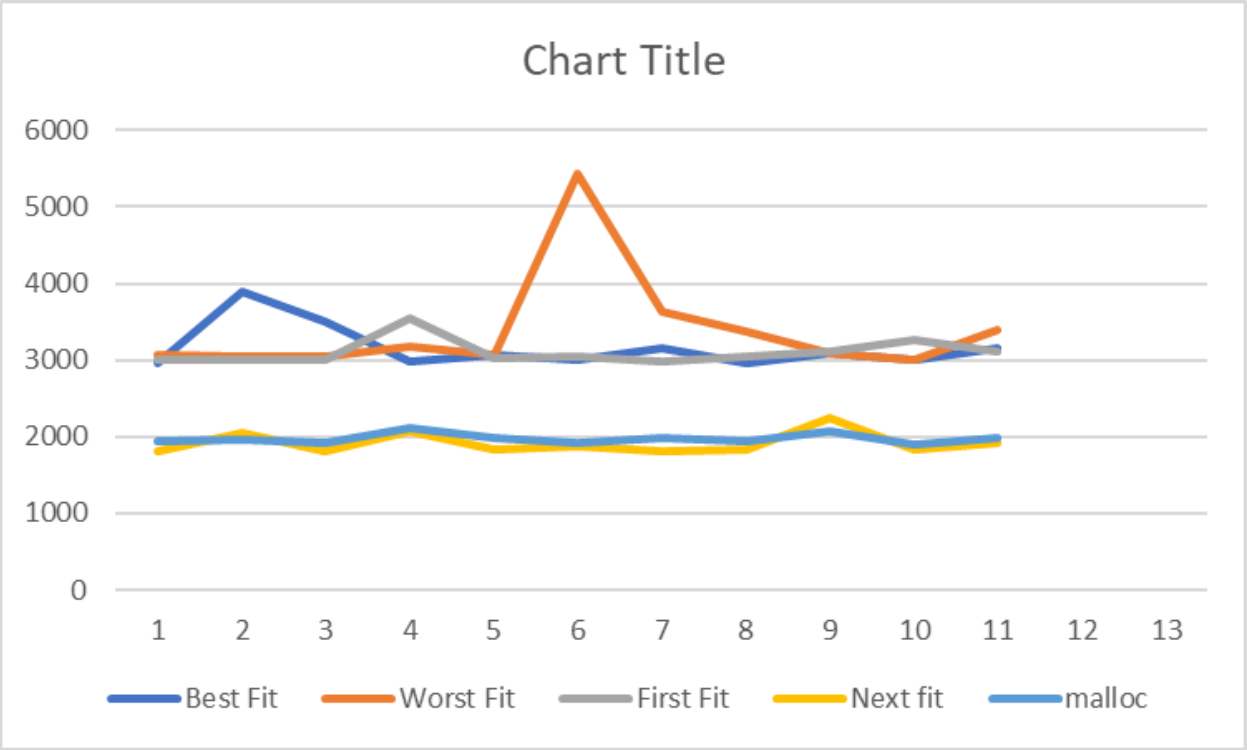
Andy Ouyang
1001934803

Executive Summary for arena allocator

Description of the project: This project involves implementing four memory implementation algorithms: Best Fit, Worst Fit, Next Fit, and First Fit and comparing it against the standard C library call malloc from the stdlib library. The project involves four functions to be implemented. The first function is memalloc_init, which allocates memory for the entire memory block on the arena through a call to malloc. The memory block allocated from this function is what we will be manipulating with our fitting algorithms. The second function is memalloc_alloc. This function allocates memories within the memory block allocated earlier based on the different fit passed in by the user in memalloc_init that is stored in a global variable. For Best Fit, we iterate through the list to find the block that will yield the least amount of spaces wasted. For Worst Fit, we iterate through the list to find the block that will waste the most space. First Fit is quite simple, it simply looks for the first block that is able to house the memory block. Next Fit is similar to first fit, except that it starts from the index of the previous allocation and only returns to the beginning only if it could not find a block big enough to house the memory. Next up is memalloc_free which passes in a pointer to a block that is to be freed. If two consecutive frees are called, then it would merge the blocks into one singular block. However, you could not use the function free() for this part. Lastly is memalloc_size. This simply checks the number of nodes that are present and returns an integer. Aside from these, there are also five benchmark files. These benchmarks are all identical with the exception of the library call malloc. A beginTimer and endTimer variable is imported from the <sys/timer> library and it captures the runtime of each program in microseconds. Each program will call memalloc_init with an arena size large enough to house all the memalloc_alloc calls that follow with their respective FIT. An array of 1000 char pointers is declared and then goes into a loop that allocates 1000 times for a memory size of 10,000. After, the char pointer array goes inside another loop that will iterate through and free all the blocks. Same thing happens with malloc except that it does not use any of the four functions implemented. After running each program 10 times, BEST FIT had an average runtime of 3161.3 microseconds, WORST_FIT had an average runtime of 3388.8 microseconds, FIRST_FIT had an average runtime of 3101.9 microseconds, NEXT_FIT had an average runtime of 1918.1 and Malloc with an average of 1975.1 microseconds. After careful examination and validation, it is clear that Next_Fit is the best algorithm here as it has a significantly faster run time than best, worst, next fit, and a slight advantage over malloc.

BENCHMARK/DATA

