Dov Kassai and Nathaniel Arussy
CS 214 Systems Programming Asst1

Malloc and Free Improvement

The code we have written is a smaller version of the malloc and free functions that already exist in c. Before this can happen there are a few steps that need to be implemented. The first one being is that we have a macro that will replace all the mallocs called in the program with mymalloc, and the same is done with the free and myfree. This is so that when the program is run, it is with the code that we have written, and not what was already in c. Another point is that we have initialized an array of 4096 to the size of the array that we are dealing with in the program, since that is the max amount of memory space we are dealing with.

The way our mymalloc function works is by first checking if the array has had any values placed into it. This is done so that we know whether or not we are able to traverse our memory array by following our metadata blocks, explained later. We check that the array has any metadata by checking the first two elements of the array for a designated value that we put as the value of those locations when we begin to place data into the array. If the requested data does not fit, we do not initialize values for our array at all, and leave the first two blocks untouched. Once we are ready to place in our first data (we check by comparing space of array without the metadata and without our magic number to requested size), we input a metadata block into locations 2 and 3 of the array, and place metadata after the allocated space. Our metadata contains a significand in the first bit of our total 16 bytes between both elements of the array that the metadata is contained in, and the last 12 bits are used for the number of elements being allocated for. The significand tells us whether or not the following data is being used or not. The way we assign each of these values into our metadata, is by placing bit values (in binary) that are higher than 256, and the significand, into the first element, and place all bit values below 256 into the second element of our metadata. Once the array has any metadata blocks in it, we can simply traverse the array by going through and finding when an open block of space, or multiple blocks, are free to be overwritten. We do this by checking the metadata for the significand and how much space it contains. When we find the space, we create metadata blocks at the beginning, make metadata blocks after the allocated space if there is more than 2 elements leftover, and return a pointer to the element of the array immediately after the metadata for that allocated space.
Myfree just goes to the location inputted, and if it isn't already freed, it changes the significand to 0.

RUNTIMES:

For the Test case A we got a run time of: 0.2 microseconds.
For the Test case B we got a runtime of: 0.24 microseconds
For the Test case C we got a runtime of: 0.163 microseconds.
For the Test case D we got a runtime of: 0.175 microseconds.
For the first custom test case, which was test E, we got a runtime of: 0.13 microseconds
For the second custom test case, which was test F, we got a runtime of: 0.3 microseconds