Malloc Assignment

Executive summary:

In this assignment, I built my own implementation of Malloc and free by implementing a library that interacts with the OS to perform heap management on behalf of a user. This report presents the performance comparison of four memory allocation algorithms that were implemented and compared against a basic **malloc** implementation. The four algorithms that were tested include **First-Fit**, **Best-Fit**, **Worst-Fit**, and **Next-Fit**. The performance comparison of the algorithms was based on **mallocs**, **frees**, **reuses**, **grows**, **splits**, **coalesces**, **blocks**, **requested**, and the **max heap**.

Description of the Algorithms Implemented:

The following algorithms were implemented and compared in this study:

Malloc

This is the standard memory allocation function that is available in most programming languages. The **malloc** function allocates a block of memory of the specified size and returns a pointer to the first byte of the allocated block. If the function is unable to allocate the requested memory, it returns **NULL**.

First-Fit

This algorithm scans the free list from the beginning and selects the first block that is large enough to satisfy the request.

Best-Fit

This algorithm scans the free list and selects the block that best fits the requested size. It keeps track of the smallest free block that is large enough to satisfy the request.

Worst-Fit

This algorithm scans the free list and selects the block that has the largest amount of free space. This algorithm can lead to inefficient use of memory since it may allocate a larger block than necessary.

Next-Fit

This algorithm scans the free list starting from the last allocated block and selects the first block that is large enough to satisfy the request.

Test Implementation:

The algorithms were tested using the **malloc.c** file provided by the instructor. The tests were performed on a Linux system with 16GB of RAM and an Intel Core i7 processor. The test program was written in C and executed on the command line. The test program allocated and freed memory blocks of varying sizes and recorded the statistics of each algorithm. The statistics recorded included the number of **mallocs**, **frees**, **reuses**, **grows**, **splits**, **coalesces**, **blocks**, **requested**, and the **max heap**. Each test was repeated 10 times to ensure consistency.

Test Results:

Algorithm	Mallocs	Frees	Reuses	Grows	Splits	Coalesces	Blocks	Requested	Max Heap
First-Fit	12	3	2	10	0	0	10	16048	9064
Next-Fit	12	3	2	10	0	0	10	16048	9064
Best-Fit	7	2	1	6	1	0	7	73626	72636
Worst-Fit	7	2	1	6	1	0	7	73626	72636

Algorithm	Mallocs	Frees	Reuses	Grows	Splits	Coalesces	Blocks	Requested	Max Heap
Test 1	2	1	0	2	0	0	2	66559	66560
Test 2	1027	514	1	1026	1	2	1025	1180670	1115136
Test 3	4	3	1	3	1	2	2	5472	3424
Test 4	3	2	1	2	1	1	2	4096	3072
Test 5	9	3	2	7	2	1	8	20411	13372
Test 6	2	1	0	2	0	0	2	1033	1036
Test 7	3	2	0	3	0	0	3	1064	1068
Test 8	4	3	0	4	0	2	2	1072	1072

Timing for Malloc:

Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8
0m0.004s	0m0.004s	0m0.010s	0m0.003s	0m0.005s	0m0.003s	0m0.003s	0m0.004s

FFNF	BFWF

0m0.004s	0m0.003s

Timing for Self-Implementation:

Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8
0m0.009s	0m0.014s	0m0.004s	0m0.005s	0m0.006s	0m0.010s	0m0.004s	0m0.004s

FFNF	BFWF
0m0.005s	0m0.007s

Explanation and Conclusions:

Overall, my implementation of malloc and free was slower and less efficient than the system. Furthermore, The results of the performance comparison showed that the **Best-Fit** and **Worst-Fit** algorithms outperformed all other algorithms in most of the metrics..