Performance Evaluation of Memory Allocation Algorithms

Joshua Dominguez

Executive Summary

The report evaluates the creation of a custom implementation of malloc and free using four different memory allocation algorithms tested versus the system malloc. The algorithms implemented in the program are Next Fit, First Fit, Best Fit, and Worst Fit to which each chooses a free node from the heap to allocate memory differently. A test program was implemented to gather results to evaluate the efficiency of heap management and execution time. Additionally, statistics were collected to show the effects on the heap based on which algorithm is used.

The findings in the report are based on the 20 trials utilizing the test program created, which causes enough variation between the four algorithms to have comprehensive results. As expected, the system malloc outperformed the four implementations of memory allocation algorithms significantly in terms of execution time. However, among the four algorithms, the best performance was using the Next Fit algorithm as it had the lowest average execution time. Alongside the good performance by Next Fit, Worst Fit had the worst performance in both execution time and heap management due to iterating through every node in the heap every single time before selection, as well as the largest number of blocks at the end of the program as a result of choosing the largest node and splitting after. Best Fit performed well in the management of memory, although it fell behind Next Fit by having roughly double the average run time.

Algorithms Implemented

There are four different algorithms implemented in the program to manage memory allocation. The algorithms differ in the way in which a free block in the heap is chosen for the requested size of data to be used. For this program, the algorithms implemented are First Fit, Next Fit, Best Fit, and Worst Fit.

As its name states, the First Fit algorithm will select the first free block of memory that is large enough to hold the size of the data requested. The algorithm starts at the beginning of the heap every time and iterates through the blocks until a node is free and can hold the data. The algorithm will end pointing to a node that can be used or will be NULL meaning that there was not a node found in the heap that satisfies both conditions of being free and having enough space as requested.

While Next Fit algorithm is very similar to First Fit, it is different in that it does not start from the beginning of the heap every time. Next Fit starts from the last position that memory was allocated and searches through the heap list onward. Given that it does not start from the beginning of the list if there was a prior memory allocation, the algorithm also searches the list from the beginning if no node was found initially.

Alternatively, both Worst Fit and Best Fit start at the beginning of the heap and search through every node to determine the block chosen for memory allocation. The major difference between the two algorithms is that Worst Fit will choose the free node with the largest size, while Best Fit selects the free node with the smallest difference between the size of the node and the requested size of memory to allocate.

Test Implementation

In order to benchmark the four implementations of memory allocation algorithms, a program was written called mytest.c (see Appendix A). The test program was written to capture the execution time of each implementation, in both tics and seconds, with enough variation to create differences between the algorithms. Additionally, the test allows for the evaluation of captured statistics such as number of splits, heap growth, heap fragmentation, and max heap size.

Moreover, the test program calls malloc() and free thousands of times, and in many different places and sizes to cause differences in both the statistics and execution time between the algorithms. In total malloc() was called 50,797 times, and free() was called 26,266 times with 71,322,360 bytes of memory requested. As calling every malloc and free in order with no variation would not cause any discrepancies in the way the algorithms chose a block to allocate memory, the program frees memory in various places after the initial mallocs and then mallocs many more times.

Furthermore, the timing for the execution time is captured by including the header file clock.h to utilize clock(). The variable start and end call clock() before and after all of the calls to malloc and free, with the difference between the two resulting in the number of tics allotted during execution time. In specifics, the program is a good test case for the algorithms because of how the mallocs and frees create heap fragmentation that is handled differently. Firstly, a pointer array of 25,000 elements is initialized with different allocations of memory. Multiples of 3 call malloc with a size of 5000 bytes, multiples of 50 with 500 bytes, and multiples of 79 are freed. Then a for loop is used to free the memory allocated previously, followed by a new pointer array with 25,000 elements malloced with 500 bytes each. Then every position of the new array that is a multiple of 17 is freed. Lastly, a third pointer array of size 1000 is created to malloc 355 bytes, causing each algorithm to have different run times due to their style of choosing a block for this memory to be allocated. Overall, the program creates enough difference in run time and statistics between the algorithms to collect comprehensive results.

