In this assignment, we were required to create our own implementation of malloc() and free(). For our metadata struct, we wanted to be space efficient so we used a char (isFree) to store whether a block of memory is free (denoted with 'y') or not free ('n'), and a short (size) to store the amount of data being asked to be allocated. As a result, this gives us a metadata size of 3 bytes (due to padding, I think, sizeof(struct metadata) returns 4 bytes instead). Mymalloc() design: for malloc(), first we check if the request is less than 0 or greater than 4096 bytes - the size of the metadata structure if so, the function returns an error message with the coreesponding file and line that the error occurrs. Then the function checks if mymalloc() has been run before by having a pointer point to the beginning of the static array and if it has been run before then the pointer should be pointing at a metadata structure. If it was the first time run then the requested byte size is allocated and then the pointer is returned. Otherwise a metadata pointer iterates through the entire static array until it either finds a big enough block of memory to fit the request which allocates the memory and return a pointer otherwise it reaches the end of the array then the request could not be satisfied and then returns NULL. Myfree() design: For free(), we were required to implement a more "intelligent" version of it. First, we check if the pointer is not NULL (and return, otherwise). Then, we needed to move the pointer to point at the metadata instead of the block of memory. After that, we can check the three cases. (A) To check if the address is not a pointer, we check if the pointer is within the range of the our char array (myblock[0] <= pointer >= myblock[4095]). (B) Then, we make sure that the pointer was allocated by malloc() and not by the user by checking each metadata block in myblock and seeing if pointer is there. (C) Lastly, we make sure that the pointer has not been already free()'d. Once these cases are checked, we can free the metadata pointer. After all the above is completed, we iterate through myblock and check for adjacent blocks that are both free using two metadata pointers. If they are, the previous pointer will gain the size of the free block in front of it, and the size of the metadata.

Memgrind results: Average runtime for testA: 13 microseconds Average runtime for testB: 88 microseconds Average runtime for testC: 23 microseconds Average runtime for testD: 23 microseconds Average runtime for testE: 1 microseconds Average runtime for testF: 1 microseconds