Samiur Rahman
Nosheen Adil
ECE 254

Lab 4 Report

Problem Statement

Memory allocation is one of many responsibilities of an operating system. The manner in which memory is allocated is based on the operating system itself, which typically implements a specific memory allocation algorithm. Based on the unique operation of each algorithm, the goal is to determine how each memory allocation algorithm compares to one another, specifically the best-fit and worst-fit algorithms.

Relevant Data Structures

As both the best-fit and worst-fit memory allocation algorithms are closely related, we decided to model the structures required to implement each algorithm based on one model struct, which appears as the following:

```
struct Memory_Allocation_Algorithm_Struct {
    void* memory_box;
    void* free_space;
    int* bit_array;
    int num_blocks;
};
```

- where memory_box is a void pointer that points to the starting address in memory that was allocated to experiment with each algorithm. Upon initialization of this memory space, the initialization function declares a control block, which is available for reference by other functions, but not available as free space;
- where free_space is a pointer to the actual location of memory that is available to the user upon the memory space initialization (this excludes the control block);
- where bit_array is a pointer to an array of bits that is used to represent the blocks in memory; and
- where num_blocks is an integer value that holds the number of blocks in memory that are currently allocated.

Using this model struct, two global structs were declared for each algorithm, Best and Worst.

```
struct Memory_Allocation_Algorithm_Struct Best;
struct Memory_Allocation_Algorithm_Struct Worst;
```

Key Algorithms

1. *Bit Manipulation Functions*

We implemented three key algorithms for setting, resetting, and testing the values of each bit in the bit array:

```
void SetBit(int array[], int index)
```

Sets the value of the bit at the current index by performing a logical OR with a 1 bit shifted by the remainder of the index divided by 32 (32 bits per block as each block is 4 bytes).

```
void ResetBit(int array[], int index)
```

Resets the value of the bit at the current index by performing a logical AND with a 1 bit shifted by the remainder of the index divided by 32.

```
int TestBit(int array[], int index)
```

Returns the value of the bit at the current index by performing a logical AND with a 1 bit shifted by the remainder of the index divided by 32 and checking if the value is equal to 0. If it is equal to zero, the function returns a one, otherwise, it returns a zero, indicating the value of the bit at that index.

2. *Initialization Algorithms*

The logic for both algorithms is identical; the only difference is that each function uses the struct corresponding to the algorithm that it was meant for. These algorithms can be described using the following pseudocode:

```
if size is less than size of block (less than 4 bytes) return -1
else allocate memory of size passed in argument

calculate number of bits (denoted by n) required to represent number of blocks allocated
set first n blocks and refer to this as the control block of the memory space
reset remaining blocks and refer to this as the free memory space

set the address of the free memory space as the beginning of the initialized memory space with an offset of the
control block and return 0
```

3. *Allocation Algorithms*

```
void *best_fit_alloc(size_t size)
```

The best-fit allocation algorithm attempts to allocate the smallest block of memory in the available memory space that is large enough to accommodate the memory request, hence the name “best fit,” in hopes of using appropriate-sized memory blocks to limit external fragmentation. It can be further described using the following pseudocode:

```
if size is less than the minimum allocation size (4 bytes = 1 block) or greater than the entire memory space return null

init all flags and counters to zero except best contiguous block size variable to
INT_MAX

while in memory block {

    if bit is not set (equal to zero) AND contiguous flag is not set {
        set contiguous flag and store current index as beginning of contiguous block
    }

    if bit is not set (equal to zero) AND contiguous flag is set
        increment contiguous block size counter

    if (bit is set (equal to one) AND contiguous flag is set) OR reached end of the memory space {
        reset the contiguous flag
        if contiguous block size counter is less than current best contiguous block size AND the difference between the
contiguous block size counter and the number of blocks requested for allocation is greater
        than zero {
            set best contiguous block size to contiguous block size counter value
            store index of start of the best contiguous block
            reset contiguous block size counter to zero
        }
    }
}

if best contiguous block size variable is still equal to INT_MAX
    not enough space available to accommodate allocation request, return null

if best contiguous block size variable is greater than 1 + the requested allocation size
```

```
    set best contiguous block size variable to 1 + requested allocation size

while in best contiguous block
    mark all blocks as allocated by setting each bit in the bit array

assign first block of requested memory with the size of memory that was allocated for the request

return a pointer holding the address to the first available block of memory in the newly allocated memory
block (excludes first block holding the quantity of memory allocated)
```

```
void *worst_fit_alloc(size_t size)
```

