Name: Viraj Sabhaya
ID: 1001828871
Name: Jose J Aguilar
ID: 1001128942

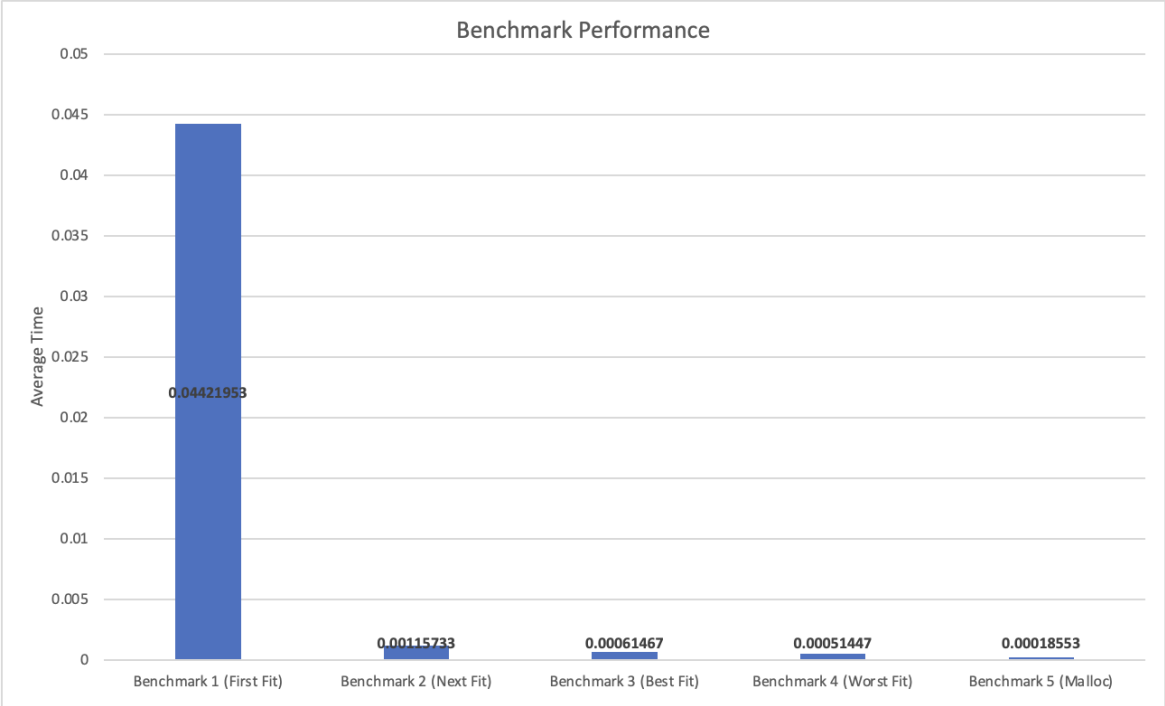
Arena Allocator Report

The focus of this project assignment was to write and test 5 different algorithms to allocate memory from the initial allocation of a pool arena. We attempted to create 5 algorithms as follows, First-Fit, Next-Fit, Best-Fit, Worst-Fit and malloc. To test each algorithm, we wrote a benchmark for each algorithm and kept record of the compile time for that benchmark using the clock function in the time.h module/library of type clock_t. Each benchmark was compiled 15 times to keep a record of average time of compile time. The results showed First Fit being the worst algorithm while worst fit and malloc both came close to being the best algorithm to use with times of 0.00051447 and 0.00018553, respectively.

The benchmarks 1,2,3 & 4 which tests the First-Fit, Next-Fit, Best-Fit, and Worst-Fit respectively all starts by defining an array and then initializing that array (unsigned char * type like arena) according to the individual benchmarks. To initialize each benchmark the function mavalloc_init (pool arena, gArena), where the first parameter is the size of the pool arena, and the second parameter is the name of the algorithm to be tested by this benchmark. Then a for loop is ran to allocate 1000 blocks of 100 bytes. Followed by another for loop to free all the memory in the array. Further 4 blocks of ptr 0,1,2, &3 are allocated 10000 bytes each. Now two blocks of allocation are freed and another 2 blocks of 500 bytes are allocated. For benchmark5, after allocating a large chunk of memory, arena is initialized and then a for loop is ran to free the memory. Now the same process is repeated for malloc benchmark like the other benchmarks of allocating 4 blocks each of 10000 bytes.

