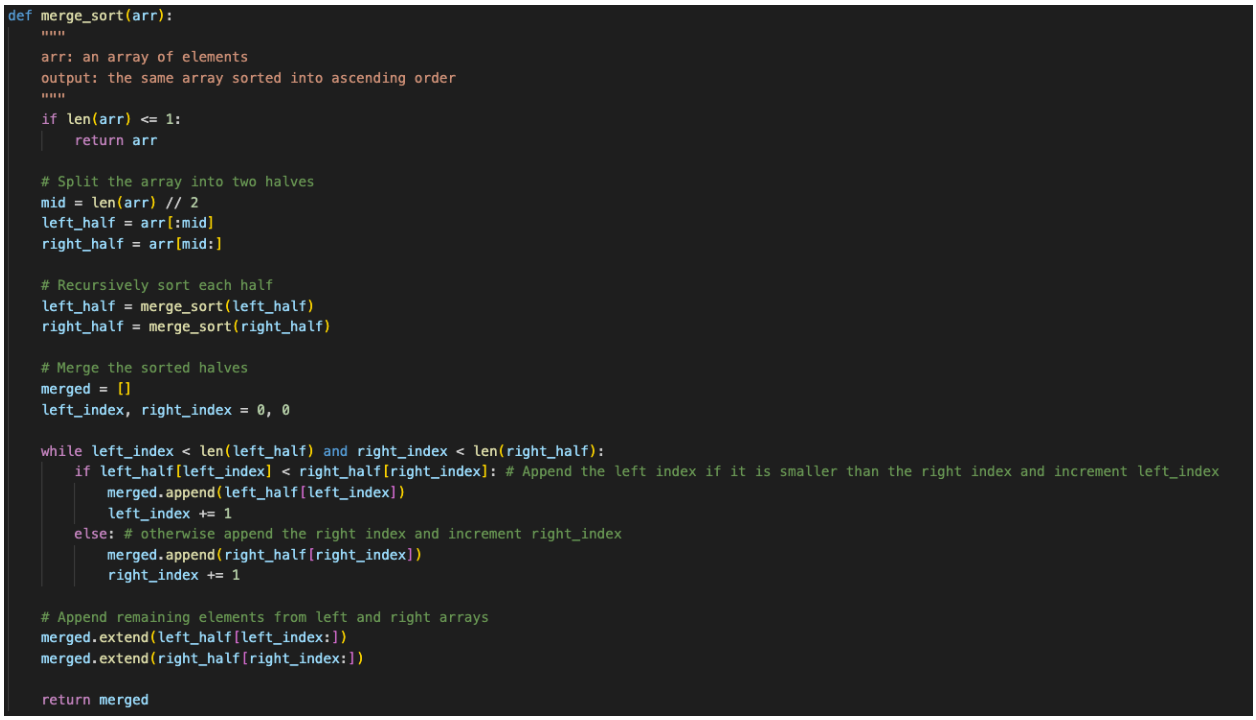
Report on Merge Sort Algorithm

Code

Below is a screenshot of the code put together for a merge sort algorithm. We take in an array of elements and bisect the array into halves recursively until the array we are left with is 1 item or less, then we merge them together.



```python
def merge_sort(arr):
    """
    arr: an array of elements
    output: the same array sorted into ascending order
    """
    if len(arr) <= 1:
        return arr

    # Split the array into two halves
    mid = len(arr) // 2
    left_half = arr[:mid]
    right_half = arr[mid:]

    # Recursively sort each half
    left_half = merge_sort(left_half)
    right_half = merge_sort(right_half)

    # Merge the sorted halves
    merged = []
    left_index, right_index = 0, 0

    while left_index < len(left_half) and right_index < len(right_half):
        if left_half[left_index] < right_half[right_index]: # Append the left index if it is smaller than the right index and increment left_index
            merged.append(left_half[left_index])
            left_index += 1
        else: # otherwise append the right index and increment right_index
            merged.append(right_half[right_index])
            right_index += 1

    # Append remaining elements from left and right arrays
    merged.extend(left_half[left_index:])
    merged.extend(right_half[right_index:])

    return merged
```

Asymptotic Analysis

Looking at the code, we can determine that there are two recursive calls each that are half the size of the input of length n. After the recursion, there is a merge of elements that will take roughly n time. Therefore, we can assume the recurrence relation is: **2T(n/2) + n**. The first **2** is the number of recursive calls, **T(n/2)** is the representation of an individual recursive call of half the size of n, and **+ n** is the time complexity of the remainder of the function.

