Executive Summary

This report explores the implementation and evaluation of custom memory allocation algorithms, including First Fit, Best Fit, Worst Fit, and Next Fit, within a custom memory management system. The algorithms were benchmarked against the default system allocator to measure performance and identify potential trade-offs.

Key Findings:

1. Performance Analysis:
    - First Fit proved efficient due to its reduced search overhead.
    - Best Fit was slower, attributed to exhaustive searches for optimal memory blocks.
    - Worst Fit displayed similar inefficiencies to Best Fit
    - Next Fit suffered from a bug, leading to an infinite loop, underscoring the need for robust debugging practices.
2. Test Metrics: The benchmarks focused on allocation counts, total memory usage, and elapsed time, demonstrating the strengths and limitations of each custom algorithm compared to the system allocator.
3. Role of AI Assistance: AI tools significantly aided in understanding concepts and developing code for the project, despite occasional implementation issues that required debugging.

Overall, the report highlights the challenges and opportunities in designing efficient custom memory management systems and the value of integrating AI support for educational and development purposes.

Algorithm Implementation

The First Fit algorithm traverses the free list from the beginning and allocates the first block that is large enough to fulfill the memory request. Once a suitable block is found: If the block is larger than required, it is split into two: one part is allocated, and the remainder stays free. The search stops as soon as a match is found.

The Best Fit algorithm searches the entire free list for the smallest free block that is large enough to satisfy the memory request. This minimizes the leftover unused memory but may require more traversal time compared to First Fit

The Worst Fit algorithm allocates memory from the largest free block that can fulfill the request. This strategy aims to leave larger chunks of memory available, potentially reducing fragmentation.

The Next Fit algorithm is similar to First Fit but starts its search from the last allocated block instead of the beginning of the free list. If the end of the list is reached without finding a suitable block, the search wraps around to the start of the list.

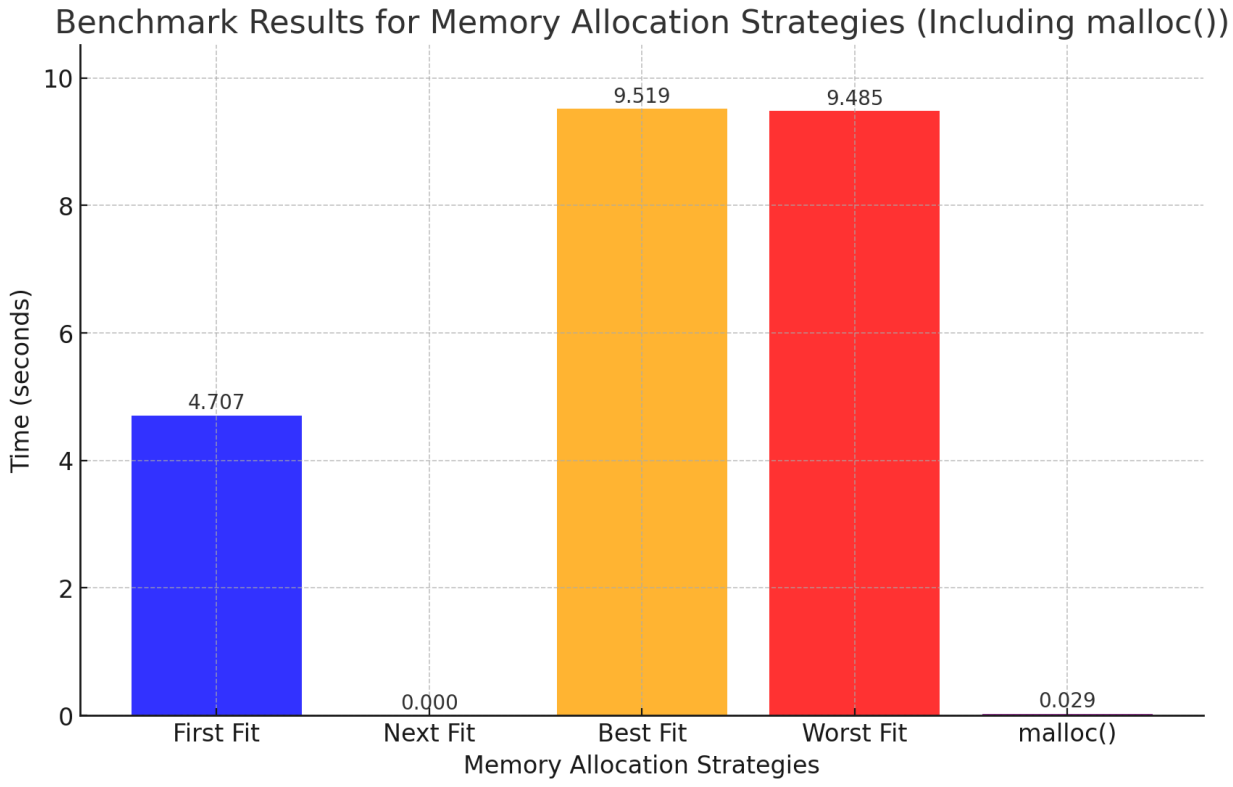
Test Implementation

The benchmark program evaluates the performance and correctness of the custom memory allocation algorithms (First Fit, Best Fit, Worst Fit, and Next Fit) implemented in the custom memory management system. When the program is run without linking it to the custom memory allocation implementation, it uses the system's default malloc() provided by the C standard library. The system call malloc() does not provide any metrics to track how many times it splits memory blocks, grows the heap, the heap fragmentation, or the max heap size. It would have been challenging to compute these metrics. Thus, only the elapsed time is measured.

Key steps in the implementation:

- The program performs NUM_ALLOCS (100,000) memory allocation requests.
- The maximum size of a single memory allocation is set to 1,024 bytes (MAX_ALLOC_SIZE).
- An array of Allocation structures is used to track allocated memory and its size.
- The program records the start and end times of the benchmark using the clock() function.
- The program prints the number of allocations, total memory allocated, and elapsed time.

Test Results



Interpretation of Results

The default system allocator is highly optimized for general-purpose use. Its speed demonstrates the effectiveness of system-level optimizations for handling memory allocations and deallocations efficiently

First Fit searches for the first free block that is large enough to satisfy the allocation request. The relatively faster time compared to other custom allocators suggests that it minimizes the search overhead compared to more complex strategies.

Best Fit finds the smallest block that is large enough to fulfill the allocation. The significantly slower time is due to the need to search through most or all free blocks to identify the best candidate, making it computationally expensive in scenarios with many allocations.

Next Fit operates similarly to First Fit but continues searching from the last allocated position instead of restarting from the beginning. The infinite loop indicates a bug or an issue with the implementation, highlighting the importance of debugging and thorough testing for custom allocators.
Worst Fit allocates from the largest available block, leaving the most space for future allocations. Its time is comparable to Best Fit because it also requires extensive searches, making it less efficient for scenarios involving many allocation requests.

Conclusion on AI performance.

Overall, my AI assistant was of great use. My assistant was useful when it came to explaining concepts such as heap management algorithms and the heap itself. It gave me guidelines on how I should implement coalescing, splitting, calloc(), and realloc(). The code it wrote was mostly fine, but it did not take into account small details which often resulted in segmentation faults . If I did not have my AI assistant, I probably would have learned more, but it would have taken longer. The AI assistant excels at explaining concepts and writes decent code. I could have learned more when it came to using pointers and accessing structures.