CSE 3320: Malloc Implementation Vs System Malloc Report

Bryan Funes
Trevor Bakker
CSE 3320-003
12/01/2025

Executive Summary

Objective:

Implemented a malloc program with 4 search algorithms, realloc, and calloc. Test this program against system malloc to see how system malloc compares to my four implementations. Report my findings with an essay format, graphs, and tables.

Algorithms:

The four algorithms implemented were First, Best, Worst, and Next fit. First fit looks for the first available free block that fits the size it's looking for. Best and Worst look through the whole heap finding a free block with either the smallest size that can hold the size requested or the largest size that hold the size requested. Next fit is like First fit, but it starts from the last allocated block to search for.

Test Procedure:

I made four programs to test my four implementations against system malloc. These programs make all forms of memory allocation and deallocation and prints the speed taken to run the program in tics. I ran each of these programs five times for each implementation, to consider edge cases. I saved the data on a table and made a graph for each program.

Findings:

I found that First Fit was the most consistent algorithm that works. Best fit and Worst fit were slower to account for searching through the whole heap to find the block. Lastly Next Fit was a double-edged sword where for 3 out of my 4 programs it ran the slowest and in one it ran almost on par to system malloc. System malloc blew away all the competition, because of years of optimization.

Understanding:

First fit was the most consistent because it doesn't search the whole heap and when a block is found it just returns it. Best and Worst fit get bogged down by having to search the whole heap list for the block it's looking for but helps fragmentation. Next fit helps in certain cases dependent on how memory is allocated and how it is positioned given that it starts from the previous allocated block and can either find the block quick or takes a while.

Conclusion:

First fit was the best overall algorithm that I implemented, since it can be mostly good for any case. We didn't beat system malloc, but we did get close with Next fit showing it is possible. There are probably better ways to do this test, but this experiment perfectly showcases the info clearly for our understanding. I now understand what's going on in the background when I call malloc in my programs.

CSE 3320: Malloc Implementation Vs System Malloc Report

For the past two weeks I've been working hard on my own implementation of malloc. Something that seemed impossible in retrospect doesn't seem too daunting now and was much easier than I once thought before. With this came multiple tests on my own to see how my own malloc stands up to the malloc in our regular C program. This report will showcase the findings between system malloc and my own malloc with the four algorithms implemented, to see how they are stacked against one another.

Benchmark 1

Runs	System	First Fit	Best Fit	Worst Fit	Next Fit
1	1630	29976	38074	32903	44112
2	1467	30304	31679	31020	44858
3	1433	29884	30942	32075	42871
4	2157	29440	30881	31581	44635
5	1488	30192	30440	31576	44206

Table 1:Data from my test on program 1

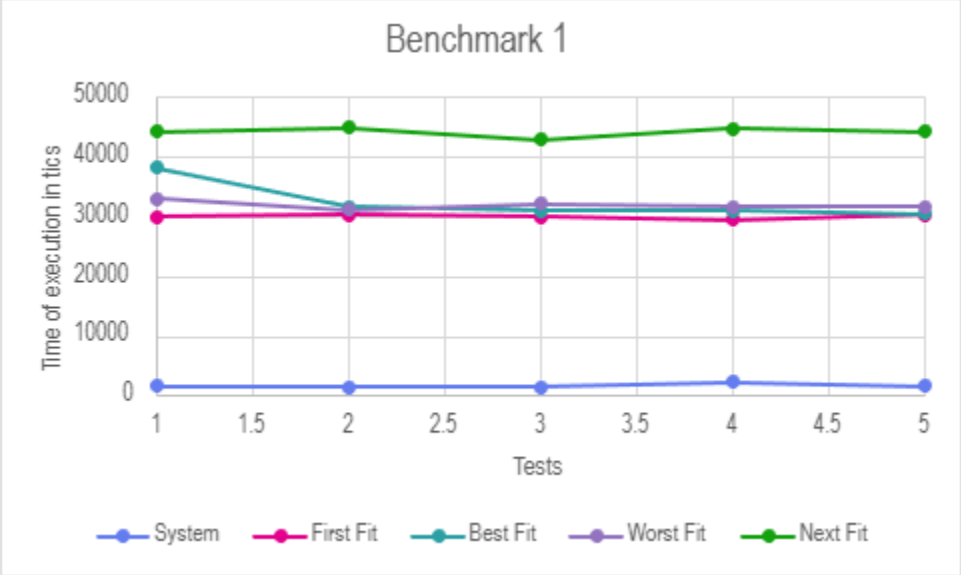

Figure 1:Graph of table 1

To begin, let's explain the algorithms in my own implementation of malloc. This malloc implementation has four algorithms First, Best, Worst, and Next fit. First fit will go through the heap and find the first free and large enough block. Best fit will go through the entire heap and find the smallest free block that the requested memory fits, Meanwhile Worst fit finds the largest free block that the requested memory can fit. Lastly, Next fit is like First fit but it starts off from where the last allocation was. We will use these different algorithms to go up against system malloc in various tests.