The worst-fit allocation algorithm attempts to allocate the largest block of memory in the available memory space that is large enough to accommodate the memory request, in hopes that whatever memory remains afterwards is still large enough to be of use, and ultimately reduce the quantity of external fragmentation. It can be further described using the following pseudocode:

```
if size is less than the minimum allocation size (4 bytes = 1 block) or greater than the entire memory space return null

init all flags and counters to zero

while in memory block {

    if bit is not set (equal to zero) AND contiguous flag is not set {
        set contiguous flag and store current index as beginning of contiguous block
    }

    if bit is not set (equal to zero) AND contiguous flag is set
        increment contiguous block size counter

    if (bit is set (equal to one) AND contiguous flag is set) OR reached end of the memory space {
        reset the contiguous flag
        if contiguous block size counter is greater than current worst contiguous block size AND the contiguous
        block size counter is greater than the number of blocks requested for allocation {
            set worst contiguous block size to contiguous block size counter value
            store index of start of the worst contiguous block
            reset contiguous block size counter to zero
        }
```

```
    }
}

if worst contiguous block size variable is still equal to 0
    not enough space available to accommodate allocation request, return null

if worst contiguous block size variable is greater than 1 + the requested allocation size
    set best contiguous block size variable to 1 + requested allocation size

while in worst contiguous block
    mark all blocks as allocated by setting each bit in the bit array

assign first block of requested memory with the size of memory that was allocated for the request

return a pointer holding the address to the first available block of memory in the newly allocated memory
block (excludes first block holding the quantity of memory allocated)
```

4. *Deallocation Algorithms*

```
void best_fit_dealloc(void *ptr)
```

```
void worst_fit_dealloc(void *ptr)
```

The logic for both algorithms is identical; the only difference is that each function uses the struct corresponding to the algorithm that it was meant for. These algorithms can be described using the following pseudocode:

```
if pointer is null, return null

retrieve size of memory allocated from first block of the allocated memory space to determine end address for loop

test first memory block; if value is not set (equal to zero), memory is not allocated, and function can return

while in allocated memory block
    reset each bit (set value to zero)
```

5. *External Fragmentation Algorithms*

The logic for both algorithms is identical; the only difference is that each function uses the struct corresponding to the algorithm that it was meant for. These algorithms can be described using the following pseudocode:

```
if size is less than the minimum allocation size (4 bytes = 1 block) return -1

init all flags and counters to zero

while in whole memory space {
    if bit is not set (equal to zero) AND contiguous flag is not set
        set contiguous flag and store current index as beginning of contiguous block

    if bit is not set (equal to zero) AND contiguous flag is set
        increment contiguous block size counter

    if (bit is set (equal to one) AND contiguous flag is set) OR reached end of the memory space {
        reset the contiguous flag
        if the contiguous block size counter is less than the specified fragment size, increment the fragment counter
        reset contiguous block size counter to zero
    }
}

return fragment counter
```

Testing Scenario Description

In the main_test.c file, a single custom test run was created, with two near-identical instances produced to test each initialization, allocation, deallocation, and fragmentation function. The only difference in each of the instances is that they use the appropriate functions relevant to the instance itself. In the test run, various calls for memory initialization, allocation, deallocation, and fragmentation are tested, with the fragmentation count being a key indicator of each algorithm's performance. Additionally, we chose to monitor the time required for each algorithm to make it through the test run, as another indicator of performance.

