Garbage Collector vs Reference Counting

In the early days of programming, programmers faced many issues concerning memory such as slow processing speed and little memory space to work with. When an object is created and a variable is used to refer to it. Upon termination, only the variable is destroyed. This means that only a reference to the object is deleted, but the actual object still lives in memory. Roughly describing, if there happens to be too many of these rogue objects in memory, the computer will be resource starved and will be unable to perform any more instructions. Programmers came up with ways to solve this problem with the most common method being manual memory management (MMM). This allows us to keep track of each object and release them at the right time.

Nowadays, we have automatic memory management (AMM) in modern language. The two most commonly used is Garbage Collection (GC) and Automatic Reference Counting (ARC). While GC is used by languages such as Java, JavaScript, C#, etc., ARC is used by Objective C, Swift, C++ smart pointers, etc.

GC is used at runtime in the background. It detects inactive objects and removes them from memory. An object is considered inactive when it is unreachable, or, simply put, there are no references to the object. ARC on the other hand is deployed automatically as a compile mechanism at compile time. It injects memory management calls such as retain and release on objects as the reference point fluctuate. A reference count indicates when an object can be destroyed. As more variables are referenced to the object, the reference count goes up. As all the variables are being nullified, the reference count goes down. Finally, when the reference count reaches zero, the object's memory is deallocated.

These two methods both have their strengths and weaknesses:

	Strengths	Weaknesses
ARC	+ Local, incremental processes + reclaims memory immediately	- Unable to handle retained cycles(circular object referencing)
GC	+ Don't need to worry about adding "free" or "delete" statements to code + Never have bugs related to dangling references/pointers (double frees, etc.)	- Random(indeterminate) deallocation - Cannot implement own class destructor - CG algorithms takes a lot of memories to execute compared to ARC

Performance wise, there is no clear winner. They both have their own advantages and disadvantages.