Ethan Sprinkle
Student ID 1002002357

Executive Summary

When allocating heap memory, there are several ways that memory can be allocated from previously freed blocks of memory. However, depending on the implementation used, there are tradeoffs between each of the different methods. In this report, the standard system call malloc() is compared with a first fit, next fit, best fit, and worst fit implementation using heap management statistics and performance metrics.

Algorithms Implemented

The four memory allocation methods implemented were first fit, next fit, best fit, and worst fit. First fit would return the first block of free memory found starting from the beginning of the heap. Next fit performed the same task, but rather than starting from the beginning of the heap, would start from the most recent allocation and loop around until it returned to the original starting point. Best fit would check the entire heap and return the free block closest to the size requested, meanwhile worst fit would return the largest free block. Additionally, if a free memory block was returned that was larger than the requested size, the memory block was split, so long as the new block had enough room for a header plus at least 4 bytes of size.

A free() method was also implemented that, given a pointer to an allocated block of memory, simply marked the block as free. After a block was freed, any two consecutive free blocks were coalesced into one block of contiguous memory. Calloc() simply called malloc() and initialized the values to 0 using memset(), and realloc() utilized memcpy() to copy the data from an initial pointer to a newly requested block of memory called by malloc().

Testing

Three different test algorithms were used to compare heap fragmentation. In each algorithm and implementation, the heap was first fragmented by calling malloc() 500 times and then freeing every other block of memory. Additionally, the memory initially requested with each call to malloc was increased to create more variety in the size of memory available.

After fragmenting the memory, malloc was called another 200 times to see how many blocks of data were reused as well as the total size of heap after allocation. The first test requested memory in increasing amounts, the second test requested memory in consistent amounts, and the third test requested memory in decreasing amounts.

The number of grows, reuses, splits, and blocks were used with max heap size and number of blocks to gauge heap fragmentation and memory utilization. The “time” terminal command was also used to measure performance time using a fourth testing program as the initial three ran too quickly to measure any noticeable changes in performance.

Ethan Sprinkle
Student ID 1002002357

Test Results

Table 1.1 & Figure 1.2 Heap Management Statistics for Increasing Malloc Requests

	First Fit	Next Fit	Best Fit	Worst Fit
mallocs	701	701	701	701
frees	250	250	250	250
reuses	200	200	200	188
grows	501	501	501	513
splits	187	187	187	186
coalesces	0	0	0	0
blocks	688	688	688	699
requested	622024	622024	622024	622024
max heap	514048	514048	514048	528304

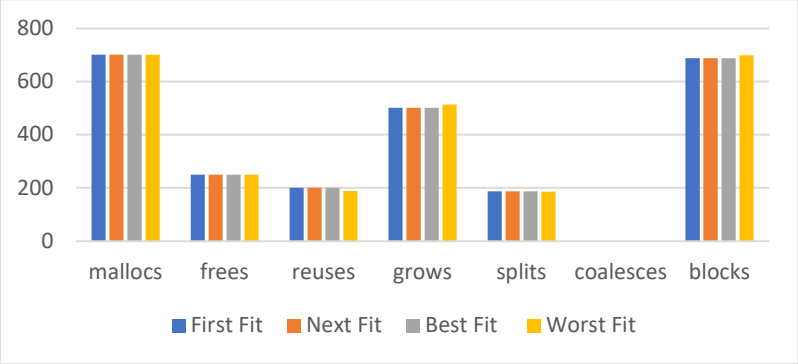

Table 1.3 & Figure 1.4 Heap Management Statistics for Consistent Malloc Requests

	First Fit	Next Fit	Best Fit	Worst Fit
mallocs	701	701	701	701
frees	250	250	250	250
reuses	125	200	125	125
grows	576	501	576	576
splits	122	122	122	122
coalesces	0	0	0	0
blocks	698	623	698	698
requested	702024	702024	702024	702024
max heap	590848	514048	590848	590848

