Stacks:

Definition: A stack is a linear data structure that follows the Last-InFirst-Out (LIFO) principle. It means that the last element added to the stack will be the first one to be removed. Think of it as a collection of elements with two primary operations: `push` (to add an element) and `pop` (to remove the top element).

Stack Operations:

1. `push(item)`: Adds an item to the top of the stack.
2. `pop()`: Removes and returns the top item from the stack.
3. `peek()`: Returns the top item without removing it.
4. `isEmpty()`: Checks if the stack is empty.
5. `size()`: Returns the number of elements in the stack.

Array Representation of Stacks:

In the array representation, a stack can be implemented using a simple array. We need to keep track of the top element's index. When an element is pushed, we increment the top index, and when an element is popped, we decrement the top index.

Stacks using Dynamic Arrays:

Dynamic arrays allow the stack to grow and shrink dynamically as elements are pushed and popped. When the array is full, a new larger array is created, and all elements are copied to it. Similarly, when the number of elements reduces to a certain threshold, a smaller array can be created to save memory.

Stack Applications:

1. Polish Notation (Prefix Notation): Polish notation is a way of representing arithmetic expressions without using parentheses. Operators come before their operands. For example: `+ 2 3` means 2 +

3. Stacks can be used to evaluate and convert expressions from and to Polish notation.

2. Infix to Postfix Conversion: Converting an arithmetic expression from infix notation (operators between operands) to postfix notation (operators after operands) can be done using a stack. It helps in simplifying the expression evaluation process.

3. Evaluation of Postfix Expression: Postfix expressions are easy to evaluate using a stack. The expression is scanned from left to right, and operands are pushed into the stack. When an operator is encountered, the top operands from the stack are popped, and the result of the operation is pushed back onto the stack until the entire expression is evaluated.

In summary, stacks are a fundamental data structure with various applications, including expression evaluation and conversion. They provide a simple and efficient way to manage data following the Last-InFirst-Out principle. Arrays and dynamic arrays can be used for stack implementation, and they find extensive use in solving many algorithmic problems.

Recursion:

Definition: Recursion is a programming technique where a function calls itself to solve a problem. It breaks down a complex problem into smaller subproblems and solves them iteratively until a base case is reached, at which point the recursion stops.

Factorial:

Factorial is the product of all positive integers up to a given number n. It is denoted by `n!`.

Factorial using Recursion:

The factorial of a number `n` can be calculated using a recursive function as follows:

```
python def
factorial(n):     if n ==
0 or n == 1:
        return 1
else:
        return n * factorial(n-1)
```

GCD (Greatest Common Divisor):

The Greatest Common Divisor (GCD) of two or more integers is the largest positive integer that divides all of them without leaving a remainder.

GCD using Recursion:

The GCD of two numbers `a` and `b` can be calculated using a recursive function based on Euclid's algorithm:

```
python def
gcd(a, b):
if b == 0:
return a
else:
        return gcd(b, a % b)
```

Fibonacci Sequence:

The Fibonacci sequence is a series of numbers in which each number (known as a Fibonacci number) is the sum of the two preceding ones. It starts with 0 and 1, and the sequence continues as 0, 1, 1, 2, 3, 5, 8, 13, 21, and so on.

Fibonacci Sequence using Recursion:

The Fibonacci sequence can be generated using a recursive function as follows:

```
python def
fibonacci(n):
if n <= 0:
return 0     elif n
== 1:
return 1     else:
        return fibonacci(n-1) + fibonacci(n-2)
```

Tower of Hanoi:

The Tower of Hanoi is a classic mathematical puzzle that consists of three rods and a number of disks of different sizes, which can slide onto any rod. The puzzle starts with all the disks on one rod, arranged in increasing order of size. The objective is to move the entire stack of disks to another rod, adhering to the following rules:

1. Only one disk can be moved at a time.
2. A larger disk cannot be placed on top of a smaller disk.

Tower of Hanoi using Recursion:

The Tower of Hanoi problem can be solved using a recursive function:

```
python def tower_of_hanoi(n, source, auxiliary, target):     if n == 1:         print(f"Move disk 1 from {source} to {target}")         return     tower_of_hanoi(n-1, source, target, auxiliary) print(f"Move disk {n} from {source} to {target}") tower_of_hanoi(n-1, auxiliary, source, target)
```

In summary, recursion is a powerful programming technique used to solve problems by dividing them into smaller instances of the same problem. The factorial, GCD, Fibonacci sequence, and Tower of Hanoi are classic examples that demonstrate the use of recursion to solve different types of problems. When using recursion, it is crucial to have a base case to stop the recursion and avoid infinite loops.

Queues:

Definition: A queue is a linear data structure that follows the First-InFirst-Out (FIFO) principle. It means that the first element added to the queue will be the first one to be removed. It is similar to standing in a queue in real life, where the first person in line is served first.

Array Representation of Queues:

A queue can be implemented using a simple array. We need to keep track of the front and rear of the queue. When an element is enqueued (added to the queue), it is added at the rear. When an element is dequeued (removed from the queue), it is removed from the front.

Queue Operations:

1. `enqueue(item)`: Adds an item to the rear of the queue. Enqueue is the process of adding an element to the rear (end) of the queue. When you enqueue an element, it becomes the last element in the queue.

2. `dequeue()`: Removes and returns the item from the front of the queue.
3. `peek()`: Returns the item at the front without removing it.
4. `isEmpty()`: Checks if the queue is empty.
5. `size()`: Returns the number of elements in the queue.

Circular Queues:

In a standard queue, after dequeueing several elements, the front of the queue moves ahead, leaving unused space at the front. In a circular queue, the front and rear are conceptually connected, forming a circle. When the rear reaches the end of the array, it wraps around to the front, making the queue more efficient in terms of space usage.

Circular Queues using Dynamic Arrays:

Circular queues can be implemented using dynamic arrays, which allow the queue to grow and shrink dynamically. When the queue is full and a new element needs to be enqueued, a new larger array is created, and all elements are copied to it. Similarly, when the number of elements reduces to a certain threshold, a smaller array can be created to save memory.

Deque:

A deque (Double-ended queue) is a variation of a queue where elements can be added or removed from both ends. It supports insertion and deletion from both the front and the rear, providing more flexibility than a standard queue.

Priority Queues:

A priority queue is an abstract data type that operates similar to a regular queue, but each element has an associated priority. The element with the highest priority is dequeued first. Priority queues are often implemented using a binary heap data structure.

Problems with Priority Queues:

Priority queues find various applications in problems that require efficient access to the element with the highest priority. Some common problems where priority queues are used include:

1. Dijkstra's Shortest Path Algorithm: Finding the shortest path in a graph from a source node to all other nodes, where the edge weights represent distances.

2. Heap Sort: A sorting algorithm that uses a binary heap (a type of priority queue) to sort an array efficiently.

3. Kth Largest (or Smallest) Element: Finding the kth largest (or smallest) element in an array efficiently using a priority queue.

4. Merging Sorted Lists: Merging multiple sorted lists into a single sorted list using a priority queue.

In summary, queues are essential data structures that follow the First-InFirst-Out (FIFO) principle. They have various representations, including circular queues and deque, which add more flexibility to the basic queue operations. Priority queues are used to address problems where elements have associated priorities, allowing efficient access to the highest-priority element.