Structure:

We used a linked list structure, with size, free and pointer to the next node. This took up 16 bytes in our array. Size holds the value of how much memory the current block is taking up. Free is used to check if the block is free (1) or not free (0). Next points to the next node in the memory.

Malloc:

When malloc is first called, it calls the init() function that initializes the array. It sets it up so that there is space for the metadata, which is 16 bytes, and is followed by a free block of memory that is 4080 bytes long. When malloc is called again, it will call the split function. In this function, the memory is split up to accommodate memory. It identifies a free block of data, checks if it has enough space, and if it does splits it up. The result of one call of malloc asking for 1 Byte looks like this: metadata (16 Bytes), 1 byte of user data, metadata for the free block (16 Bytes), and the free block (4063). Repeated calls of malloc will add them likewise.

The Malloc function recognizes multiple errors. It will indicate if you are trying to allocate a negative size of memory, more than the maximum amount of memory which is 4080, or if there is no space to allocate more memory.

Free:

In the free function, it checks if the passed in address has been previously allocated. It will throw an error if the user is trying to deallocate something twice or something that has not been assigned. It will then set the block free. Defrag is then called within the free function. It checks to see if there are any consecutive free blocks. If it finds any, it will combine both of them and set the whole block free. If it does not find any, defrag will not perform anything to the block of memory.

Test Results:

From the test results, we can see that F takes the most time to complete. This is because when the free function and defrag function is called, it will run through the entire array each time to absorb memory. Part D is the next one that takes the most time, and it follows the same reasoning. This is one area where this code can be improved. We sacrificed program efficiency for increased memory efficiency. We could have gone for a doubly-linked list, which will perform the free and defrag function faster but will take up an additional 8 bytes of memory.

One run of memgrind:

Average A: 298.000000 microseconds

Average B: 6648.000000 microseconds

Average C: 1284.000000 microseconds

Average D: 185692.000000 microseconds

Average E: 34703.000000 microseconds

Average F: 702980.000000 microseconds