Experimental Data to Support Comparison Results

Best Fit Test Run

```
INIT BEST FIT TEST RUN.

TEST CASE 1: INIT MEMORY LESS THAN MINIMUM ALLOWED.
~~Best fit - attempting to initialize 3 bytes.~~
```

~~Returning -1.~~

~~Best fit - attempting to initialize 1024 bytes.~~
~~Initializing 1024 bytes of memory.~~
~~# of blocks: 256, # of bits required to represent the # of blocks: 8~~
111111110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
~~External frag: 0~~

TEST CASE 2.1: INIT ALLOC MEMORY LESS THAN MINIMUM ALLOWED.
~~Best fit - attempting to allocate 2 bytes.~~
~~Returning null.~~

TEST CASE 2.2: INIT ALLOC MEMORY GREATER THAN ENTIRE MEMORY SPACE.
~~Best fit - attempting to allocate 1021 bytes.~~
~~Returning null.~~

~~Best fit - attempting to allocate 763 bytes.~~
~~Best fit - allocated 1 + 191 blocks.~~
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111110000
0000000000000000000000000000000000000000000000000000
~~Best fit: p=0xef803c~~

~~External frag: 0~~

~~Best fit - attempting to allocate 8 bytes.~~
~~Best fit - allocated 1 + 2 blocks.~~
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111110
0000000000000000000000000000000000000000000000000000
~~Best fit: p=0xef833c~~

~~External frag: 0~~

~~Best fit - deallocated 192 blocks.~~
111111110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001110
0000000000000000000000000000000000000000000000000000
~~External frag: 0~~

~~Best fit - attempting to allocate 544 bytes.~~
~~Best fit - allocated 1 + 136 blocks.~~
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111100000000000000000000000000000000000000000000000000000001110
0000000000000000000000000000000000000000000000000000
~~Best fit: p=0xef803c~~

~~External frag: 0~~

TEST CASE 3: ALLOC MEMORY GREATER THAN AVAILABLE MEMORY SPACE.~~
~~Best fit - attempting to allocate 810 bytes.~~
~~Returning null.~~

~~Best fit - attempting to allocate 24 bytes.~~
~~Best fit - allocated 1 + 6 blocks.~~

11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111100000000000000000000000000000000000000000000000000000000000001111
11111100000000000000000000000000000000000000000000
~~Best fit: p=0xef8348~~

~~External frag: 0~~

~~Best fit - deallocated 3 blocks.~~
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111100000000000000000000000000000000000000000000000000000000000000001
11111100000000000000000000000000000000000000000000
~~External frag: 0~~

~~Best fit - attempting to allocate 200 bytes.~~
~~Best fit - allocated 1 + 50 blocks.~~
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111100000001
11111100000000000000000000000000000000000000000000
~~Best fit: p=0xef8260~~

~~External frag: 1~~

TEST CASE 4: DEALLOC POINTER ALREADY DEALLOCATED, CONSECUTIVELY.~~
~~Best fit - deallocated 137 blocks.~~
11111111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000011111111111111111111111111111111111111111111111111111100000001
11111100000000000000000000000000000000000000000000
DUPLICATE DEALLOC UP NEXT, SHOULD SKIP TO NEXT INSTRUCTION.
~~Memory not currently allocated. Returning.~~

~~External frag: 1~~

~~Best fit - attempting to allocate 56 bytes.~~
~~Best fit - allocated 1 + 14 blocks.~~
11111111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000011111111111111111111111111111111111111111111111111111100000001
11111111111111111111000000000000000000000000000000
~~Best fit: p=0xef8364~~

~~External frag: 1~~

~~External frag: 1~~

~~Best fit - attempting to allocate 482 bytes.~~
~~Best fit - allocated 1 + 121 blocks.~~
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111110000000000000001111111111111111111111111111111111111111111111111111100000001
11111111111111111111000000000000000000000000000000
~~Best fit: p=0xef803c~~

~~External frag: 2~~

~~Best fit - deallocated 15 blocks.~~
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111110000000000000001111111111111111111111111111111111111111111111111111100000001
11111100000000000000000000000000000000000000000000
~~External frag: 2~~

~~Best fit - deallocated 7 blocks.~~
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111000000000000000011111111111111111111111111111111111111111111111111111111100000000
00000000000000000000000000000000000000000000000000
~~External frag: 1~~

~~Best fit - attempting to allocate 37 bytes.~~
~~Best fit - allocated 1 + 10 blocks.~~
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111000011111111111111111111111111111111111111111111111111111111100000000
00000000000000000000000000000000000000000000000000
~~Best fit: p=0xef8224~~
~~External frag: 1~~

~~Best fit - attempting to allocate 179 bytes.~~
~~Best fit - allocated 1 + 45 blocks.~~
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111110000111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111000000000000000
~~Best fit: p=0xef832c~~
~~External frag: 2~~

~~Best fit - deallocated 51 blocks.~~
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111110000000000000000000000000000000000000000000000000000000000011111111
11111111111111111111111111111111111111100000000000000
~~External frag: 1~~

TEST CASE 5: PASS LESS THAN MINIMUM VALUE TO EXTERNAL FRAG FUNCTION.~~
~~Returning -1.~~

~~External frag: -1~~

~~Best fit - deallocated 122 blocks.~~
11111111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000001111111111100000000000000000000000000000000000000000000000000000000000011111111
11111111111111111111111111111111111111100000000000000
~~External frag: 1~~

END BEST FIT TEST RUN.