Test Results

Time (Seconds)					
Trial	FF	NF	BF	WF	System Malloc
1	11.572978	9.368211	18.9299	21.491647	0.025559
2	11.666787	9.402681	19.308627	21.336609	0.026212
3	11.630069	9.400607	18.913403	21.364911	0.024532
4	11.605252	9.386126	18.929737	21.463461	0.024813
5	12.567057	9.37727	18.814497	21.452869	0.026155
6	11.687373	9.605062	19.021505	21.690214	0.031592
7	11.705003	9.386126	18.905647	21.805009	0.028794
8	11.756796	9.491174	18.977348	21.6049	0.025365
9	11.619382	9.430606	19.087499	21.449448	0.025317
10	11.575426	9.466607	18.943746	21.381341	0.03175
11	11.676348	9.448637	18.682499	21.311449	0.027712
12	11.666375	9.370198	19.033032	21.481083	0.02611
13	11.623429	9.338052	18.889799	21.487223	0.025145

14	11.639093	9.4592	18.995683	21.38648	0.025164
15	11.615297	9.523982	18.990069	21.594764	0.024798
16	11.639093	9.475071	19.043278	21.47247	0.024537
17	11.630059	9.495401	19.003043	21.598357	0.025453
18	12.578219	9.59353	18.902885	21.49799	0.025936
19	11.716464	9.71808	19.043019	21.510467	0.024865
20	11.730269	9.623358	19.139748	21.485714	0.025007
Average	11.74503845	9.46799895	18.9777482	21.4933203	0.0262408