Ethan Sprinkle
Student ID 1002002357

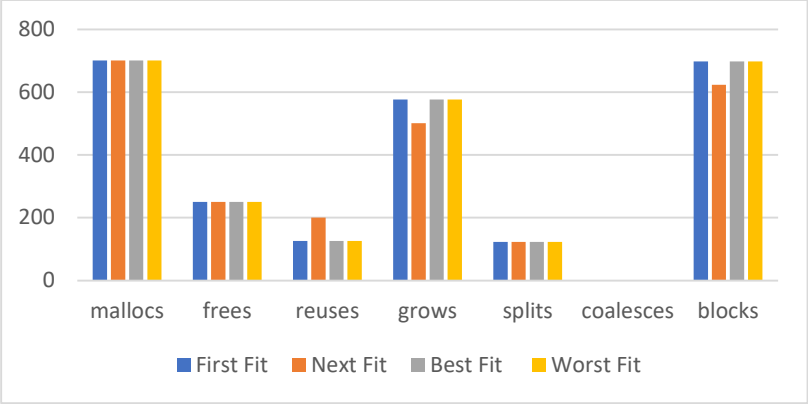

Table 1.5 & Figure 1.6 Heap Management Statistics for Decreasing Malloc Requests

	First Fit	Next Fit	Best Fit	Worst Fit
mallocs	701	701	701	701
frees	250	250	250	250
reuses	200	200	200	200
grows	501	501	501	501
splits	28	195	27	200
coalesces	0	0	0	0
blocks	529	696	528	701
requested	633224	623224	623224	623224
max heap	514048	514048	514048	514048

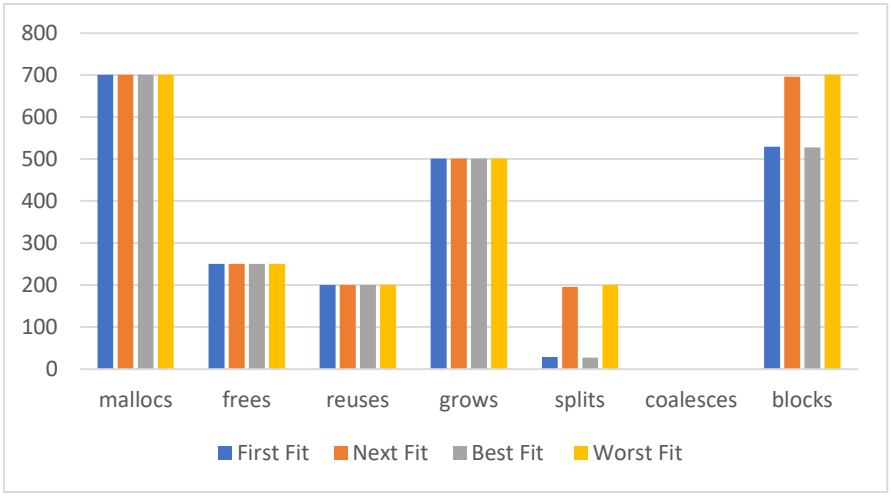

Table 1.7 Performance Times

Standard malloc()	First Fit	Next Fit	Best Fit	Worst Fit
0.024 sec	2.401 s	2.479 s	2.405 s	2.475 s

Ethan Sprinkle
Student ID 1002002357

Interpretation of Results

In the first test case, the worst fit algorithm seemed to have the worst performance, having to request more grows than the other algorithms and reusing less existing blocks. This can likely be due to how the memory blocks were fragmented. By filling up the larger memory blocks first, as the malloc requests grew larger, there were fewer free memory blocks large enough to meet the requested size, hence the more grows and more total memory blocks.

In the second test case, Next Fit had more block reuses than any of the other algorithms, which resulted in a smaller max heap size overall. A possible explanation for this occurrence could be which memory blocks are split during reallocation. Since next fit will continue to point at blocks in increasing in size, when larger memory blocks will continue to be split, with the leftover size large enough to be reused by the next malloc() request.

In the third test case, first fit and best had way fewer splits and block than next fit and worst fit, but the total max heap remained constant. In this case, while the total number of memory stayed the same across all cases, there was less fragmentation in the first fit and best algorithms. This makes sense since the memory blocks at the end of the heap are larger than the ones near the beginning, so first fit will also act like the best fit algorithm for decreasing malloc requests, resulting in fewer splits.

Since it is difficult to see inside the standard system call to malloc() for heap splits and reuses, the “time” terminal command was used to compare runtimes of a program that called malloc 10000 times, freed every other block of memory, called malloc another 200 times with increasing size requests, then freed all remaining pointers, totaling 10,200 calls to both malloc and free. The performance time for the standard system call to malloc was significantly faster than any of the other algorithms, which were all within 0.1 seconds of each other. The large increase in performance is likely due to more efficient search and allocation algorithms than the ones made for this report.

Conclusion

Although the standard system malloc proved to be significantly more efficient and optimized than the search algorithms created for this assignment, the utilization of benchmark statistics provide a better insight into the strength and weaknesses of each of the implemented algorithms. The first fit algorithm is the simplest of the four and requires the least amount of overhead, but time performance can suffer if the memory block list becomes long with few calls to free. Next fit requires more overhead to implement to keep track of the most recent allocation, but performance time can decrease depending on the order of how memory is freed and malloc is called again. Best fit seems to be the most efficient algorithms by using the smallest available blocks of memory, but if the remainder blocks that split are too small to be utilized by other malloc calls, there ends up being more heap fragmentation overall. Conversely, worst fit can reduce heap fragmentation by leaving more blocks of large memory available for future malloc calls.