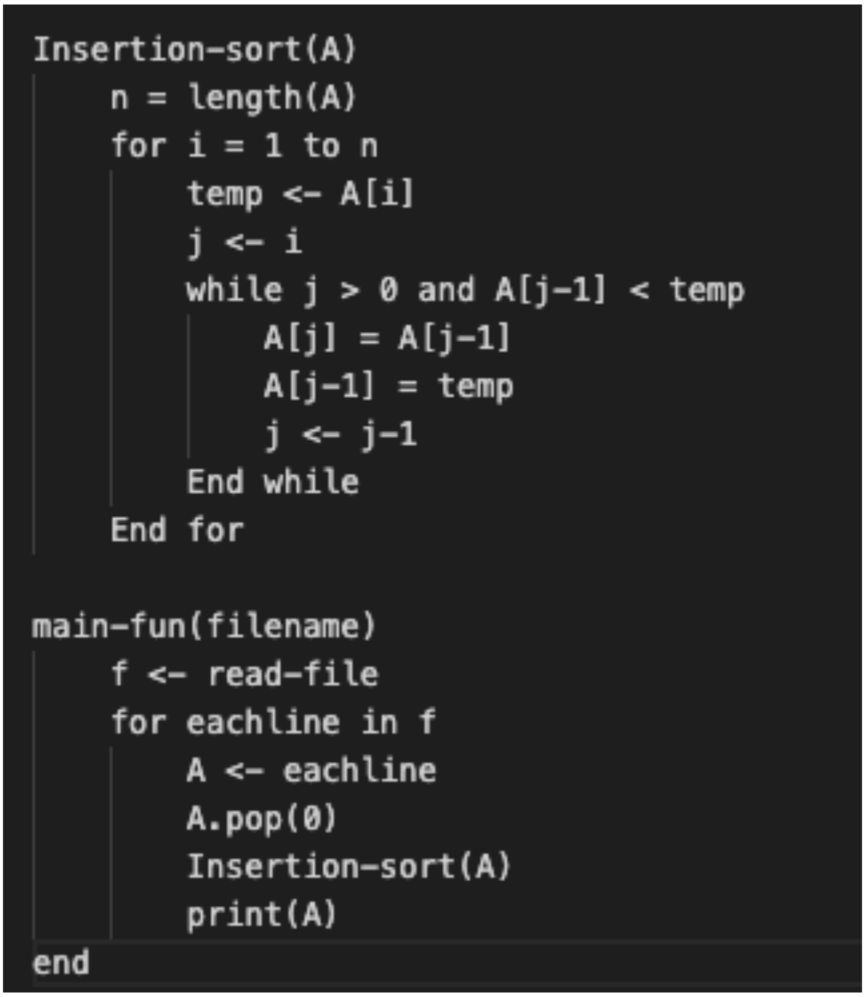
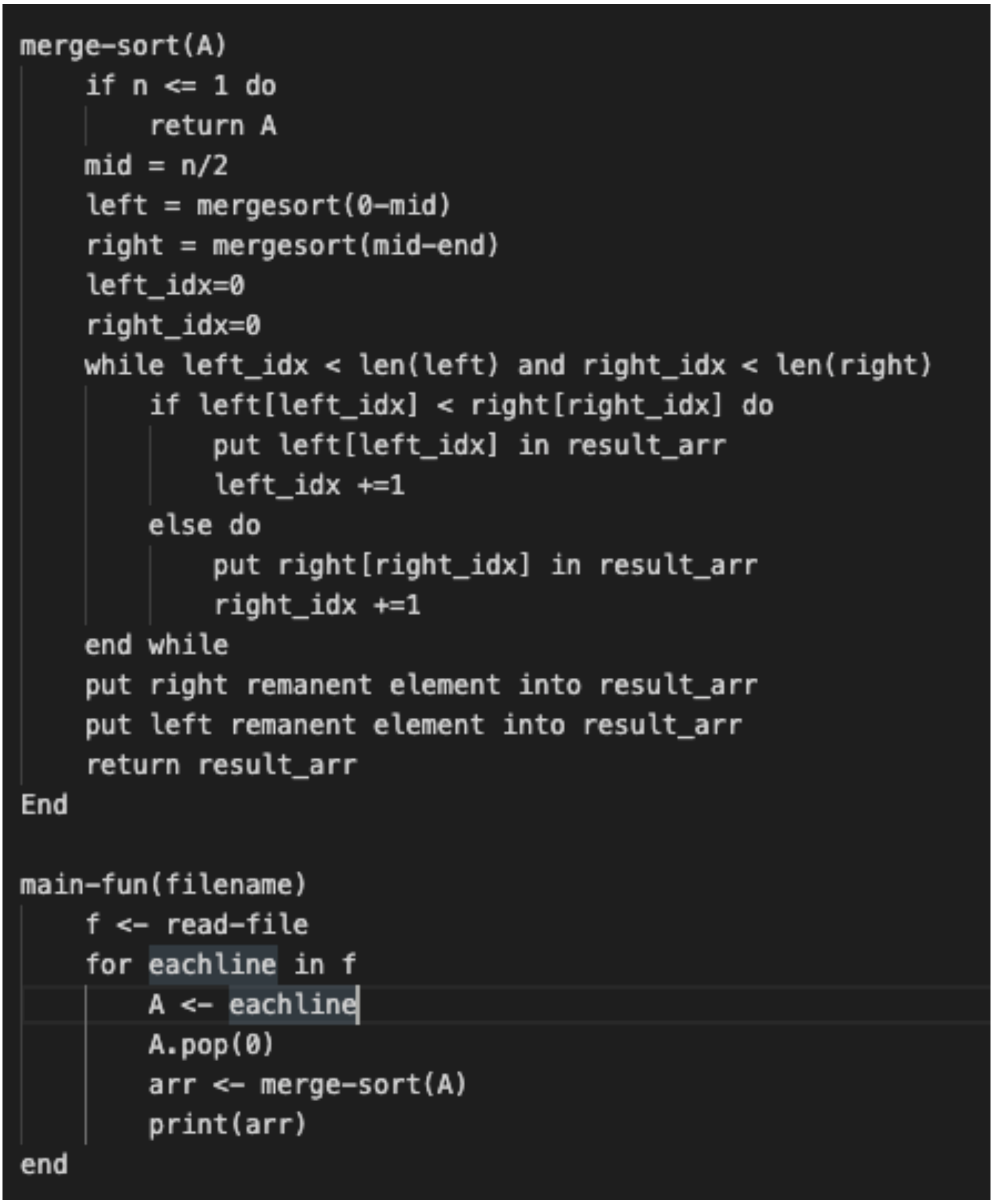
Pseudocode:

