h) Random crashes - you are given a source code to test and it randomly crashes and it never crashes in the same place (you have attached a debugger and you find this). Explain what all you would suspect and how would you go about with isolating the cause

Bonus - The deeper you go into computer architecture and explain, better.

1. **Review the code:** Starting off by thoroughly reviewing the source code. Look for potential programming errors such as memory leaks, null pointer dereferences, buffer overflows, or race conditions. Pay attention to areas that involve concurrency, memory management, input/output operations, or system resources.
2. **Review error logs:** Checking for any error messages or log files generated by the program during the crashes. Looking for recurring error codes, warnings, or exceptional events that might provide insights into the cause.
3. **Memory-related issues:** Random crashes can often be caused by memory-related problems. Monitor memory usage, including heap and stack allocations, to identify potential memory leaks, buffer overflows, or other memory management issues.
4. **Hardware considerations:** Random crashes could also be related to hardware issues. Checking for potential hardware faults such as faulty RAM, overheating, power supply problems, or faulty disk drives. Run hardware diagnostics and stress tests to identify any hardware-related issues.
5. **Operating system and dependencies:** Ensuring that the software program is compatible with the operating system and required dependencies. Checking for compatibility issues, outdated libraries, or conflicts that might lead to crashes.
6. **Logging and monitoring:** Implementing robust logging and monitoring mechanisms within the code to capture more detailed information about the crashes. This can provide valuable insights into the system's state leading up to the crash.
7. **Code instrumentation:** Introducing additional logging or tracing statements in critical sections of the code to gather more specific information about the crash conditions. This can help narrow down the problematic areas.
8. **Systematic testing:** Performing systematic testing by narrowing down the problem space. Disable or isolate specific sections of the code to identify if a particular component or functionality is causing the crashes. Using techniques like binary search or divide-and-conquer to pinpoint problematic code segments.
9. **Collaborate and seek expertise:** If the issue persists, seek help from experienced developers, software architects in the particular programming language or framework iam working with. Collaborate with others to gain different perspectives and insights.