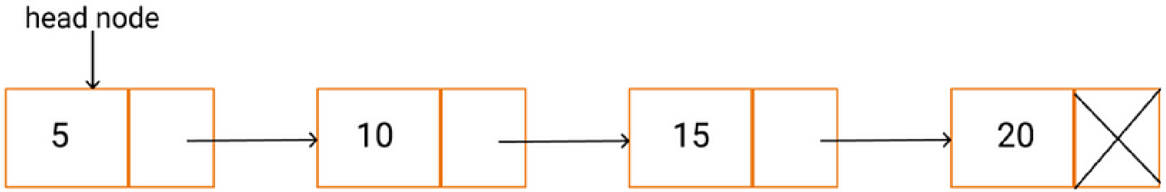
Select pivot element of in each half and put at correct place using recursion

Quicksort Complexity

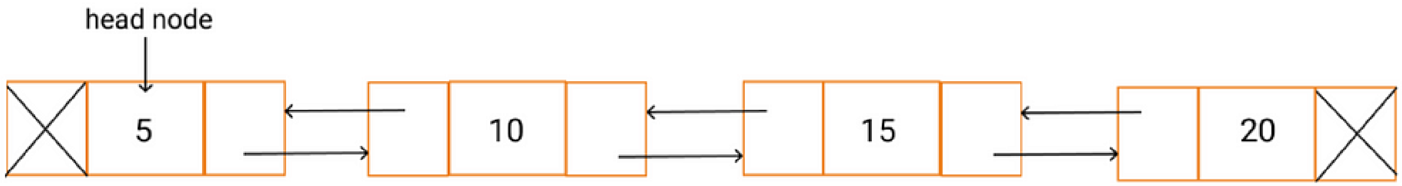
Time Complexity	
Best	$O(n*\log n)$
Worst	$O(n^2)$
Average	$O(n*\log n)$
Space Complexity	$O(\log n)$
Stability	No

Linked List

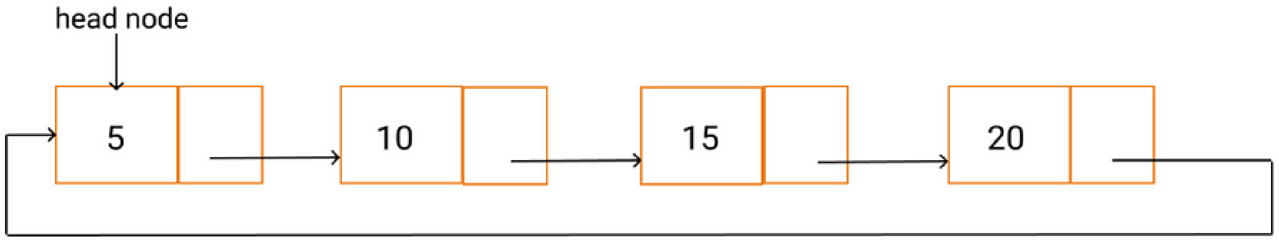
Singly Linked List



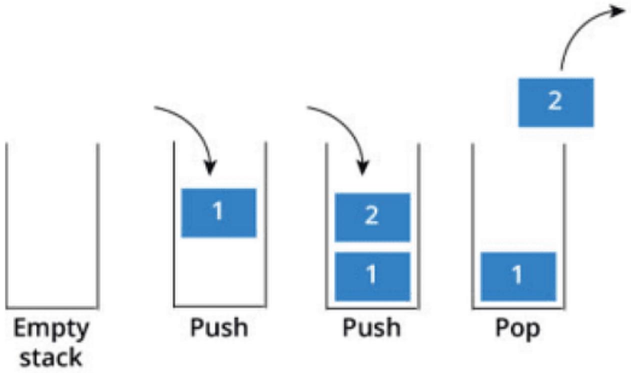
Doubly Linked List



Circular Linked List



Stack and Queue



Stack

Queue

Tower of Hanoi

- Move n-1 Discs from A to B using C
- Move a Disc from A to C
- Move n-1 Discs from B to C using A

```
void TOH(int n,  int A,  int B,  int C)
{
        if(n>0)
        {
                TOH(n-1, A , C , B);
                printf( "Move a Disc from %d to %d", A , C);
                TOH(n-1, B , A , C);
        }
}
```

Application of Stacks: Convert Infix to Postfix Expression

a + (b - c + h) * D + E

Input Exp	Stack	Postfix Exp
a		a
+	+	a
(	+ (	a
b	+ (	ab
-	+ (-	ab
c	"	abc
+	+ (+	abc -
h	"	abc - h
)	+	abc - h +
*	+ *	
D	+	
E		abc - h + D * + E +