Insertion sort:



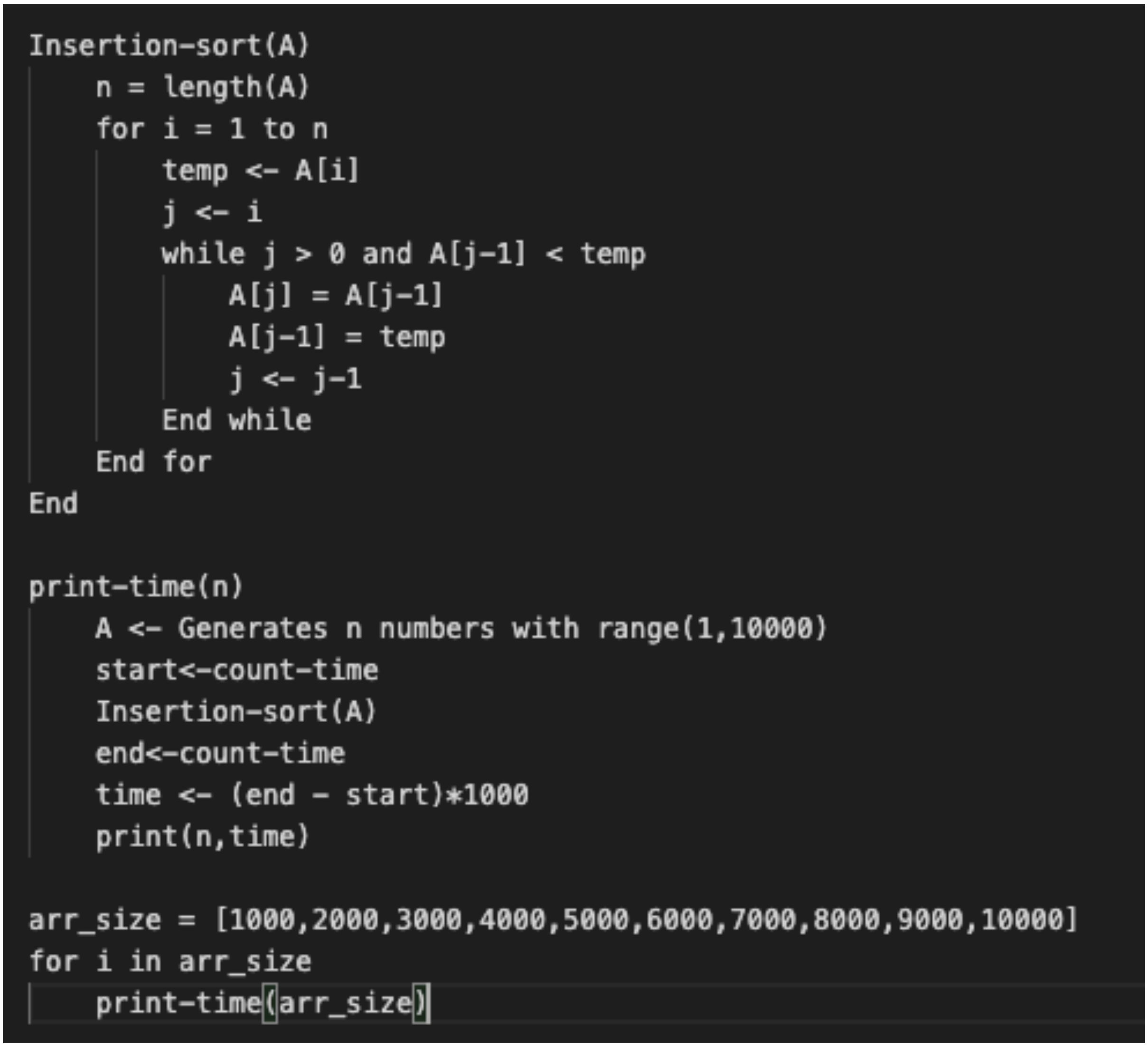
```
Insertion-sort(A)
    n = length(A)
    for i = 1 to n
        temp <- A[i]
        j <- i
        while j > 0 and A[j-1] < temp
            A[j] = A[j-1]
            A[j-1] = temp
            j <- j-1
        End while
    End for

main-fun(filename)
    f <- read-file
    for eachline in f
        A <- eachline
        A.pop(0)
        Insertion-sort(A)
        print(A)
end
```

