Rudy Orozco
Prof. Trevor Bakker
CSE 3320 - 003
03/18/2024

Malloc Report

Introduction

With there being different memory searching algorithms for searching through different memory blocks, there are both benefits and drawbacks to each algorithm due to how they search for a memory block. With that, I have created two different test codes that allow me to see the performance and differences with each algorithm. There will be our first test that will allocate a pointer array in large quantities and iterations to check for runtime performance and then a simple multiple malloc program that will check how each algorithm behaves. This gives us a way to compare how each algorithm performs with each other.

Malloc and Algorithm Performance Benchmarking

With the first test, this will let us see how each algorithm performs in regards to how long it will take to run. There will be two different runtimes but the same amount of iterations. The program will malloc 50,000 pointers with 5 different predetermined malloc sizes and free all 50,000. It will run this again but will malloc 5 different sizes than before and will have a higher total combined amount of mallocs. To get accurate results, each algorithm is run 3 times to get an average runtime for the initial and second time mallocs.

fig. 1 initial runtimes

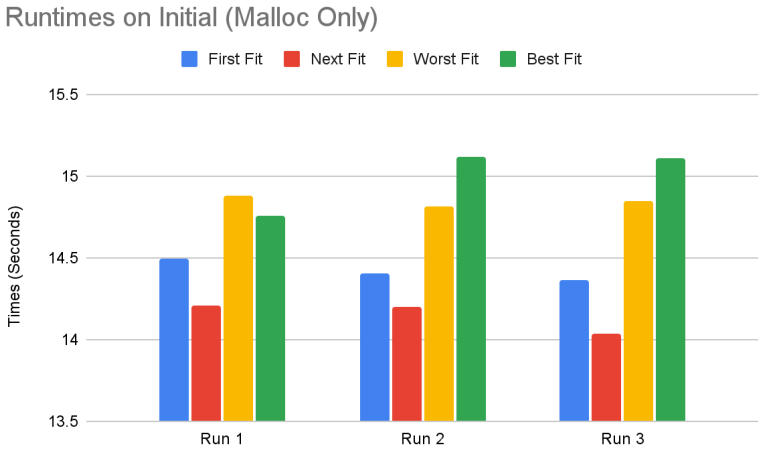


fig 2. Second Runtimes

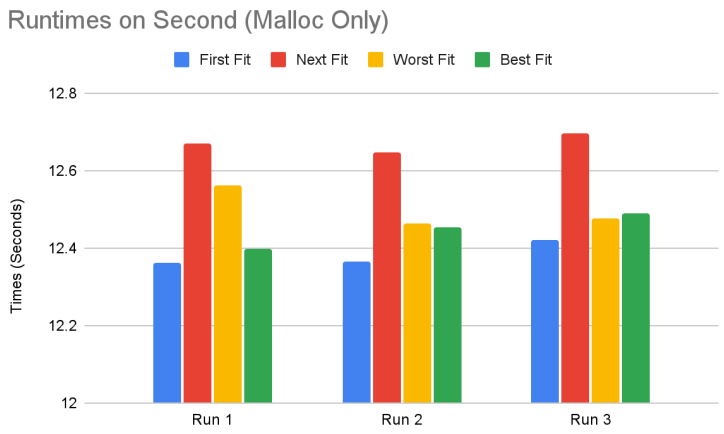
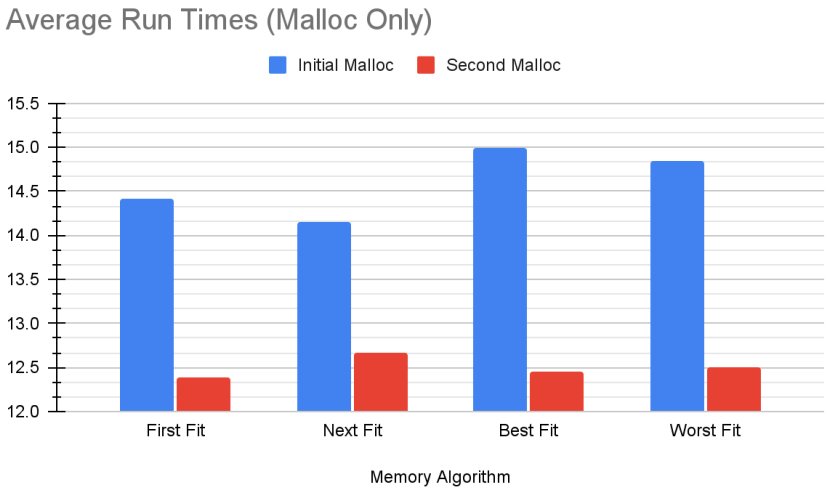


fig 3. Average Runtimes



Starting with the first initial runtimes, the one thing to note is that the first initial run takes a bit longer than the second run to do. From what I can think of why this happens is because of the fact that the heap has to grow as there were no initial memory blocks made. By the time the second run comes around and all the previous memory is free, there are already blocks in the heap that can be filled and with that, the shorter of a time that needs to be taken to complete this process.

