Custom Memory Allocation Implementation and Analysis

Executive Summary:

This report presents the implementation and analysis of custom memory allocation functions for a C program. The implemented memory allocation techniques include First Fit, Best Fit, Worst Fit, and Next Fit algorithms. These algorithms have been compared and evaluated based on various performance metrics such as the number of mallocs, frees, reuses, grows, splits, and coalesces. Additionally, the report provides an explanation of the test implementation, which demonstrates how the algorithms were implemented and executed.

Description of the Algorithms Implemented:

1. First Fit: This algorithm searches for the first available free block in the memory that can accommodate the requested size. It stops the search as soon as it finds a suitable block.
2. Best Fit: This algorithm searches the entire memory for the smallest free block that can accommodate the requested size. It aims to minimize internal fragmentation by choosing the block with the least wasted space.
3. Worst Fit: This algorithm searches the entire memory for the largest free block that can accommodate the requested size. It aims to leave larger blocks available for future allocations.
4. Next Fit: This algorithm searches for a suitable free block starting from the last allocated block. If it reaches the end of the memory, it wraps around to the beginning and continues the search. This approach aims to spread the allocations more evenly across memory.

The main function of the implementation is findFreeBlock(), which finds a suitable free block of memory based on the algorithm specified. The function takes the last block pointer and requests size as input parameters and returns the selected block based on the algorithm specified by preprocessor directives (FIRST_FIT, BEST_FIT, WORST_FIT, NEXT_FIT).

For Best Fit, the function iterates through the entire memory and finds the smallest free block that can accommodate the requested size. The Worst Fit algorithm is similar to Best Fit, but instead, it finds the largest free block. The Next Fit algorithm starts searching from the last allocated block and wraps around when it reaches the end of the memory.

Other functions in the implementation include growHeap(), splitBlock(), and coalesce(). growHeap() is responsible for requesting more space from the operating system when there are no suitable free blocks available. splitBlock() is used to split a block when the block found is larger than needed, and the leftover space is greater than the size of the block header plus 4 bytes. coalesce() is responsible for combining adjacent free blocks in memory to reduce external fragmentation.

Test Implementation:

The test implementation consists of 8 test cases, each designed to evaluate the performance of the five memory allocation strategies: system malloc, best fit, first fit, next fit, and worst fit. Each test case focuses on different aspects of the memory allocation process, such as simple allocation and deallocation, exercising malloc and free, block coalescing, block splitting, and realloc operations. Each test case is implemented as a separate function to keep the code clean and modular, making it easier to understand the purpose and focus of each test. To measure the execution time of each test, I used the 'clock()' function provided by the 'time.h' library.

The 'clock()' function is a standard library function that returns the processor time consumed by the program up to the point it is called. The value returned by 'clock()' is of type 'clock_t', which represents the number of clock ticks. To convert this into seconds, we divide the number of clock ticks by a constant 'CLOCKS_PER_SEC', which is also provided by the 'time.h' library.

Test Results:

The test results for all five candidates are as follows:

TEST NUMBER	SYSTEM MALLOC	BEST FIT	FIRST FIT	NEXT FIT	WORST FIT
1 (Simple malloc and free)	0.000062 sec	0.000037 sec	0.000032 sec	0.000043 sec	0.000039 sec
2 (Exercise malloc and free)	0.000599 sec	0.004286 sec	0.004047 sec	0.000402 sec	0.004211 sec
3 (Test coalesce)	0.000034 sec	0.000010 sec	0.000008 sec	0.000005 sec	0.000004 sec
4 (Block split and reuse)	0.000005 sec	0.000004 sec	0.000005 sec	0.000005 sec	0.000010 sec
5 BFWF (Best fit and worst fit)	0.000022 sec	0.000038 sec	0.000019 sec	0.000018 sec	0.000018 sec
6 Calloc (Calloc)	0.000009 sec	0.000025 sec	0.000012 sec	0.000011 sec	0.000007 sec
7 FFNF (First fit and next fit)	0.000020 sec	0.000059 sec	0.000025 sec	0.000018 sec	0.000011 sec
8 Realloc (Realloc)	0.000009 sec	0.000027 sec	0.000011 sec	0.000010 sec	0.000007 sec

Explanation and Interpretation:

1. **Test 1 (Simple malloc and free)** shows that all algorithms perform similarly, with only minor differences in execution time. This is expected as the test case is relatively simple and does not involve complex allocation patterns.

2. **Test 2 (Exercise malloc and free)** demonstrates that the best fit and worst fit algorithms take significantly more time than the other three. This is likely because these algorithms require searching through the entire free list to find the best or worst fit, while the other algorithms can allocate memory more quickly.

3. **Test 3 (Test coalesce)** highlights the efficiency of coalescing in all algorithms, as the execution times are relatively low across the board. This indicates that the implemented algorithms are able to merge adjacent free blocks effectively.

4. **Test 4 (Block split and reuse)** shows that all algorithms perform similarly, with only slight variations in execution time. This is expected, as the test case is designed to evaluate block splitting and reuse, which should not differ significantly between the algorithms.

5. **Test BFWF (Best fit and worst fit)** is designed to highlight the differences between the best fit and worst fit algorithms. The results show that the best fit algorithm is slightly slower than the worst fit algorithm in this test, possibly due to the additional overhead of searching for the smallest fitting block.

6. **Test Calloc (Calloc)** demonstrates that calloc performance is generally similar across all algorithms, with the system malloc showing slightly faster execution time than the custom implementations.

7. **Test FFNF (First fit and next fit)** is designed to highlight the differences between the first fit and next fit algorithms. The results indicate that the first fit algorithm is slightly faster than the next fit algorithm in this

test, as it is generally quicker to find the first available block that fits the request. The next fit algorithm may require more time to cycle through the free list, especially if the allocator has to wrap around to the beginning of the list to find a suitable block.

8. **Test Realloc (Realloc)** shows that realloc performance is relatively similar across all algorithms. The system malloc has a slightly faster execution time compared to the custom implementations, which could be attributed to the system malloc having optimizations specifically for realloc operations that are not present in the custom algorithms.

In summary, the test results show that each memory allocation algorithm has its own strengths and weaknesses, depending on the specific allocation pattern and use case. The system malloc performs consistently well across all tests, which is expected since it is optimized for general-purpose memory allocation. Among the custom implementations, the first fit and next fit algorithms perform well in most tests, with faster execution times than the best fit and worst fit algorithms, especially in cases involving more complex allocation patterns. The best fit and worst fit algorithms, while generally slower, may be more suitable for specific use cases were minimizing fragmentation or maximizing allocation efficiency is a priority.