Overall, after using the master method, we can determine that the above merge_sort algorithm is:

O(nlogn)

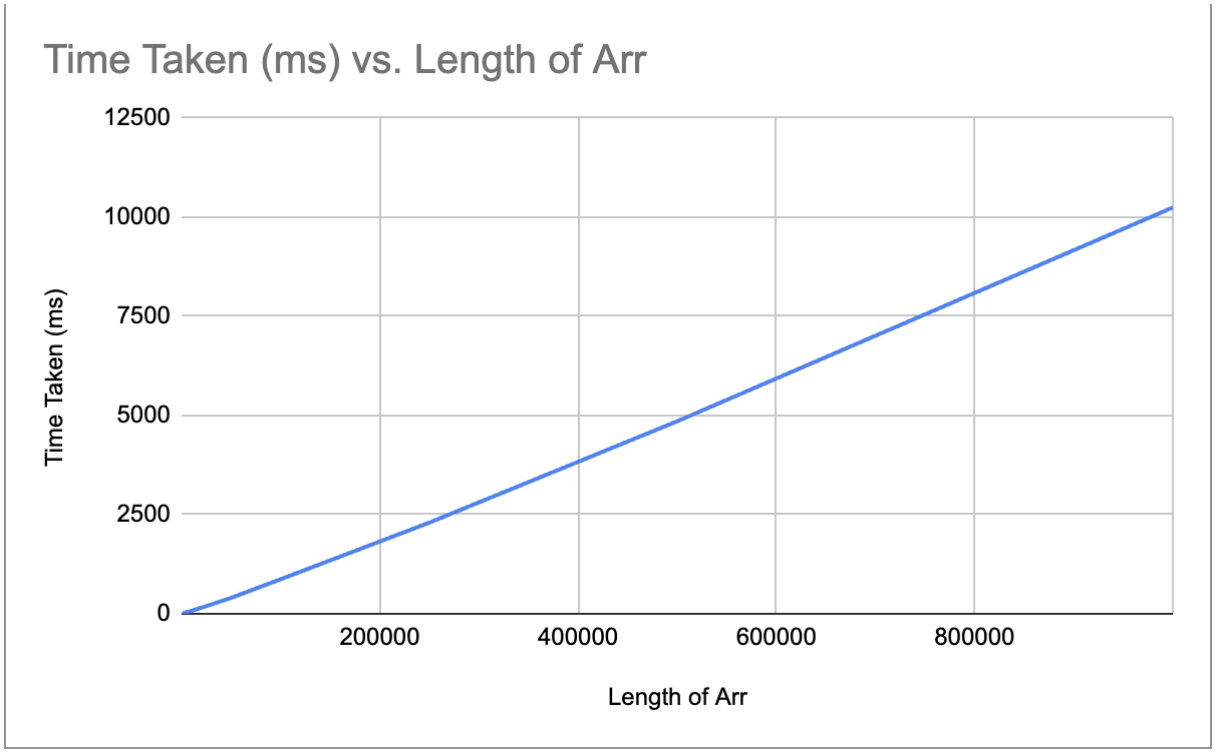
Time Taken

Below is a table of the length of the array and the average time taken over 5 tests of an array of that size. The range is from 1k elements to 1m elements. Additionally, there is a graph below charting out the data in the table.

Time in a table:

Length of Arr	Time Taken (ms)
1000	6.2
2000	14
5000	35.8
10000	72.4
25000	191.2
50000	399
100000	869.2
250000	2296.2
500000	4851.6
1000000	10242.6

Time on a graph from 1k to 1m elements:



cProfiler Examples

Below are examples of the cProfiler results at 1k through 1m entries. I am only selecting one cProfiler example per case so as to not bloat the document, though as stated in the Time section on the previous page, I had run each input size 5 times and taken the average.