Table 1: Execution Time of Algorithms and System Malloc

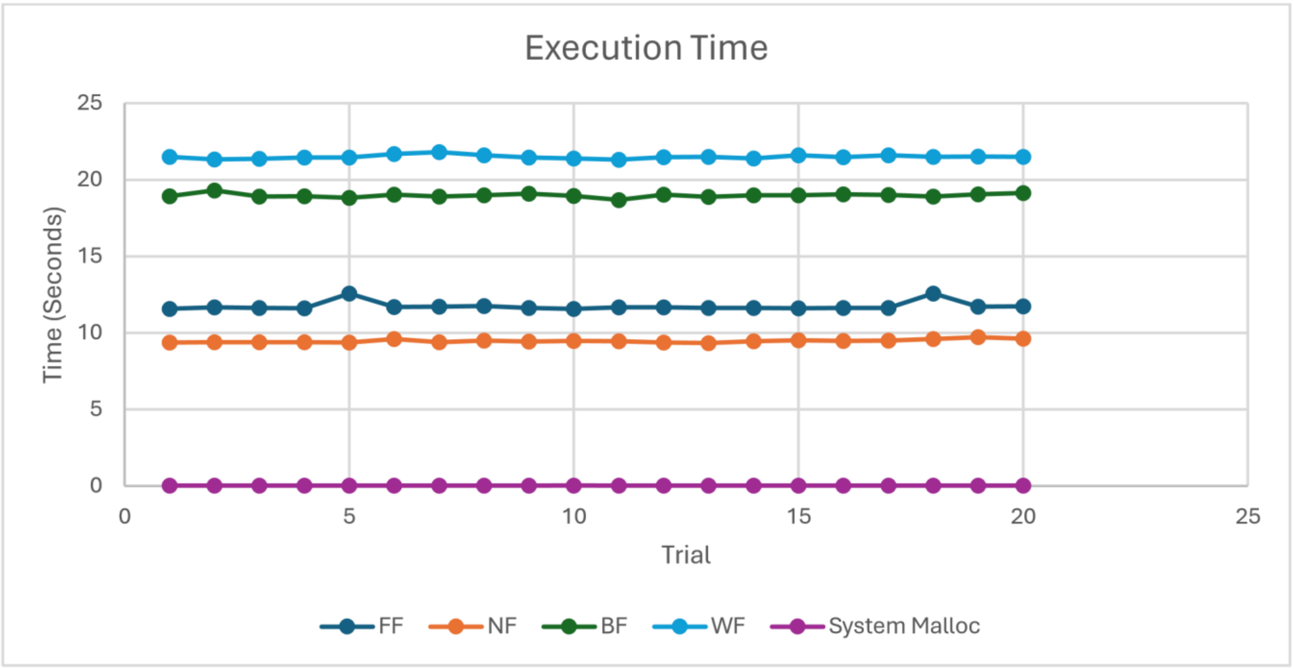


Figure 1: Chart of Execution Times

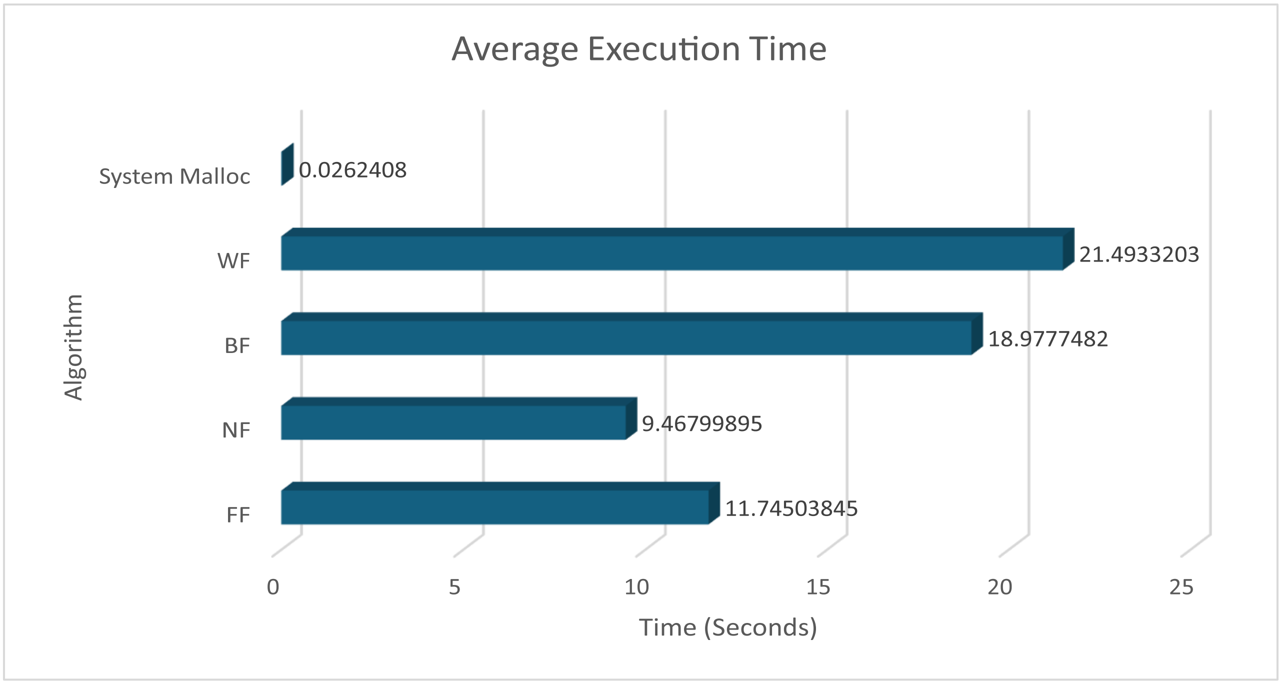
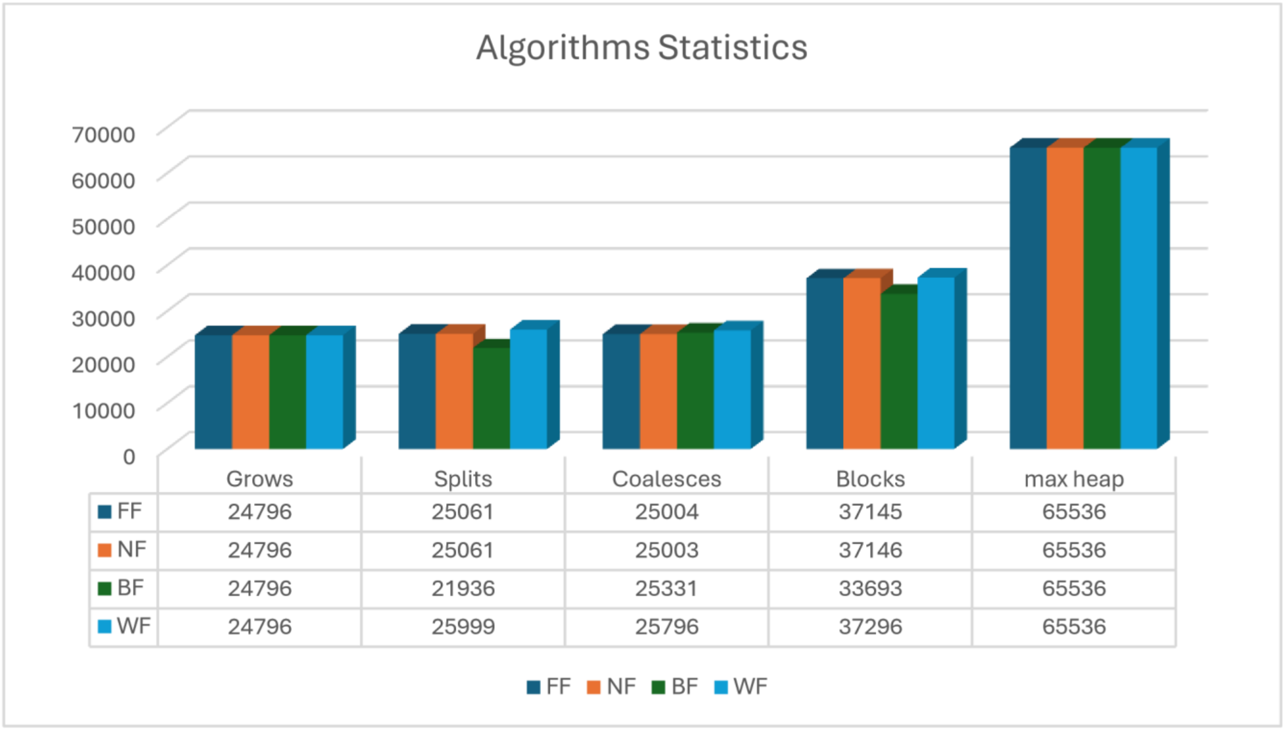


