The comparison of different garbage collection algorithms in Java :

**Serial GC**

- Uses a single thread to perform garbage collection

- Pauses the application during garbage collection

- Suitable for small applications with limited heap size

- Advantages:

- Simple implementation

- Low overhead

- Disadvantages:

- Can cause significant pauses in the application

- Not suitable for large applications

**Parallel GC**

- Uses multiple threads to perform garbage collection

- Pauses the application during garbage collection

- Suitable for applications with large heap sizes

- Advantages:

- Faster garbage collection times

- Better performance for large applications

- Disadvantages:

- Can still cause significant pauses in the application

- Increased overhead due to multiple threads

**CMS (Concurrent Mark-Sweep) GC**

- Uses multiple threads to perform garbage collection concurrently with the application

- Attempts to minimize pauses in the application

- Suitable for applications with large heap sizes and low pause time requirements

- Advantages:

- Low pause times

- Good performance for large applications

- Disadvantages:

- Complex implementation

- Can still cause occasional pauses

- Increased overhead due to concurrent garbage collection

**G1 (Garbage-First) GC**

- Uses multiple threads to perform garbage collection concurrently with the application

- Divides the heap into regions and collects garbage in each region separately

- Suitable for applications with large heap sizes and low pause time requirements

- Advantages:

- Low pause times

- Good performance for large applications

- Flexible region sizing

- Disadvantages:

- Complex implementation

- Increased overhead due to concurrent garbage collection

**ZGC (Z Garbage Collector) GC**

- Uses a single thread to perform garbage collection concurrently with the application

- Uses a mark-sweep-compact algorithm to minimize pauses

- Suitable for applications with large heap sizes and low pause time requirements

- Advantages:

- Very low pause times (typically <10ms)

- Good performance for large applications

- Simple implementation

- Disadvantages:

- Currently experimental and not yet widely adopted

- May require additional tuning for optimal performance