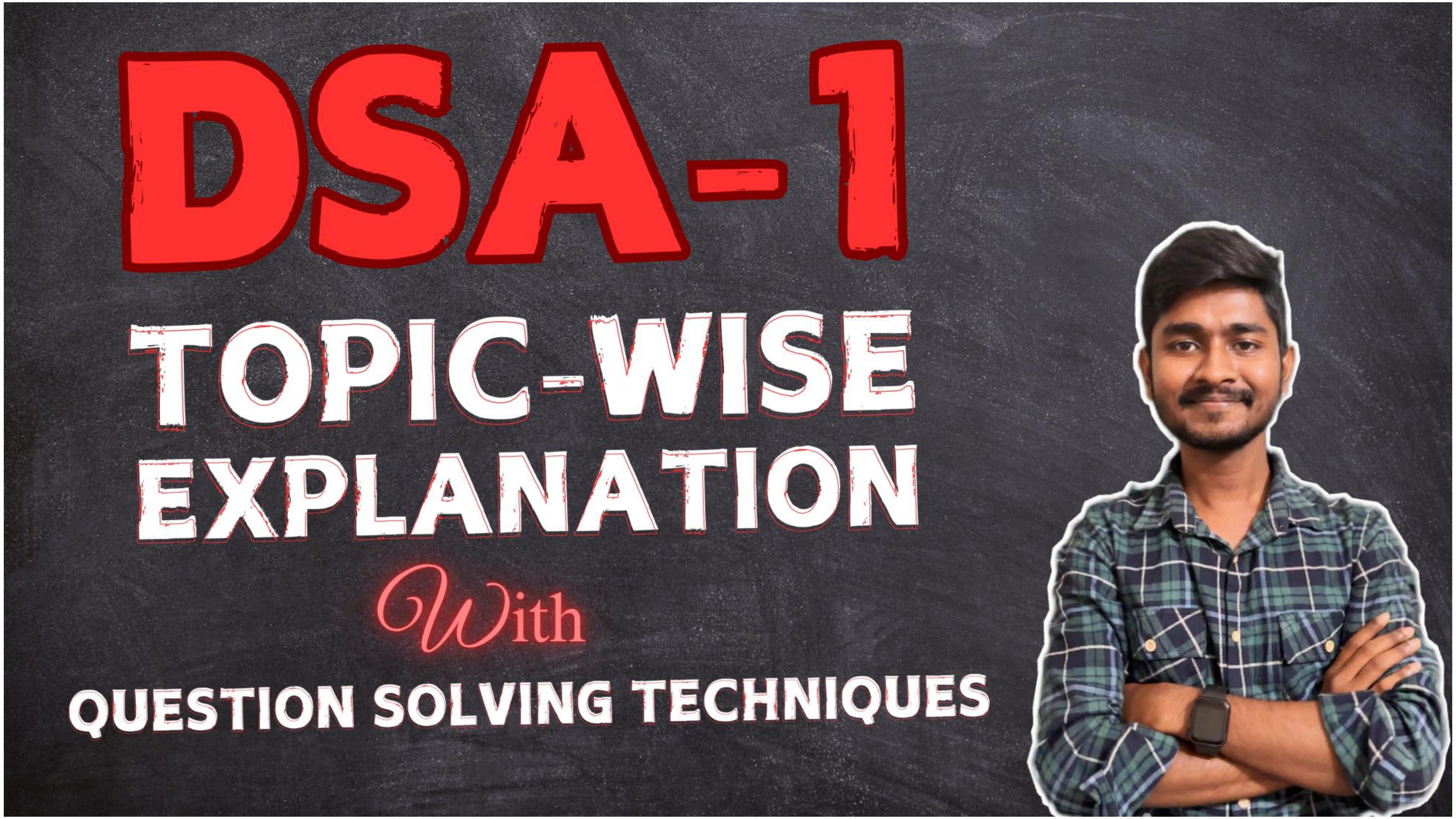
Presendce:

1. ( ) →
2. ^ → R → L
3. * / } L → R
4. + -

x + y → operator, operand → Exp. √

operator:

1. stack (empty) → push
2. high → push; low → pop, check
3. same → L → R : pop, ch; R → L : push



Click [here] to see this video!

THANK YOU!

Learn With Mahfuz

Let's Code Together