Benchmark 2

Runs	System	First Fit	Best Fit	Worst Fit	Next Fit
1	446	4410	4265	4434	5485
2	423	4428	4653	4762	5243
3	461	4080	4928	5026	5199
4	474	4403	4343	4312	5450
5	427	4240	4323	4371	5396

Table 2:Data from my test on program 2

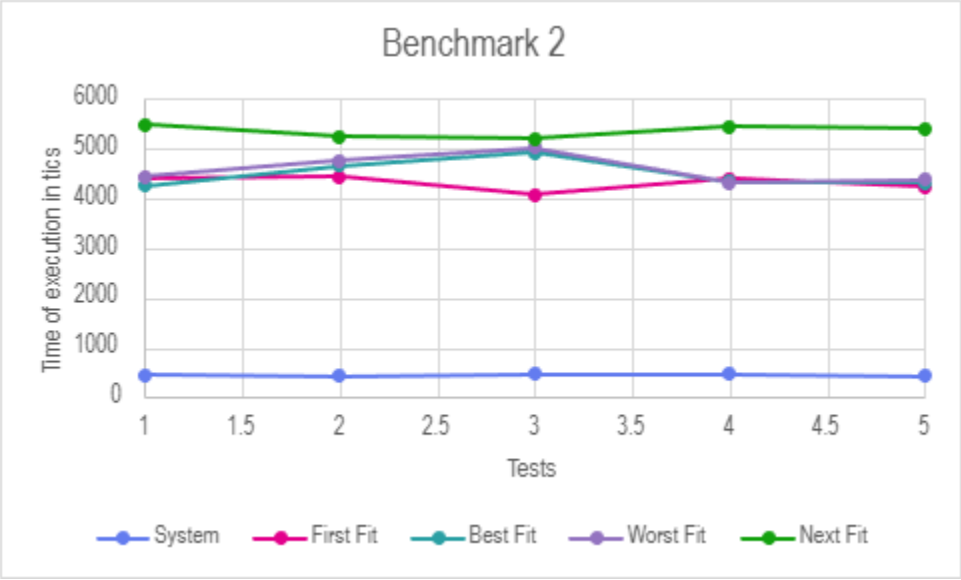

Figure 2:Graph of table 2

Up next let's explain my tests. I made four programs that make a bunch of allocation and deallocations and print out the time running in tics. I get the time by subtracting the end time by

the start time to get the number of tics it took to run the program. I ran each program using system malloc five times and ran each algorithm in my malloc implementation 5 times. This was done so edge cases are avoided and to see around how long it took for it to run on average. I recorded this data on tables for each test separately along with graphs, as seen throughout this report.

Benchmark 3

Runs	System	First Fit	Best Fit	Worst Fit	Next Fit
1	138	12176	12712	12047	806
2	136	12028	12137	12175	837
3	139	12107	12126	12074	856
4	138	12252	12125	12384	841
5	136	13994	12224	12211	1201

