Python Data Structures and Algorithms

1. Arrays & Linked Lists

Arrays

- **Definition**: A collection of elements stored at contiguous memory locations.

- **Characteristics:**

  - Fixed size.

  - Fast access by index.

- **Example:**

```
# Creating an array (list in Python)
numbers = [1, 2, 3, 4, 5]
print(numbers[2])  # Output: 3
```

Linked Lists

- **Definition**: A collection of nodes where each node contains data and a reference to the next node.

- **Characteristics**:

  - Dynamic size.

  - Fast insertion/deletion at the cost of slower access.

- **Example**:

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def append(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        last = self.head
        while last.next:
            last = last.next
        last.next = new_node

ll = LinkedList()
ll.append(1)
ll.append(2)
```

2. Heap & Stack

Heap

- **Definition**: A special tree-based data structure that satisfies the heap property.

- **Characteristics:**

  - Can be a Max Heap (parent node > child nodes) or Min Heap (parent node < child nodes).

- **Example:**

```python
import heapq

# Creating a min-heap
heap = [3, 1, 4, 1, 5, 9]
heapq.heapify(heap)
heapq.heappush(heap, 2)
print(heap)  # Output: [1, 1, 4, 3, 5, 9, 2]
```

Stack

- **Definition**: A linear data structure following Last In, First Out (LIFO) principle.

- **Characteristics**:

  - **Operations**: Push (add), Pop (remove), Peek (view top element).

- **Example**:

```python
stack = []
stack.append(1)  # Push
stack.append(2)
print(stack.pop())  # Output: 2 (Pop)
```

3. Binary Search

- **Definition**: An efficient algorithm for finding an item from a sorted list of items.

- **Characteristics**:

  - Time complexity: O(log n).

  - Requires a sorted array.

- **Example**:

```python
def binary_search(arr, target):
    low, high = 0, len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1

sorted_array = [1, 3, 5, 7, 9]
print(binary_search(sorted_array, 7))  # Output: 3
```

4. Recursion

- **Definition**: A method where the solution to a problem depends on solutions to smaller instances of the same problem.

- **Characteristics**:

  - Base case to end recursion.

  - Example: Calculating factorial.

- **Example**:

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

print(factorial(5))  # Output: 120
```

5. The Sorting Algorithm

Bubble Sort

- **Definition**: A simple comparison-based sorting algorithm.

- **Characteristics:**

  - Time complexity: O(n^2).

  - Repeatedly swaps adjacent elements if they are in the wrong order.

- **Example**:

```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]

arr = [64, 34, 25, 12, 22, 11, 90]
bubble_sort(arr)
print(arr)  # Output: [11, 12, 22, 25, 34, 64, 90]
```

Conclusion

- **Arrays** are great for indexed access but have fixed sizes. Linked Lists offer dynamic sizing and easy insertions/deletions.

- **Heaps** and **Stacks** provide different methods of managing data based on specific use cases.

- **Binary Search** is efficient for finding elements in sorted arrays.

- **Recursion** allows solving problems by breaking them into smaller sub-problems.

- **Sorting Algorithms** like **Bubble Sort** help in arranging data, though more efficient algorithms exist.
