1. Garbage Collection Algorithms

1. Mark-and-Sweep Algorithm

- **Process**:

    1. **Mark phase**: Traverse the object graph starting from GC Roots (local variables, static fields, etc.), mark all reachable (alive) objects.

    2. **Sweep phase**: Scan heap, reclaim (delete) all unmarked objects.

- **Pros**: Simple, doesn’t need moving objects.

- **Cons**: Causes **memory fragmentation**, which can slow down future allocations.

---

2. Replication (Copying) Algorithm

- **Process**:
  - Divide memory into two equal semi-spaces.
  - Allocate objects in one space.
  - When GC happens, copy all live objects to the other space and clean up the first one.
- **Pros**: Eliminates fragmentation, allocation is fast (sequential).
- **Cons**: Wastes half of the memory, copying overhead when many live objects.

---

3. Mark-Clear (Mark-Compact) Algorithm

- **Process**:
  - First mark live objects.
  - Then instead of sweeping, move live objects together (compact) to eliminate fragmentation.
- **Pros**: Solves fragmentation problem of mark-sweep.
- **Cons**: Higher overhead because objects must be moved.

---

4. Generational Collection Algorithm

Idea: Objects have different lifetimes, so use different algorithms for different generations.

- **Young Generation**: Most objects die quickly → use **copying algorithm** (fast).
- **Old Generation**: Objects live long → use **mark-sweep** or **mark-compact**.

- **Pros**: Matches real-world object lifetime patterns, improves efficiency.

---

2. Garbage Collectors in HotSpot JVM

1. Serial Collector

- **Single-threaded** collector.
- Uses **stop-the-world** (pauses all application threads).
- Young gen: Copying algorithm.
- Old gen: Mark-Compact.
- **Use case**: Single-core CPU, small heap.

---

2. ParNew Collector

- Multi-threaded version of **Serial collector**.
- Young gen: Copying algorithm.
- Often used with **CMS** for old generation.

---

3. Parallel Scavenge Collector

- Focused on **throughput** (maximize work done vs. GC time).
- Young gen: Copying algorithm, multi-threaded.
- Has **adaptive tuning**: JVM automatically adjusts GC behavior for performance.
- **Use case**: Background tasks, batch jobs.

---

4. Serial Old Collector

- Old generation version of **Serial collector**.
- Uses **mark-compact**.
- Backup for CMS when concurrent collection fails.

---

5. Parallel Old Collector

- Old gen companion to **Parallel Scavenge**.
- Multi-threaded mark-compact.
- Good for throughput-focused applications.

---

6. CMS (Concurrent Mark-Sweep) Collector:

- Old generation collector.
- Aims to reduce pause time.
- Steps:
    1. Initial mark (short stop-the-world).

2. Concurrent mark (application still running).
3. Remark (short stop-the-world).
4. Concurrent sweep.

- **Pros**: Low pause time.
- **Cons**: Memory fragmentation, CPU overhead.

---

7. G1 (Garbage-First) Collector

- Splits heap into **regions** instead of fixed young/old.
- Collects regions with most garbage first.
- Uses **mark-compact** (with region-based compaction).
- **Pros**: Predictable pause time, avoids fragmentation.
- Default GC in modern JDK (since Java 9).

---

8. ZGC (Z Garbage Collector) (Almost all GC work is concurrent with main program)

- **Low-latency GC** (pause times < 10ms, regardless of heap size).
- Works with **huge heaps** (TB scale).
- Uses **colored pointers** and **load barriers** to manage memory concurrently.
- **Pros**: Extremely short pause times, scales well.
- **Cons**: Higher CPU/memory overhead compared to simpler GCs.

✅ Summary Table

GC Algorithm	Used In	Pros	Cons
Mark-Sweep	Old gen	Simple	Fragmentation
Copying	Young gen	Fast, no fragmentation	Wastes memory

Mark-Compact	Old gen	No fragmentation	Slower (object moving)
Generational	All	Efficient, practical	Complexity

Collector	Target	Characteristics
Serial	Small heap	Single-threaded, simple
ParNew	Young gen	Multi-threaded Serial
Parallel Scavenge	Young gen	Throughput-oriented
Serial Old	Old gen	Single-threaded, backup
Parallel Old	Old gen	Multi-threaded, throughput

CMS	Old gen	Low latency, concurrent
G1	Whole heap	Region-based, balanced
ZGC	Whole heap	Ultra-low pause, scalable