1k:

```
L54GKMFP2P:project Jacques.Plante@disney.com$ python3 SortingAlgorithm.py
         37990 function calls (35992 primitive calls) in 0.007 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.006    0.006 <string>:1(<module>)
   1999/1    0.004    0.000    0.006    0.006 SortingAlgorithm.py:3(merge_sort)
        1    0.000    0.000    0.006    0.006 SortingAlgorithm.py:38(main1k)
      999    0.000    0.000    0.000    0.000 random.py:242(_randbelow_with_getrandbits)
        1    0.000    0.000    0.001    0.001 random.py:350(shuffle)
        1    0.000    0.000    0.006    0.006 {built-in method builtins.exec}
    21838    0.001    0.000    0.001    0.000 {built-in method builtins.len}
     8678    0.001    0.000    0.001    0.000 {method 'append' of 'list' objects}
      999    0.000    0.000    0.000    0.000 {method 'bit_length' of 'int' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
     1998    0.000    0.000    0.000    0.000 {method 'extend' of 'list' objects}
     1474    0.000    0.000    0.000    0.000 {method 'getrandbits' of '_random.Random' objects}
```

2k:

```
L54GKMFP2P:project Jacques.Plante@disney.com$ python3 SortingAlgorithm.py
         81953 function calls (77955 primitive calls) in 0.013 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.013    0.013 <string>:1(<module>)
   3999/1    0.008    0.000    0.012    0.012 SortingAlgorithm.py:3(merge_sort)
        1    0.000    0.000    0.013    0.013 SortingAlgorithm.py:44(main2k)
     1999    0.001    0.000    0.001    0.000 random.py:242(_randbelow_with_getrandbits)
        1    0.000    0.000    0.001    0.001 random.py:350(shuffle)
        1    0.000    0.000    0.013    0.013 {built-in method builtins.exec}
    47770    0.003    0.000    0.003    0.000 {built-in method builtins.len}
    19378    0.002    0.000    0.002    0.000 {method 'append' of 'list' objects}
     1999    0.000    0.000    0.000    0.000 {method 'bit_length' of 'int' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
     3998    0.000    0.000    0.000    0.000 {method 'extend' of 'list' objects}
     2805    0.000    0.000    0.000    0.000 {method 'getrandbits' of '_random.Random' objects}
```

5k:

```
L54GKMFP2P:project Jacques.Plante@disney.com$ python3 SortingAlgorithm.py
         225488 function calls (215490 primitive calls) in 0.036 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.036    0.036 <string>:1(<module>)
   9999/1    0.020    0.000    0.033    0.033 SortingAlgorithm.py:3(merge_sort)
        1    0.000    0.000    0.036    0.036 SortingAlgorithm.py:50(main5k)
     4999    0.001    0.000    0.002    0.000 random.py:242(_randbelow_with_getrandbits)
        1    0.001    0.001    0.003    0.003 random.py:350(shuffle)
        1    0.000    0.000    0.036    0.036 {built-in method builtins.exec}
   132803    0.007    0.000    0.007    0.000 {built-in method builtins.len}
    55244    0.004    0.000    0.004    0.000 {method 'append' of 'list' objects}
     4999    0.000    0.000    0.000    0.000 {method 'bit_length' of 'int' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
     9998    0.001    0.000    0.001    0.000 {method 'extend' of 'list' objects}
     7441    0.000    0.000    0.000    0.000 {method 'getrandbits' of '_random.Random' objects}
```