Benchmark 1	Benchmark 2	Benchmark 3	Benchmark 4	Benchmark 5
0.042844	0.000333	0.000633	0.000644	0.00017
0.036339	0.00056	0.001089	0.000506	0.000174
0.041003	0.00133	0.000283	0.000446	0.000393
0.041111	0.000484	0.000494	0.000604	0.000174
0.048627	0.000883	0.000262	0.000518	0.00012
0.038231	0.000585	0.000421	0.000302	0.000175
0.048766	0.000494	0.000483	0.000422	0.000078
0.048342	0.001864	0.000287	0.000444	0.000108
0.038391	0.006619	0.000467	0.000545	0.000181
0.054033	0.000329	0.000222	0.000339	0.000144
0.043005	0.000686	0.000542	0.000382	0.000185
0.041819	0.000643	0.000394	0.000661	0.000128
0.053734	0.001402	0.002884	0.000242	0.000173
0.040337	0.000846	0.000221	0.000953	0.000431
0.046711	0.000302	0.000538	0.000709	0.000149
0.04421953	0.00115733	0.00061467	0.00051447	0.00018553

Name: Viraj Sabhaya
ID: 1001828871
Name: Jose J Aguilar
ID: 1001128942



For the First-Fit algorithm we initially ran a for loop to transverse the array. If the LinkedList array type was hole, meaning that location was empty, and no process allocated and in use, and the size passed in was less than the overall size of the arena we enter the if statement. Inside the if statement we calculated the new arena which equals the current arena plus the size and the leftover size of the allocation LinkedList[0].size -size. Those two values would be passed in the function insert node and then to insert node internal. The insert node internal function would shift and make space for the new node insertion. We return the value -1 if it failed, if not we returned the arena of that certain index that passed the conditions and set the type to P meaning process allocated. The average time of compilation from our First-Fit algorithm was 0.04421953 seconds. The highest out of all the algorithms. This meant the insert node function failed to split blocks correctly.

The Best-Fit algorithm used the same initial implementation as First-Fit algorithm with the use of the for loop and the if statement. If our current linkedlist size was smaller than the size attempting to be allocated, then we would return null. We would then calculate the current leftover size and enter the if state statement. If the current leftover size was smaller or equal to previous leftover size, we would insert the current leftover space into previous leftover size variable. After the for loop got to the end of the link list the previous leftover size would contain the smallest value possible necessary for Best-Fit. We would then return the linkedlist arena where the leftover size was the smallest and change the type to P, process allocated. We left the implementation of insert node out of best fit because it wasn't splitting the blocks correctly and increased the number of errors. The Best-Fit algorithm was the third best algorithm with an average time of 0.00061467 seconds.

Name: Viraj Sabhaya
ID: 1001828871
Name: Jose J Aguilar
ID: 1001128942

The Worst-Fit algorithm used the same implementation as Best-Fit. The difference came in the if statement where we tested if current leftover size was bigger than previous leftover size. If it passed, we would save that value into previous leftover size. We also implemented the insert node function but according to errors we received the blocks once again weren't splitting correctly. We then proceeded to return the linkedlist arena for that index and set it to process allocated. The Worst-Fit algorithm came with the second fastest time of 0.00051447 seconds.

The Next-Fit algorithm used the same implementation as First-fit. The only difference is it started from last used node and checks using the if statement if the node has a Hole, is in use and if the size of the arena is lesser than the size that needs to fit in. If the size is bigger that the Allocated size it returns NULL. The average time of Next-Fit is 0.00115733 seconds the fourth best time. According to the errors in compilation the Next-Fit algorithm was incorrectly splitting the blocks causing it to run slower.

From our end results we concluded that the following things: Firstly, our insertNode was not splitting the blocks correctly for First-Fit, Best-Fit, Next-Fit and Worst-Fit so, it resulted in deviating results. This made malloc the best allocation algorithm in terms of the least average time taken after Worst Fit. Secondly, the first fit was not able to reuse the correct node. It couldn't find the block that was already used once for storing the allocated result. Lastly, the next fit picked wrong node which is due to the incorrect return value from the insertNode function. The nodes weren't split from P and H.