Algorithm	Result #1	Result #2	Result #3	Result #4	Result #5	Result#6	Result#7	Result #8	Result #9	Result #10	Mean
Best_fit	2970	3899	3500	2974	3059	2994	3158	2968	3093	2998	3161.3
Worst_fit	3058	3039	3041	3168	3071	3428	3628	3361	3088	3006	3188.8
First_fit	3004	2997	3009	3542	3016	3055	2975	3041	3118	3262	3101.9
Next fit	1812	2058	1820	2081	1832	1872	1802	1836	2235	1833	1918.1
Malloc	1947	1953	1929	2104	1979	1929	1980	1948	2079	1903	1975.1



The strategy for these benchmarks is that I get two variables from the sys/time library and I mark one variable as beginning time before I call all the memory allocation and loops. I then call the for loops for memory allocation and another for loop that calls for freeing each node. Both of these loops run for 1000 times each. After the loops finish, I mark the endTime and then store it in a double variable which will print the duration of this process in microseconds. As for the performance testing, I made sure that the memalloc_init has enough memory to house all the allocations so it has a size of (number of allocations) * (number of bytes for each allocation). The loop is iterated 1000 times for the sake of accuracy. After, it goes into another for loop to be freed 1000 times. I believe this is succinct enough to get a runtime of the tests.

GRAPH/TABLE INTERPRETATION:

The run time of each algorithm is recorded ten times and given in table format. The numbers are then added together and divided by the number of tests to get the average. The average is what will be used to test against other algorithms. If a certain run time is abnormally higher than usual, then it will not be considered. For example, if Best Fit was to run at about 3000 microseconds give or take, but one occurrence happens at 5000 microseconds, that single occurrence will be ignored. After the testing, Best Fit has an average runtime of 3161.3 microseconds, Worst Fit has 3188.8 microseconds, First Fit has 3101.9 microseconds, Next Fit has 1918.1 microseconds, and malloc has 1975.1 microseconds.

Best Fit, Worst Fit, and First Fit have very similar performances of around 3000ish microsecond. However, Next Fit ran surprisingly fast at 1918.1 microseconds, even faster than the traditional system call malloc 1975.1 microseconds. To ensure the accuracy, I ran Next Fit for another long period, and it yielded similar performance. After careful speculation, the reason due to this occurrence is possible due to my implementation. I implemented Next Fit using an array of nodes. One advantage of this method is that it can insert and search at any given index whereas for the traditional linked list you would have to iterate through the whole thing. This approach ensures a very low time complexity, hence the reason for its speed.

CONCLUSION:

Again, looking at the average run times of each algorithm, it is clear that Next Fit is the winner here as its average runtime of 1918.1 seconds tops Best, Worst, and First Fit and has a slight edge over Malloc.