10k:

```
L54GKMFP2P:project Jacques.Plante@disney.com$ python3 SortingAlgorithm.py
         480641 function calls (460643 primitive calls) in 0.081 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.081    0.081 <string>:1(<module>)
  19999/1    0.047    0.000    0.075    0.075 SortingAlgorithm.py:3(merge_sort)
        1    0.000    0.000    0.081    0.081 SortingAlgorithm.py:56(main10k)
     9999    0.002    0.000    0.004    0.000 random.py:242(_randbelow_with_getrandbits)
        1    0.002    0.002    0.006    0.006 random.py:350(shuffle)
        1    0.000    0.000    0.081    0.081 {built-in method builtins.exec}
   285459    0.016    0.000    0.016    0.000 {built-in method builtins.len}
   120528    0.010    0.000    0.010    0.000 {method 'append' of 'list' objects}
     9999    0.001    0.000    0.001    0.000 {method 'bit_length' of 'int' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
    19998    0.002    0.000    0.002    0.000 {method 'extend' of 'list' objects}
    14654    0.001    0.000    0.001    0.000 {method 'getrandbits' of '_random.Random' objects}
```

25k:

```
L54GKMFP2P:project Jacques.Plante@disney.com$ python3 SortingAlgorithm.py
         1299906 function calls (1249908 primitive calls) in 0.201 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.201    0.201 <string>:1(<module>)
  49999/1    0.116    0.000    0.187    0.187 SortingAlgorithm.py:3(merge_sort)
        1    0.000    0.000    0.201    0.201 SortingAlgorithm.py:62(main25k)
    24999    0.006    0.000    0.010    0.000 random.py:242(_randbelow_with_getrandbits)
        1    0.004    0.004    0.014    0.014 random.py:350(shuffle)
        1    0.000    0.000    0.201    0.201 {built-in method builtins.exec}
   779402    0.040    0.000    0.040    0.000 {built-in method builtins.len}
   334087    0.026    0.000    0.026    0.000 {method 'append' of 'list' objects}
    24999    0.002    0.000    0.002    0.000 {method 'bit_length' of 'int' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
    49998    0.005    0.000    0.005    0.000 {method 'extend' of 'list' objects}
    36417    0.002    0.000    0.002    0.000 {method 'getrandbits' of '_random.Random' objects}
```

50k:

```
L54GKMFP2P:project Jacques.Plante@disney.com$ python3 SortingAlgorithm.py
         2749855 function calls (2649857 primitive calls) in 0.435 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.001    0.001    0.435    0.435 <string>:1(<module>)
  99999/1    0.251    0.000    0.406    0.406 SortingAlgorithm.py:3(merge_sort)
        1    0.001    0.001    0.435    0.435 SortingAlgorithm.py:68(main50k)
    49999    0.012    0.000    0.019    0.000 random.py:242(_randbelow_with_getrandbits)
        1    0.008    0.008    0.028    0.028 random.py:350(shuffle)
        1    0.000    0.000    0.435    0.435 {built-in method builtins.exec}
  1658631    0.089    0.000    0.089    0.000 {built-in method builtins.len}
   718148    0.057    0.000    0.057    0.000 {method 'append' of 'list' objects}
    49999    0.004    0.000    0.004    0.000 {method 'bit_length' of 'int' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
    99998    0.009    0.000    0.009    0.000 {method 'extend' of 'list' objects}
    73076    0.004    0.000    0.004    0.000 {method 'getrandbits' of '_random.Random' objects}
```

100k:

```
L54GKMFP2P:project Jacques.Plante@disney.com$ python3 SortingAlgorithm.py
         5800773 function calls (5600775 primitive calls) in 1.136 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.003    0.003    1.135    1.135 <string>:1(<module>)
  199999/1    0.651    0.000    1.037    1.037 SortingAlgorithm.py:3(merge_sort)
        1    0.003    0.003    1.132    1.132 SortingAlgorithm.py:74(main100k)
    99999    0.032    0.000    0.053    0.000 random.py:242(_randbelow_with_getrandbits)
        1    0.038    0.038    0.092    0.092 random.py:350(shuffle)
        1    0.000    0.000    1.135    1.135 {built-in method builtins.exec}
  3517993    0.220    0.000    0.220    0.000 {built-in method builtins.len}
  1536604    0.145    0.000    0.145    0.000 {method 'append' of 'list' objects}
    99999    0.011    0.000    0.011    0.000 {method 'bit_length' of 'int' objects}
        1    0.001    0.001    0.001    0.001 {method 'disable' of '_lsprof.Profiler' objects}
   199998    0.023    0.000    0.023    0.000 {method 'extend' of 'list' objects}
   146176    0.011    0.000    0.011    0.000 {method 'getrandbits' of '_random.Random' objects}
```