Merge sort:



```
merge-sort(A)
    if n <= 1 do
        return A
    mid = n/2
    left = mergesort(0-mid)
    right = mergesort(mid-end)
    left_idx=0
    right_idx=0
    while left_idx < len(left) and right_idx < len(right)
        if left[left_idx] < right[right_idx] do
            put left[left_idx] in result_arr
            left_idx +=1
        else do
            put right[right_idx] in result_arr
            right_idx +=1
    end while
    put right remanent element into result_arr
    put left remanent element into result_arr
    return result_arr
End

main-fun(filename)
    f <- read-file
    for eachline in f
        A <- eachline
        A.pop(0)
        arr <- merge-sort(A)
        print(arr)
end
```

Insert Time:

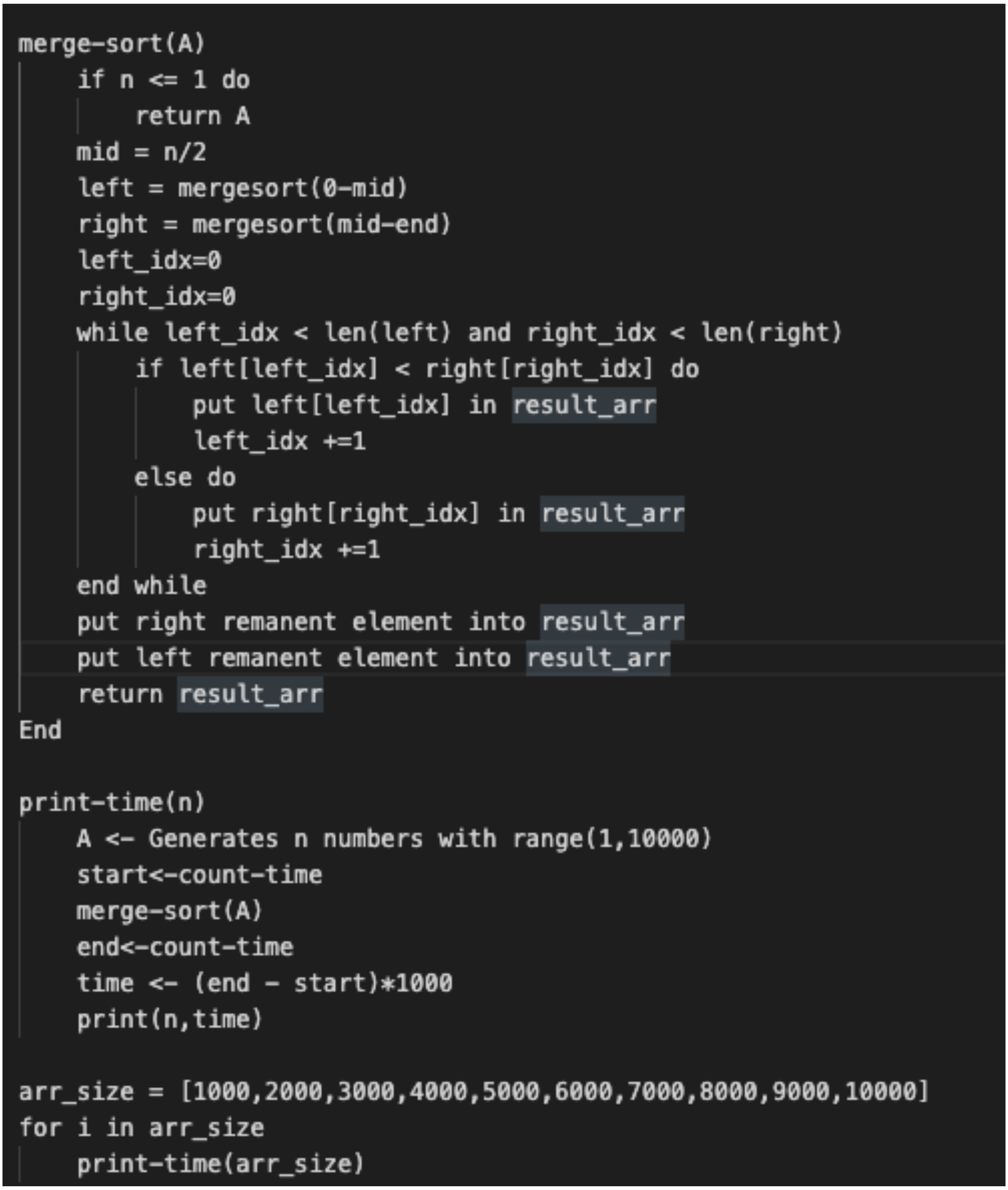


```
Insertion-sort(A)
    n = length(A)
    for i = 1 to n
        temp <- A[i]
        j <- i
        while j > 0 and A[j-1] < temp
            A[j] = A[j-1]
            A[j-1] = temp
            j <- j-1
        End while
    End for
End

print-time(n)
    A <- Generates n numbers with range(1,10000)
    start<-count-time
    Insertion-sort(A)
    end<-count-time
    time <- (end - start)*1000
    print(n,time)

arr_size = [1000,2000,3000,4000,5000,6000,7000,8000,9000,10000]
for i in arr_size
    print-time(arr_size)
```