Figure 2: Average Execution Time

Stats									
	Mallocs	Frees	Reuses	**Grows**	**Splits**	**Coalesces**	**Blocks**	Requested	**Max heap**
FF	50797	26266	26001	24796	25061	25004	37145	71322360	65536
NF	50797	26266	26001	24796	25061	25003	37146	71322360	65536
BF	50797	26266	26001	24796	21936	25331	33693	71322360	65536
WF	50797	26266	26001	24796	25999	25796	37296	71322360	65536

Table 2: Algorithm Statistics



	Grows	Splits	Coalesces	Blocks	max heap
FF	24796	25061	25004	37145	65536
NF	24796	25061	25003	37146	65536
BF	24796	21936	25331	33693	65536
WF	24796	25999	25796	37296	65536

Figure 3: Algorithm Statics

<u>Evaluation of Results</u>

Each memory allocation algorithm, and system malloc, was run with the test program 20 times capturing the run time as shown in Table 1. The system malloc outperformed the implementation of the four algorithms significantly. However, the differences between the other four implementations are very telling of how the algorithms handled the memory allocation. As both best fit and worst fit search from the beginning and through all of the blocks in the heap list before selection, their run times are higher than that of Next Fit and First Fit. While Next Fit and Next Fit are similar in implementation just as Worst Fit and Best Fit are similar in implementation, the results collected from the test program can be analyzed to see which algorithm handled the memory allocation more efficiently.