250k:

```
L54GKMFP2P:project Jacques.Plante@disney.com$ python3 SortingAlgorithm.py
         15478302 function calls (14978304 primitive calls) in 2.474 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.003    0.003    2.474    2.474 <string>:1(<module>)
  499999/1    1.440    0.000    2.313    2.313 SortingAlgorithm.py:3(merge_sort)
        1    0.004    0.004    2.471    2.471 SortingAlgorithm.py:80(main250k)
   249999    0.058    0.000    0.093    0.000 random.py:242(_randbelow_with_getrandbits)
        1    0.061    0.061    0.154    0.154 random.py:350(shuffle)
        1    0.000    0.000    2.474    2.474 {built-in method builtins.exec}
  9458818    0.502    0.000    0.502    0.000 {built-in method builtins.len}
  4168429    0.325    0.000    0.325    0.000 {method 'append' of 'list' objects}
   249999    0.017    0.000    0.017    0.000 {method 'bit_length' of 'int' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
   499998    0.046    0.000    0.046    0.000 {method 'extend' of 'list' objects}
   351055    0.018    0.000    0.018    0.000 {method 'getrandbits' of '_random.Random' objects}
```

500k:

```
L54GKMFP2P:project Jacques.Plante@disney.com$ python3 SortingAlgorithm.py
         32457534 function calls (31457536 primitive calls) in 5.206 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.009    0.009    5.206    5.206 <string>:1(<module>)
 999999/1    3.024    0.000    4.852    4.852 SortingAlgorithm.py:3(merge_sort)
        1    0.009    0.009    5.197    5.197 SortingAlgorithm.py:86(main500k)
   499999    0.116    0.000    0.187    0.000 random.py:242(_randbelow_with_getrandbits)
        1    0.149    0.149    0.336    0.336 random.py:350(shuffle)
        1    0.000    0.000    5.206    5.206 {built-in method builtins.exec}
 19917912    1.054    0.000    1.054    0.000 {built-in method builtins.len}
  8837280    0.683    0.000    0.683    0.000 {method 'append' of 'list' objects}
   499999    0.035    0.000    0.035    0.000 {method 'bit_length' of 'int' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
   999998    0.091    0.000    0.091    0.000 {method 'extend' of 'list' objects}
   702342    0.036    0.000    0.036    0.000 {method 'getrandbits' of '_random.Random' objects}
```

1m:

