The basic design of how we store each block of memory in the global array is having 4 bytes of metadata followed by however many bytes of memory needed to be allocated. The first two bytes of metadata is dedicated to the magic number. This is to properly identify both where blocks of memory actually start and end and also to identify if a given block of memory is allocated or free to be allocated. Because the numbers were stored in a char array, we thought it would be easier just to store and compare the ascii values of predetermined chars in the metadata slots. The chars for unallocated memory were ‘U’ and ‘A’ while allocated memory would have the values ‘A’ and ‘L’. The next two bytes of metadata contain how long the given block of memory actually is. We decided the most efficient way to hold the size is with the ascii values of chars, since that would make it two bytes as opposed to a four byte int. The first char would hold the thousands and hundreds digits of the block size, and the second char would hold the tens and ones digits of the block size. Converting the size back and forth between chars and ints is still relatively convenient because of the many helper methods we use.

The workloads A through E test that our malloc and free functions work properly and efficiently. Workload A loops 150 times, where in each iteration we malloc 1 byte of memory and immediately free it. Workload B mallocs 50 1-byte chunks in a row, while storing each returned pointer in an array. Once it reaches 50 allocated bytes, we free the 50 pointers one by one, and then we repeat the whole process until we’ve malloced 150 bytes. Workload C includes randomizing whether we malloc 1 byte or free it until we’ve allocated 50 times, which then we free the rest of the pointers in the array. Whenever we free, the most efficient way was to free the most recent pointer returned, so that we optimize the amount of contiguous available space. Doing it this way also helps us keep track of how many pointers are remaining in the array. Workload D randomly chooses between a randomly sized malloc or freeing a pointer, until we reach 50 malloc()ed times. The random allocation size is between the range of 1 and 64 bytes, so the workload will never exceed our total memory capacity, because our metadata is size of 4 bytes. Worst case: (64 + 4) * 50 = 3400. Workload E first allocates 512 4-byte blocks to fill up our static array (because 512 4-byte metadata blocks). Then we free the last two blocks of memory, and reallocate the combined size(+ metadata size). We repeat this process until we have one 4-byte metadata block, and one big block of size 4092. No errors should arise when testing this, but it tests that our free function works as it should. Workload F tests cases which should produce errors, like freeing a pointer that has already been freed, but catches them gracefully. Overall, workloads A, C, and D take an average runtime of 8-10 microseconds, B takes an average runtime of 80 microseconds, and E and F take longer (4790-4850 microseconds) because they test a lot more operations than the other workloads do. Overall, they’re all really fast workloads, but E and F are slightly slower compared to the rest, but they are not significantly slower because they take a lot less than a second.