System execution time: 0.004148 seconds

Worst Fit Test Run

INIT WORST FIT TEST RUN.

TEST CASE 1: INIT MEMORY LESS THAN MINIMUM ALLOWED.
~~Worst fit - attempting to initialize 3 bytes.~~
~~Returning -1.~~

~~Worst fit - attempting to initialize 1024 bytes.~~
~~Initializing 1024 bytes of memory.~~
~~# of Blocks: 256, # of bits required to represent the # of blocks: 8~~
11111111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
~~External frag: 0~~

TEST CASE 2.1: INIT ALLOC MEMORY LESS THAN MINIMUM ALLOWED.
~~Worst fit - attempting to allocate 2 bytes.~~
~~Returning null.~~

TEST CASE 2.2: INIT ALLOC MEMORY GREATER THAN ENTIRE MEMORY SPACE.
~~Worst fit - attempting to allocate 1021 bytes.~~
~~Returning null.~~

~~Worst fit - attempting to allocate 763 bytes.~~
~~Worst fit - allocated 1 + 191 blocks.~~
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111110000
00000000000000000000000000000000000000000000000000
~~Worst fit: p=0x135c03c~~

~~External frag: 0~~

~~Worst fit - attempting to allocate 8 bytes.~~
~~Worst fit - allocated 1 + 2 blocks.~~
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111110
00000000000000000000000000000000000000000000000000
~~Worst fit: p=0x135c33c~~

~~External frag: 0~~

~~Worst fit - deallocated 192 blocks.~~
11111111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001110
00000000000000000000000000000000000000000000000000
~~External frag: 0~~

~~Worst fit - attempting to allocate 544 bytes.~~
~~Worst fit - allocated 1 + 136 blocks.~~
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111000000000000000000000000000000000000000000000000000000000000001110
00000000000000000000000000000000000000000000000000
~~Worst fit: p=0x135c03c~~

~~External frag: 0~~

TEST CASE 3: ALLOC MEMORY GREATER THAN AVAILABLE MEMORY SPACE.~~
~~Worst fit - attempting to allocate 810 bytes.~~
~~Returning null.~~

~~Worst fit - attempting to allocate 24 bytes.~~
~~Worst fit - allocated 1 + 6 blocks.~~
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111000000000000000000000000000000000000000000000000000000001110
00000000000000000000000000000000000000000000000000
~~Worst fit: p=0x135c260~~

~~External frag: 0~~

~~Worst fit - deallocated 3 blocks.~~

11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111100000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
~~External frag: 0~~

~~Worst fit - attempting to allocate 200 bytes.~~
~~Worst fit - allocated 1 + 50 blocks.~~
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111110
00000000000000000000000000000000000000000000000000
~~Worst fit: p=0x135c27c~~

~~External frag: 0~~

TEST CASE 4: DEALLOC POINTER ALREADY DEALLOCATED, CONSECUTIVELY.~~
~~Worst fit - deallocated 137 blocks.~~
11111111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000011111111111111111111111111111111111111111111111111111111111111110
00000000000000000000000000000000000000000000000000
DUPLICATE DEALLOC UP NEXT, SHOULD SKIP TO NEXT INSTRUCTION.
~~Memory not currently allocated. Returning.~~

~~External frag: 0~~

~~Worst fit - attempting to allocate 56 bytes.~~
~~Worst fit - allocated 1 + 14 blocks.~~
11111111111111111111100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000011111111111111111111111111111111111111111111111111111111111111110
00000000000000000000000000000000000000000000000000
~~Worst fit: p=0x135c03c~~

