Based on individual values of *'i'* in the for loop
Preliminary memory leak value: 1762 bytes lost
After best fit + doubly linked list: 1423 bytes lost

```
Total bytes lost: 1762

Heap end address: 0x564560fc3cf5
```

```
Total bytes lost: 1423

Heap end address: 0xb730dc1
```

Report
I calculated memory leaks by calling my function *check_memory_leak()* which traversed through the heap and checked if the block was freed or not, after *free()* is called. If not, then it would increment an int for the number of bytes lost and return this value.

I converted the single linked list implementation by adding a previous pointer to the struct, which would allow the heap to traverse backwards. I implemented this mechanism in the find_free_block() function, which would be able to go backwards and forwards through the heap depending on where the current pointer is located. I also had to change the *request_space()* function to allow the current block to have a pointer to the previous block.

For the best fit algorithm, I also changed the *find_free_block()* function to allow it to find the best block for the size that it is trying to give. I checked if current has a value stored and if the size of the block being created was larger than the one of the current block. If it was, then it has the possibility to fit the block the best, so it checks if the current block's size is less than the best and if it is then it will change the current to the best. It also updates the pointers so that the next block will become the next current. More specifically, it traverses through the available memory blocks for the smallest block which is large enough to accommodate for the one that is being requested.

Issues
I did not test for traversal of a doubly linked list, however I believe that the implementation is correct. I don't think that my calculation of checking memory leaks is correctly calculating because I believe that there is an issue with the blocks that I call and their size. This could be an issue because I am calling the same size for each malloc, and it would probably return a different value if I used a random integer value. This would impact my test for the bonuses. I also believe that there could be a problem with my *realloc()* and it working with my driver function.

Bonus
I implemented a *split_block()* function which checks if the size is greater than the requested size and if it is, then it will split the block. After this it updates the metadata of the block with the appropriate values.

If the new block has a next block, the function updates the previous pointer of the next block to point to the new block. This ensures that the linked list of memory blocks is properly maintained after the split. I call this in *malloc()*, so that the most appropriate size is accommodate for the block being created.

While I did not call malloc and free with random sizes, I implemented a merging mechanism in when *free()* was called. The function checks if the previous memory block in the heap is free. If it is, the function merges the current memory block with the previous memory block by updating the size and pointer fields in the metadata blocks. The function then checks if the next memory block in the heap is also free. If it is, the function merges the current memory block with the next memory block by updating the size and pointer fields in the metadata blocks.