With the initial malloc, the fastest overall would be the next fit algorithm and the slowest would be best fit. The assumption that next fit is fastest is possibly due to the fact that when malloc is called, there hasn't been a free called yet in this part of the program so it still starts at the end of the list or where it stopped last so it doesn't have to start at the beginning like the other algorithms. My assumption on why best fit is the slowest in the initial malloc is possibly due to the fact that it not only has to search through the entire list to find an available block, it

also has to find the best block that can fit the requested size while being the smallest.The complexity of the algorithm makes it the slowest out of the 4 algorithms.

With the second malloc, the overall fastest algorithm that ran is first fit while the slowest would be next fit. My overall assumption with why the first fit would be the fastest is possibly due to the fact that the algorithm ignores other cases like if the block is the least or the greatest or where the current pointer is at. It is also the simplest implementation of the algorithms out of the three so there is less overhead that is needed to run. With first fit having to locate the first immediate available free block that is able to fit the requested size and not to locate through the entire list, it doesn't take too much time to assign. My assumption on why the next fit takes too long, is that it may be due to not only the complexity of the implementation of the code but how it searches for the ideal memory block. Next fit not only starts at the location it stopped but if there isn't an available memory to fit the requested size, it has to go through the entire list until it reaches where it had started initially and then grow the heap.

One thing to note is that I had exclusively left out the default malloc function until now because of the performance it has compared to the others. When running with the default malloc, the runtimes of malloc were instantaneous compared to the other algorithms.

fig 4. Initial Runtimes for default malloc

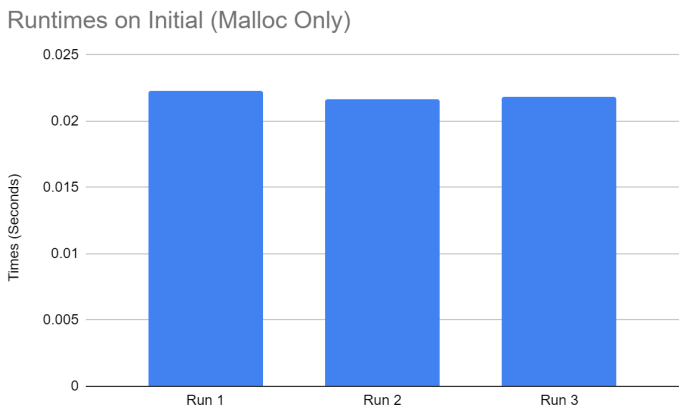
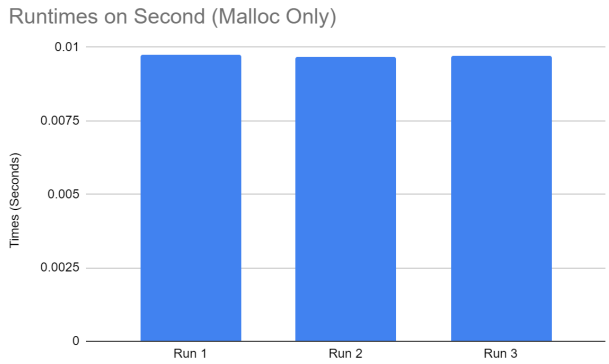


fig 5. Second Runtimes for default malloc



The same behavior between the initial and second runtimes are almost identical to what we saw with the other algorithms. The initial mallocs took longer while the second time run was faster. Though, it takes less than a second to malloc memory, which is almost instantaneous for it to iterate through all 50,000 pointers. The main assumption that I can make about this is due to the fact that malloc is already integrated within the libraries of the C language and has been optimized to run efficiently and does not need any additional code within which can add to the overhead and decrease performance of the function.

Algorithm Behavior Benchmarking

This second test does a controlled malloc and freeing in a determined area with various malloc sizes. This code is to check how the algorithm deals with splitting, coalescing, and the behaviors of memory searching. At the bottom are the results that have been seen.

fig 6. Behavior of algorithms

	First Fit	Next Fit	Best Fit	Worst Fit
mallocs	18	18	18	18
frees	17	17	17	17
reuses	1	1	1	3
grows	17	17	17	15
splits	1	1	1	3
coalesces	6	6	6	5
blocks	17	17	17	15
requested	25872	25872	25872	25360
max heap	5000	5000	5000	5000

The one thing I have noticed with this these numbers is that with First, Next, and Best fit is that they have the same exact numbers and it may be due to misplacement of the count but one thing to note is that worst fit has slightly different results than the others, as it has a couple more reuses, less coalesces and blocks and even the amount of memory requested.

From what I can think of why this is the case is possibly due to the fact that the worst fit utilizes how the memory is chosen in which it chooses the largest available memory available and then splits. The resulting split ends up leaving a bigger remaining portion of the memory block still available to use. In a sense, worst fit may be efficient in dealing with memory and minimizing usage of creating more unnecessary memory. But with the potential that this may be inaccurate due to the results being the same and a possible misinput in the code, these are my assumptions based on these results.

Conclusion

Overall, these algorithms can be useful in different use cases and scenarios. First fit may be good for initial mallocs and possibly any mallocs that may not require freeing up memory, but in other cases like worst fit can be useful to minimize wasted memory. With the behaviors, the assumptions may be inconclusive due to possible errors in the code but overall, there are clear visual differences in regards to runtime and with the tests being at 50,000 iterations, if there were to be a case where there maybe more iterations, the differences in runtimes may be more wider and the use of a certain algorithm may matter more.