Memory: Stack vs Heap – Summary

1. Introduction

In programming, memory is managed in two main areas: **Stack** and **Heap**. Understanding the differences helps in writing efficient code and avoiding errors.

2. Stack Memory

- **Definition**: Stack is a region of memory used for **static memory allocation**.
- **Usage**: Stores local variables, function parameters, and return addresses.
- **Lifetime**: Memory is automatically allocated and deallocated when functions are called and return.
- **Size**: Limited and predefined by system.
- **Access**: Fast because it works in **LIFO (Last-In-First-Out)** order.
- **Example**:

```
void func() {
    int x = 10; // stored in stack
}
```

3. Heap Memory

- **Definition**: Heap is a region of memory used for **dynamic memory allocation**.
- **Usage**: Stores objects and variables whose lifetime is controlled manually.
- **Lifetime**: Memory persists until it is explicitly freed.
- **Size**: Much larger than stack.
- **Access**: Slower than stack because allocation/deallocation is manual.
- **Example**:

```
int* ptr = new int(10); // stored in heap
// ... use ptr
delete ptr; // free heap memory
```

4. Key Differences

Feature	Stack	Heap
Allocation	Static	Dynamic
Management	Automatic	Manual
Lifetime	Function scope	Until manually freed

Feature	Stack	Heap
Size	Limited	Large
Speed	Fast	Slower
Access Pattern	LIFO	Random
Example	Local variables	Objects, dynamic arrays

5. Advantages & Disadvantages

Stack: - Advantages: Fast, managed automatically. - Disadvantages: Limited size, cannot resize dynamically.

Heap: - Advantages: Large size, flexible allocation. - Disadvantages: Slower, manual management required, risk of memory leaks.

6. Tips for Using Stack and Heap

- Use **stack** for small, short-lived variables.
- Use **heap** for large, dynamic, or long-lived objects.
- Always **free heap memory** to avoid memory leaks.
- Avoid deep recursion to prevent stack overflow.

7. Summary

- Stack = automatic, fast, LIFO, limited size.
- Heap = manual, slower, large, flexible.
- Correct usage improves performance and prevents errors.