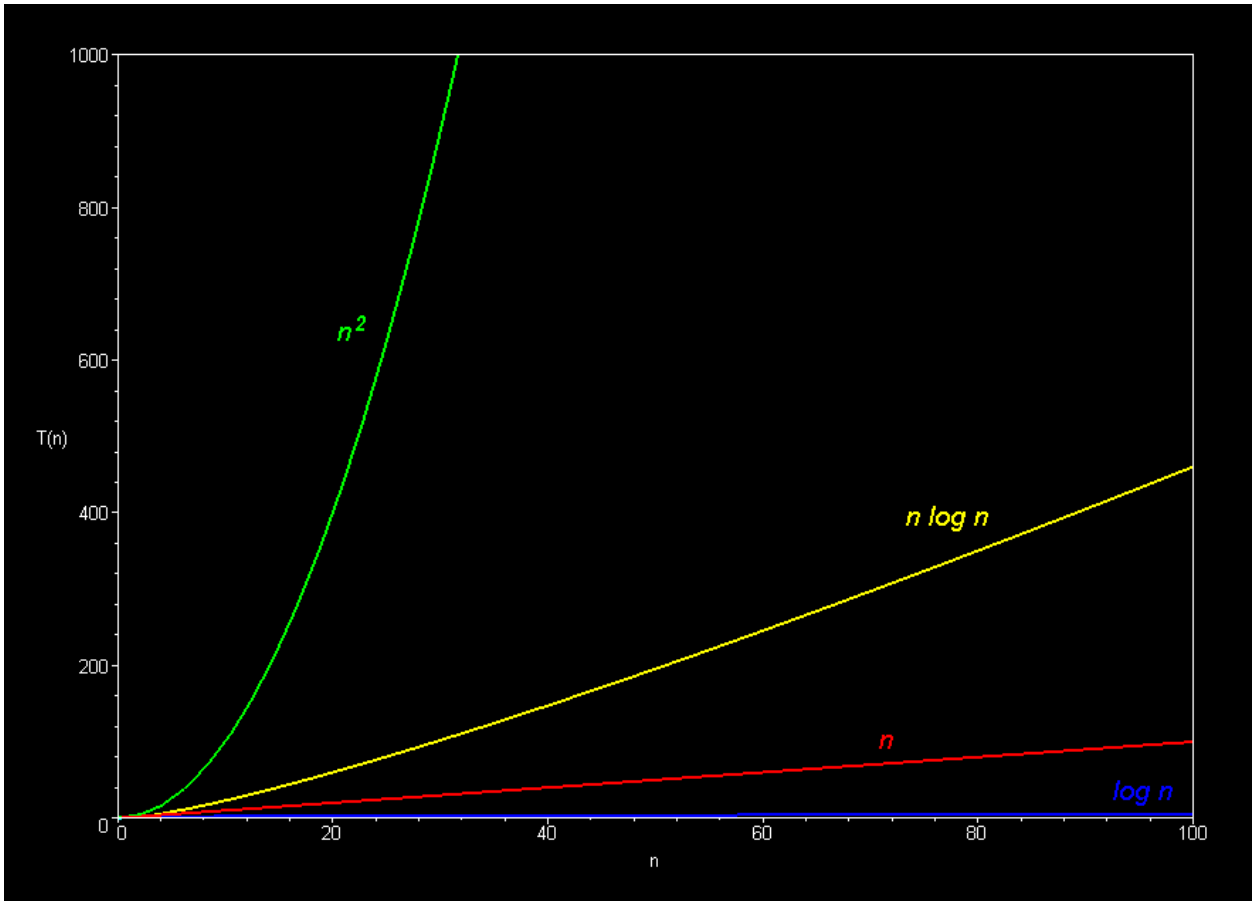
```
L54GKMFP2P:project Jacques.Plante@disney.com$ python3 SortingAlgorithm.py
         67911623 function calls (65911625 primitive calls) in 11.086 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.022    0.022   11.085   11.085 <string>:1(<module>)
1999999/1    6.464    0.000   10.338   10.338 SortingAlgorithm.py:3(merge_sort)
        1    0.021    0.021   11.064   11.064 SortingAlgorithm.py:92(main1m)
   999999    0.229    0.000    0.370    0.000 random.py:242(_randbelow_with_getrandbits)
        1    0.335    0.335    0.705    0.705 random.py:350(shuffle)
        1    0.000    0.000   11.086   11.086 {built-in method builtins.exec}
 41834183    2.242    0.000    2.242    0.000 {built-in method builtins.len}
 18673688    1.448    0.000    1.448    0.000 {method 'append' of 'list' objects}
   999999    0.070    0.000    0.070    0.000 {method 'bit_length' of 'int' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
  1999998    0.184    0.000    0.184    0.000 {method 'extend' of 'list' objects}
  1403752    0.071    0.000    0.071    0.000 {method 'getrandbits' of '_random.Random' objects}
```

Reflection

At first, I was really confused when I was looking at my graph. I knew that merge sort would be O(nlogn) but looking at the graph from 1,000 to 10,000 looked linear. After expanding to 1,000,000 elements, it still looked linear! I did a little digging online to see if others had been noticing similar patterns, and the answer was yes, that it is hard to perceive the logarithmic curve of the merge sort algorithm with the naked eye even up to a million elements. For me to be able to more clearly see the curve, I would need to input some massively larger numbers. Looking at the below graphic, I better understand and compare my graph to the *slight* curvature of the nlogn curve.



https://science.slc.edu/jmarshall/courses/2002/spring/cs50/BigO/index.html