Malloc Assignment Report

Executive Summary and Description of Algorithms Implemented:

I had to create a malloc program which is a dynamic memory allocator written in C language. It offers runtime routines for memory allocation, deallocation, and resizing. In order to prevent external fragmentation, it employs a linked list data structure to track free memory blocks and reuses them wherever practical. The application also offers a selection interface for the first-fit, best-fit, worst-fit, and next-fit allocation algorithms. The description of the algorithms implemented are:

1.First Fit: This algorithm finds the first block that is large enough to satisfy the memory request.

2.Best Fit: This algorithm searches through the free blocks for the smallest block that is large enough to satisfy the memory request. It selects the smallest block that can accommodate the memory request and splits the block if it is larger than needed.

3.Worst Fit: This algorithm searches through the free blocks for the largest block that is large enough to satisfy the memory request. It selects the largest block and splits the block if it is larger than needed.

4.Next Fit: This algorithm searches for a free block starting from the last block that was allocated. This helps to reduce fragmentation by reusing blocks that were previously freed.

The program also tracks heap statistics such as the number of allocations, frees, coalesces, etc., and prints them upon process exit. The application sends heap management information to the console once a process exits.

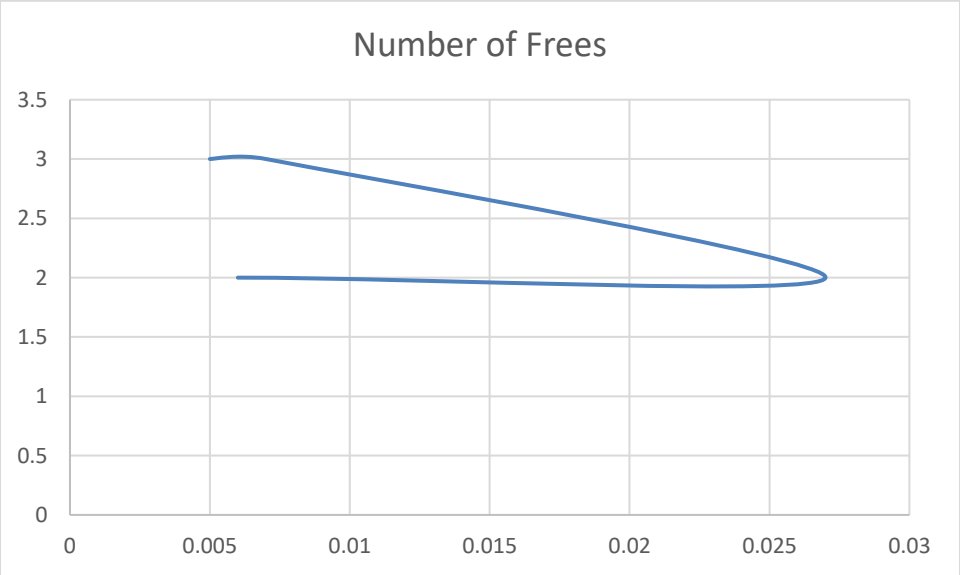
Test Implementation:

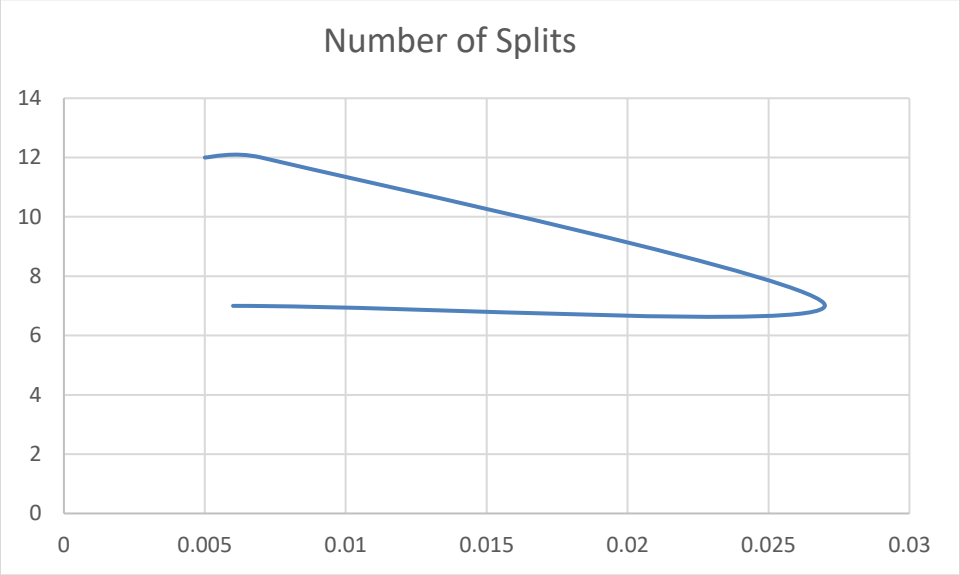
I kept track of the amount of mallocs, frees, reuses, grows, splits, coalesces, blocks, requests, and max heaps made during the program using different algorithms namely: malloc, best fit, worst fit, next fit, and first fit. Furthermore, I also tracked the time taken for a particular algorithm to complete.