Table 3:Data from my test on program 3

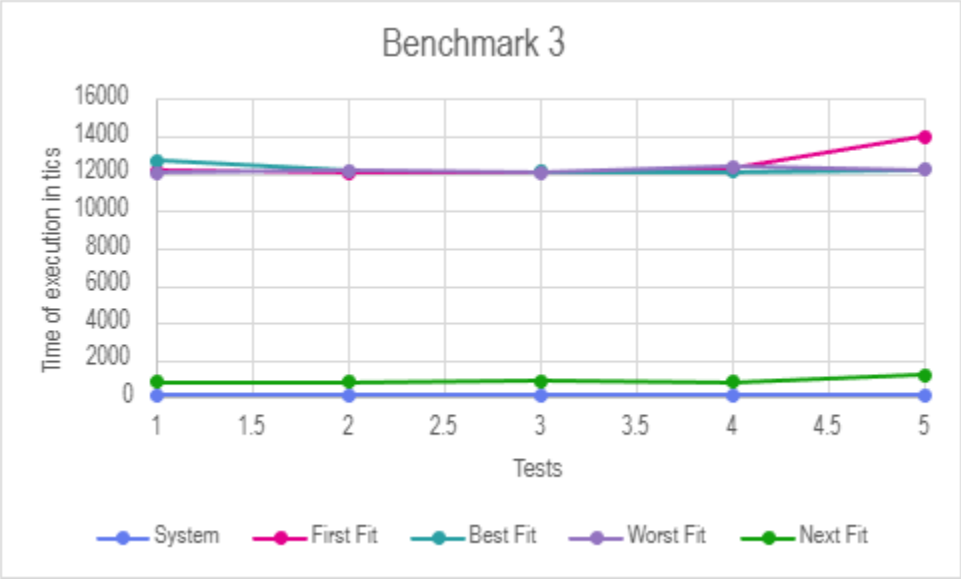

Figure 3:Graph of table 3

Now let's talk about the results. To start with, system malloc beat all my implemented algorithms by a far mile. Makes sense given that malloc has been optimized for many many years meanwhile I've been working on mine for 2 weeks, so there is no surprise there. Other than that, for all the tests, by far first fit was the best algorithm when it came to speed. It was always

the most consistent algorithm no matter what test I threw at it. It makes sense since best and worst fit need to look through the whole heap to find the block meanwhile first fit doesn't. My tests have a lot of blocks, so it doesn't help in that case, and I know that if I made a smaller heap and a smaller program, best and worst fit wouldn't be too far from first fit and would thrive more. Nevertheless, all the algorithms were fairly close to one another and not too surprising except next fit.

Next fit on tests 1, 2 and 4 ranked the slowest of all the implemented algorithms and there was no contest but test 3 was different. Next fit in test 3 was super close to the runtime of system malloc throughout all its runs. To make it more surprising it was the close's any of the algorithms were to the systems malloc speed. Now you may be wondering why? Well next fit as mentioned previously starts from the last allocated block and moves from there like first fit, until it finds a block or reaches a full loop. This saves time so you don't have to search the entire heap each time like best and worst fit, while shifting the starting point to attempt to make it easier unlike first fit. This doesn't work all the time, and it depends how the memory is allocated and how it's moved to get optimal performance. My tests show this perfectly, because we see in 3 tests how next fit did not do any benefit and actually hindered more the search by making probably way too many full heap searches. Meanwhile, in test 3 it probably kept shifting the starting position really close to an available block, which saves time and gives the needed boost to almost reach the level of system malloc.

Benchmark 4

Runs	System	First Fit	Best Fit	Worst Fit	Next Fit
1	900	7446	7626	7861	12306
2	852	7744	7802	7687	12837
3	871	7822	7738	7644	13945
4	867	8162	7616	7608	12392
5	939	7772	9752	7628	13414

Table 4:Data from my test on program 4

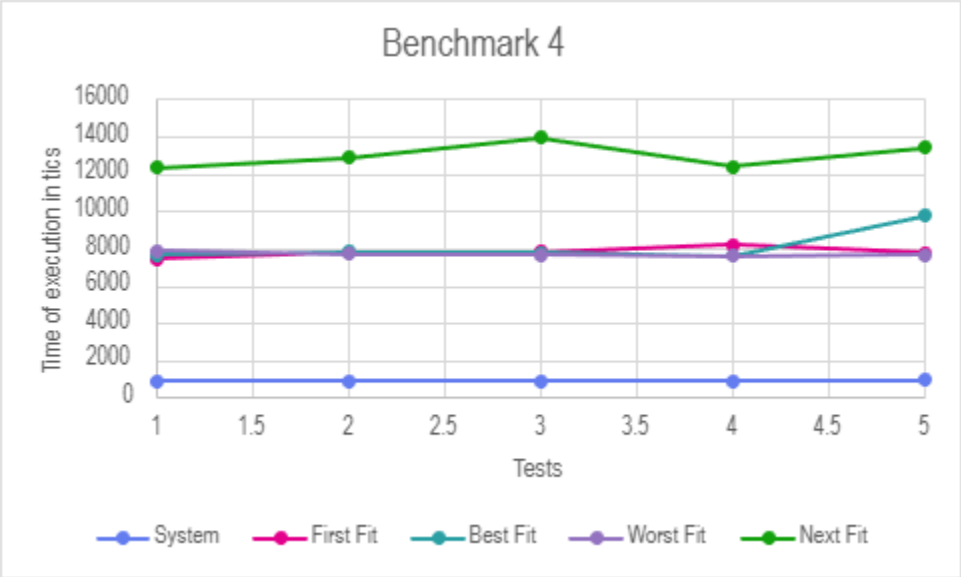

Figure 4:Graph of table 4

In conclusion, the best overall algorithm was first fit. First fit was on average the fastest of all the algorithms and was the most consistent. Best and worst fit were fairly average, given that it needs to search the whole heap to find their specific block. Meanwhile next fit depends on the allocation of the blocks and data, since it can be the slowest or can be the fastest algorithm implemented. I can see more intensive testing being done, but this shows pretty clearly the speeds of each and their benefits and drawbacks.