~~External frag: 0~~

~~External frag: 0~~

~~Worst fit - attempting to allocate 482 bytes.~~
~~Worst fit - allocated 1 + 121 blocks.~~
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111110
00000000000000000000000000000000000000000000000000
~~Worst fit: p=0x135c078~~

~~External frag: 0~~

~~Worst fit - deallocated 15 blocks.~~
11111111000000000000000111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111110
00000000000000000000000000000000000000000000000000
~~External frag: 1~~

~~Worst fit - deallocated 7 blocks.~~
11111111000000000000000111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111100000001111111111111111111111111111111111111111111111111111111110
00000000000000000000000000000000000000000000000000
~~External frag: 2~~

~~Worst fit - attempting to allocate 37 bytes.~~
~~Worst fit - allocated 1 + 10 blocks.~~

```
11111111000000000000000111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111000000011111111111111111111111111111111111111111111111111111
11111111110000000000000000000000000000000000000000
~~Worst fit: p=0x135c348~~
~~External frag: 2~~

~~Worst fit - attempting to allocate 179 bytes.~~
~~Returning null.~~

~~Worst fit: p=(nil)~~
~~External frag: 2~~

~~Worst fit - deallocated 51 blocks.~~
11111111000000000000000111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111000000000000000000000000000000000000000000000000000000000001
11111111110000000000000000000000000000000000000000
~~External frag: 1~~

TEST CASE 5: PASS LESS THAN MINIMUM VALUE TO EXTERNAL FRAG FUNCTION.~~
~~Returning -1.~~

~~External frag: -1~~

~~Worst fit - deallocated 122 blocks.~~
11111111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
11111111110000000000000000000000000000000000000000
~~External frag: 0~~

END WORST FIT TEST RUN.

System execution time: 0.004614 seconds
```

Critical Analysis of Experimental Data

Observation 1: Notice in the worst-fit test run, the allocation call for 179 bytes (highlighted in red) actually fails due to a lack of contiguous, free memory large enough to accommodate the request, which does not occur in the best-fit test run.

- This is a disadvantage of worst-fit, as if a request to larger memory arrives at a later stage, then it can't be accommodated as it is likely that the largest contiguous memory block has already been split and occupied.

Observation 2: In both test runs, the fragmentation count peaks at a value of 2, however the best-fit test run completes with a fragmentation count of 1, while the worst-fit test run completes with a fragmentation count of 0. Additionally, the smallest contiguous memory block in best-fit is 4 blocks in size, while the smallest contiguous memory block in worst-fit is 7 blocks in size.

- This is a disadvantage of best-fit, as the memory space may tend to fill up with smaller, less useful contiguous blocks of memory.

Observation 3: The running time for the worst-fit test run is actually longer than the running time of that for the best-fit test run, even though the worst-fit test run failed on the allocation call for 179 bytes, which effectively would have saved it from entering two loops, each with a running

time complexity of *O*(n). Because we found this strange, we decided to execute the program a few more times to compare running time results, and we were able to gather these results:

Test Run	Best-fit (seconds)	Worst-fit (seconds)
1	0.004148	0.004614
2	0.003836	0.003809
3	0.003684	0.006566
4	0.004176	0.003727
5	0.006662	0.004133
6	0.004060	0.004219
7	0.004451	0.003743
8	0.003877	0.000822
9	0.003842	0.003721
10	0.003760	0.003750
Average	0.004250	0.003910

According to the average produced based on the running time of each test run, the worst-fit test run was performed in a shorter period of time. This can be attributed to the fact that the allocation call for 179 bytes failed, as mentioned earlier.

- In general, best-fit is observed to be slower than the worst-fit allocation algorithm.

Conclusion

In the analysis section of this report, we identified key characteristics of each allocation algorithm that relate to their overall performance. The worst-fit allocation algorithm was unable to satisfy a larger memory request at a later stage, while the best-fit allocation algorithm produced smaller, less useful external fragments. Both algorithms have a similar performance, with their flaws being presented in a variety of circumstances, but typically not under the exact same conditions.

From the perspective of satisfying memory allocation requests, it would be fair to conclude that the best-fit allocation algorithm is the better performing algorithm. However, from the perspective of external fragmentation, the worst-fit algorithm is the better performing algorithm, as the fragments produced over the course of the test run were still of greater length, and potentially of greater use than that produced by the best-fit algorithm.