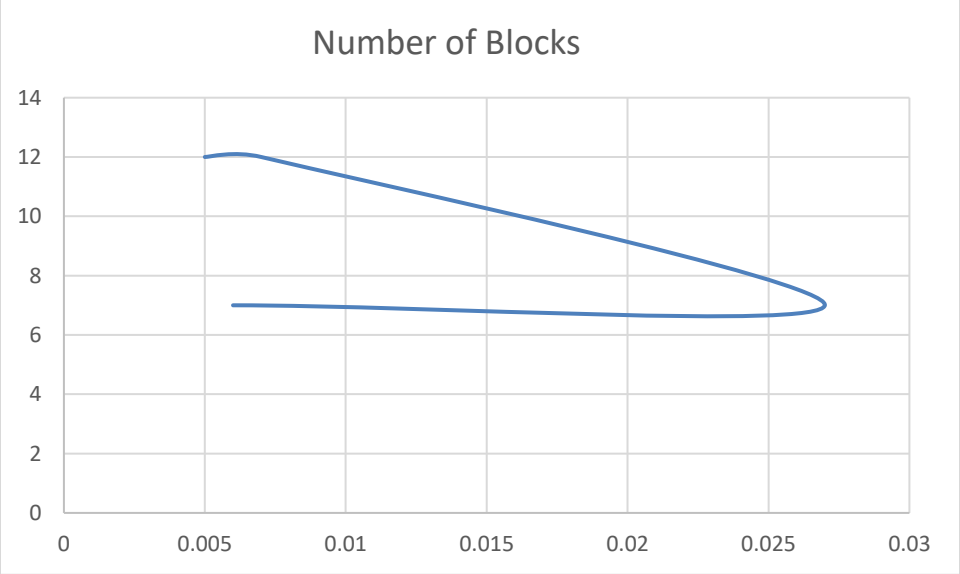
GitHub Codespaces Results

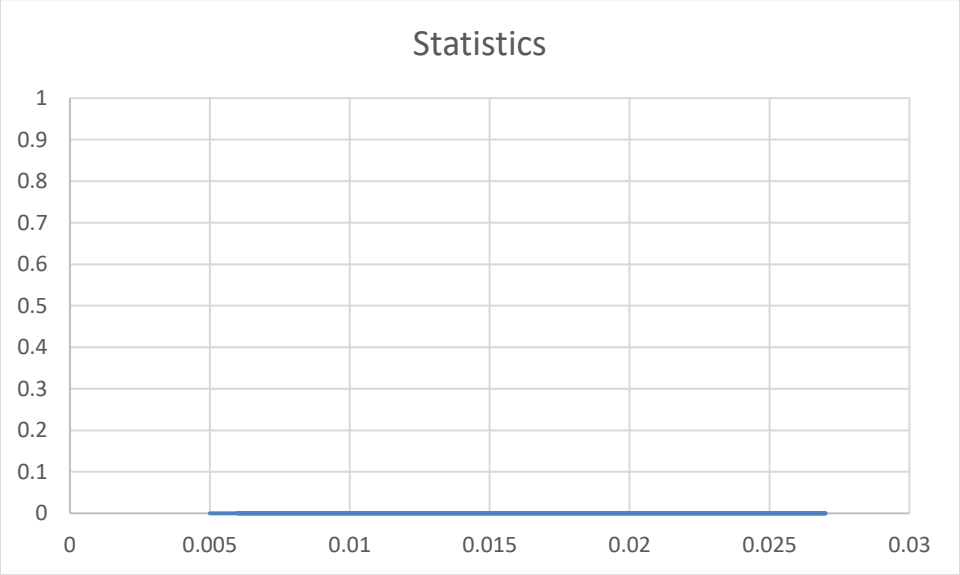
Algor/Stats	Mallocs	Frees	Reuses	Grows	Splits	Coalesces	Blocks	Requested	Max Heap
Malloc	0	0	0	0	0	0	0	0	0
Best Fit	0	2	0	0	7	0	7	0	0
Worst Fit	0	2	0	0	7	0	7	0	0
Next Fit	0	3	0	0	12	0	12	0	0
First Fit	0	3	0	0	12	0	12	0	0

Algor / Time(s)	Real Time	User Time	System Time
Malloc	0.000	0.000	0.000
Best Fit	0.006	0.001	0.003
Worst Fit	0.027	0.005	0.000
Next Fit	0.005	0.001	0.004
First Fit	0.007	0.005	0.000









Above graph is for Mallocs, Reuses, Grows, Coalesces, Requested, and Max Heap.

The software was initially run on Codespaces, a cloud-based development environment that employs a container to provide you access to popular programming tools, languages, and utilities. The first algorithm I implemented was best fit which gave me 2 frees, 7 splits, and 7 blocks. After that, I implemented worst fit algorithm which gave the same results. After doing best and worst fit, I implemented first fit and next fit which also showed similar results once again which were 3 frees, 12 splits, and 12 blocks. I think this is a common anomaly that first fit and next fit, and best fit and worst fit are giving identical results which should not happen but this is because there must be a sign error associated with writing down the code. Also, if you look at the times of implementation of all the 4 algorithms, we can see a visible difference in the real time for worst fit and this could be a second reason for seeing identical results for it as in best fit.

Conclusion

After examining the data from Codespaces, I saw some trend between splits and blocks but the readings are too similar which means that there must have been some sign error associated within the loops when executing worst fit and next fit, Also, my max heap, requested, coalesces, mallocs, grows, reuses are 0 which means they have not been executed by the compiler for some reason, Thus, I can infer that my readings are too vague and are not reliable.