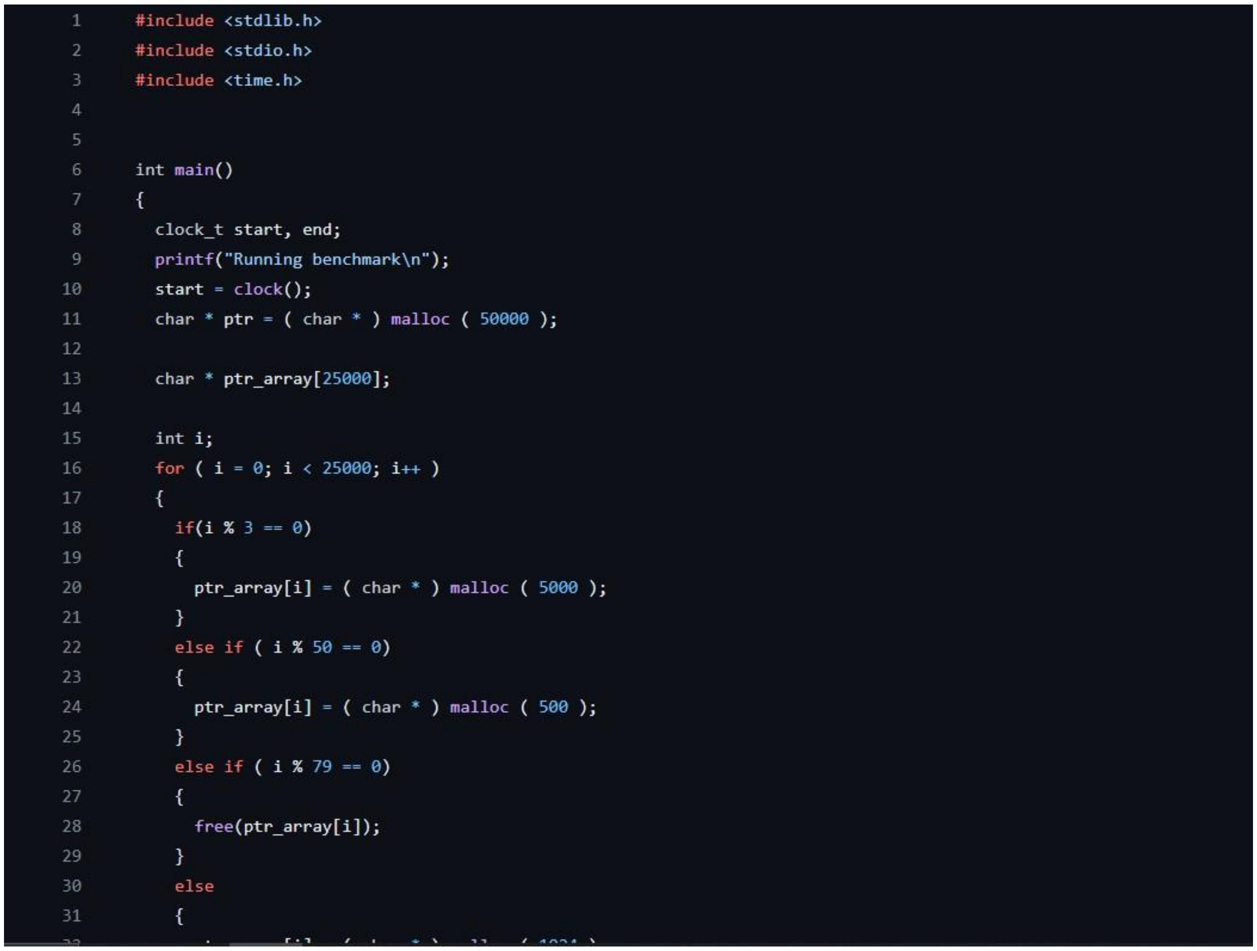
On average, out of the four algorithms Next Fit had the lowest run time of 9.46799895. This result makes sense as blocks were being freed and allocated in various places, and the Next Fit algorithm starts searching for the node to use based on the last allocation meaning that it will have to traverse the heap list less than that of every other algorithm. Alternatively First Fit has to begin at the beginning of the heap list every single time, which resulted in a higher average run time than Next Fit. Best Fit was the third highest of the four algorithms in terms of run time, given that it loops over the entire heap list like Worst Fit, resulting in running 13% quicker than Worst Fit. (Table 2)

The statistics output by every program can be used to evaluate how efficiently the algorithms handled memory allocation, as shown in Figure 3. One of the takeaways from the data is that Best Fit split the least amount of times due to how the algorithm works, which also resulted in the heap growth being the least out of all algorithms. On the other hand, the most splits and blocks at the end of the program was with Worst Fit due to finding the largest node to choose even if there was a massive amount of leftover space. Additionally, one other important statistic is the number of coalesces being different by one between Next Fit and First Fit, which also changes the amount of blocks at the end of the program by one. This was caused due to the fact that Next Fit began searching at the last allocation whereas First Fit started at the beginning of the list, finding a previous block that was free alongside the current block causing it to coalesce.

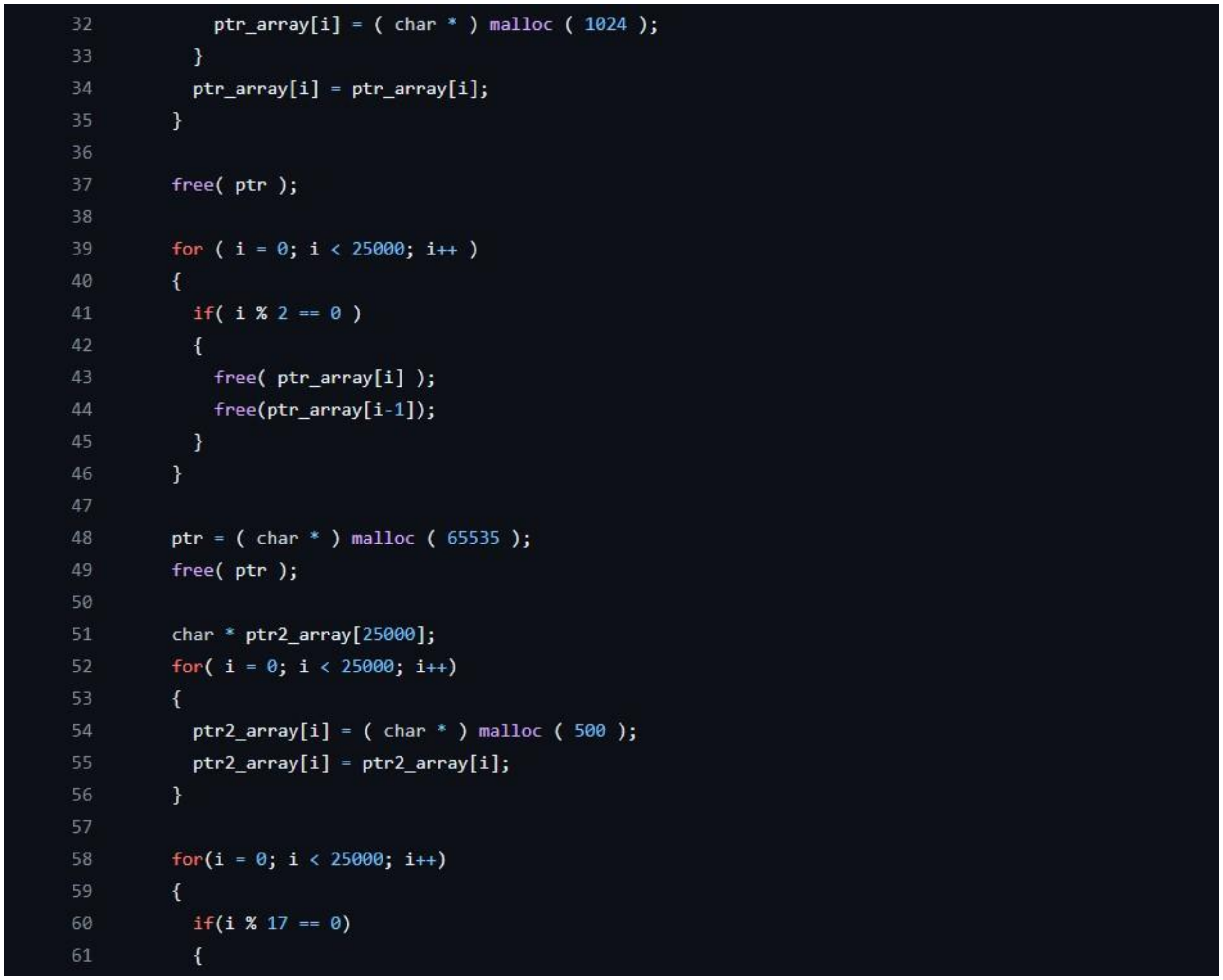
Conclusion

In conclusion, Next Fit performed the best in terms of performance out of the four memory allocation algorithms implemented due to run time. However, the actual management of and heap fragmentation was better performed by Best Fit because of the way it chooses which block to use for the requested data size. As the heap split less than any of the others, there was better allocation of the blocks available. System Malloc outperformed all of the algorithms by a very significant amount, on average being almost 818% faster than the worst run time of Worst Fit. The Next Fit algorithm performed the best for the test program implemented based on run time, and Best fit in terms of managing heap fragmentation.

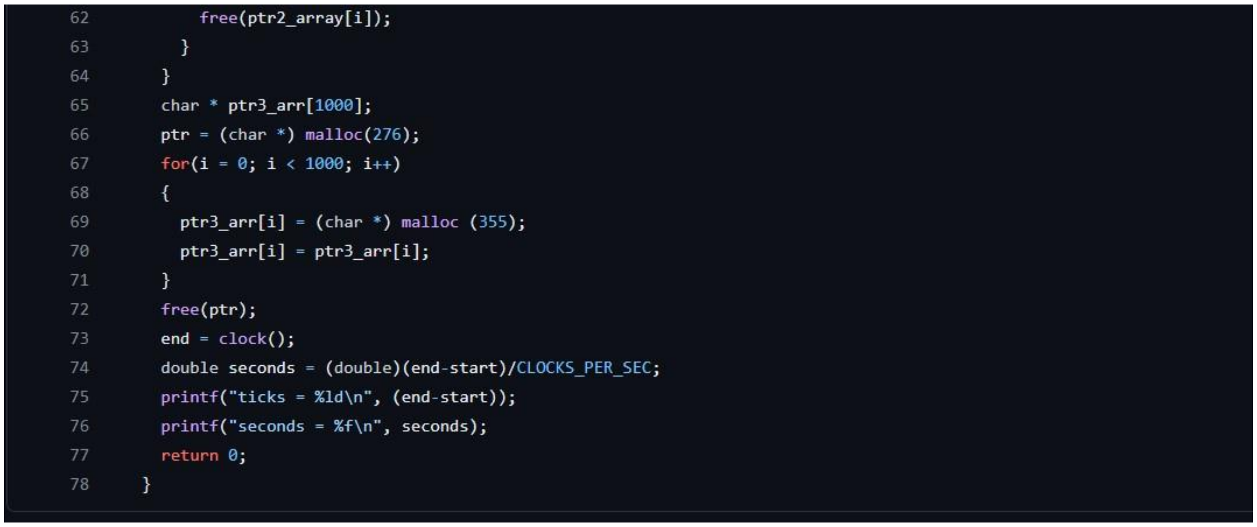
Appendix A: mytest.c



```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>


int main()
{
  clock_t start, end;
  printf("Running benchmark\n");
  start = clock();
  char * ptr = ( char * ) malloc ( 50000 );

  char * ptr_array[25000];

  int i;
  for ( i = 0; i < 25000; i++ )
  {
    if(i % 3 == 0)
    {
      ptr_array[i] = ( char * ) malloc ( 5000 );
    }
    else if ( i % 50 == 0)
    {
      ptr_array[i] = ( char * ) malloc ( 500 );
    }
    else if ( i % 79 == 0)
    {
      free(ptr_array[i]);
    }
    else
    {
```



```c
      ptr_array[i] = ( char * ) malloc ( 1024 );
    }
    ptr_array[i] = ptr_array[i];
  }

  free( ptr );

  for ( i = 0; i < 25000; i++ )
  {
    if( i % 2 == 0 )
    {
      free( ptr_array[i] );
      free(ptr_array[i-1]);
    }
  }

  ptr = ( char * ) malloc ( 65535 );
  free( ptr );

  char * ptr2_array[25000];
  for( i = 0; i < 25000; i++)
  {
    ptr2_array[i] = ( char * ) malloc ( 500 );
    ptr2_array[i] = ptr2_array[i];
  }

  for(i = 0; i < 25000; i++)
  {
    if(i % 17 == 0)
    {
```



```c
      free(ptr2_array[i]);
    }
  }
  char * ptr3_arr[1000];
  ptr = (char *) malloc(276);
  for(i = 0; i < 1000; i++)
  {
    ptr3_arr[i] = (char *) malloc (355);
    ptr3_arr[i] = ptr3_arr[i];
  }
  free(ptr);
  end = clock();
  double seconds = (double)(end-start)/CLOCKS_PER_SEC;
  printf("ticks = %ld\n", (end-start));
  printf("seconds = %f\n", seconds);
  return 0;
}
```