In InsertTime.py, I have import time and random library. I used time.perf_counter() to record the time.

And I have used random.choices(range(0,10000), k=n) to generated n numbers with range is from 0 to 10000. For the number n, I just put these number in to a list. Using the for loop to put these number in to the function.

Merge Time:



```
merge-sort(A)
    if n <= 1 do
        return A
    mid = n/2
    left = mergesort(0-mid)
    right = mergesort(mid-end)
    left_idx=0
    right_idx=0
    while left_idx < len(left) and right_idx < len(right)
        if left[left_idx] < right[right_idx] do
            put left[left_idx] in result_arr
            left_idx +=1
        else do
            put right[right_idx] in result_arr
            right_idx +=1
    end while
    put right remanent element into result_arr
    put left remanent element into result_arr
    return result_arr
End

print-time(n)
    A <- Generates n numbers with range(1,10000)
    start<-count-time
    merge-sort(A)
    end<-count-time
    time <- (end - start)*1000
    print(n,time)

arr_size = [1000,2000,3000,4000,5000,6000,7000,8000,9000,10000]
for i in arr_size
    print-time(arr_size)
```

In the MergeTime.py, I use the same method with insertTime.py. I have made several different functions, so I can just copy that function which just modification some variable in different file. In order to compare the time of these two algorithms, I chose the same array size.

PART 3

A. Collect Running Times:

Insert Sort:

Best case (All already sorted):

In python use arr.sort() function to sort these array at first. Then put these arrays into the Insert sort function.

Array Size	Running Times (MS)
1000	0.23
2000	0.46
3000	0.68
4000	0.91
5000	1.16
6000	1.39
7000	1.64
8000	1.84
9000	2.17
10000	2.33

Average case (Random list):

Array Size	Running Times (MS)
1000	99.65
2000	426.62
3000	986.93
4000	1717.43
5000	2669.44
6000	3912.20
7000	5321.60
8000	6883.60
9000	8835.50
10000	10914.07

Worst case (All unsorted):

In python use arr.sort() function to sort these array at first. And use arr.reverse() function to reverse this array. Then we get a descending list. Next, put these arrays into the Insert sort function.

Array Size	Running Times (MS)
1000	207.59
2000	871.53
3000	1985.07
4000	3553.99
5000	5533.38
6000	8114.60
7000	11082.14
8000	14647.87
9000	18760.47
10000	23432.62

Merge Sort:

Best case (All already sorted):

In python use arr.sort() function to sort these array at first. Then put these arrays into the merge sort function.

Array Size	Running Times (MS)
1000	4.47
2000	9.53
3000	14.21
4000	19.79
5000	25.00
6000	30.27
7000	35.25
8000	41.40
9000	46.75
10000	53.49

Worse case (Random list):

In the merge sort the worst-case mean is the array is totally unordered.

Array Size	Running Times (MS)
1000	6.79
2000	14.52
3000	23.10
4000	31.82

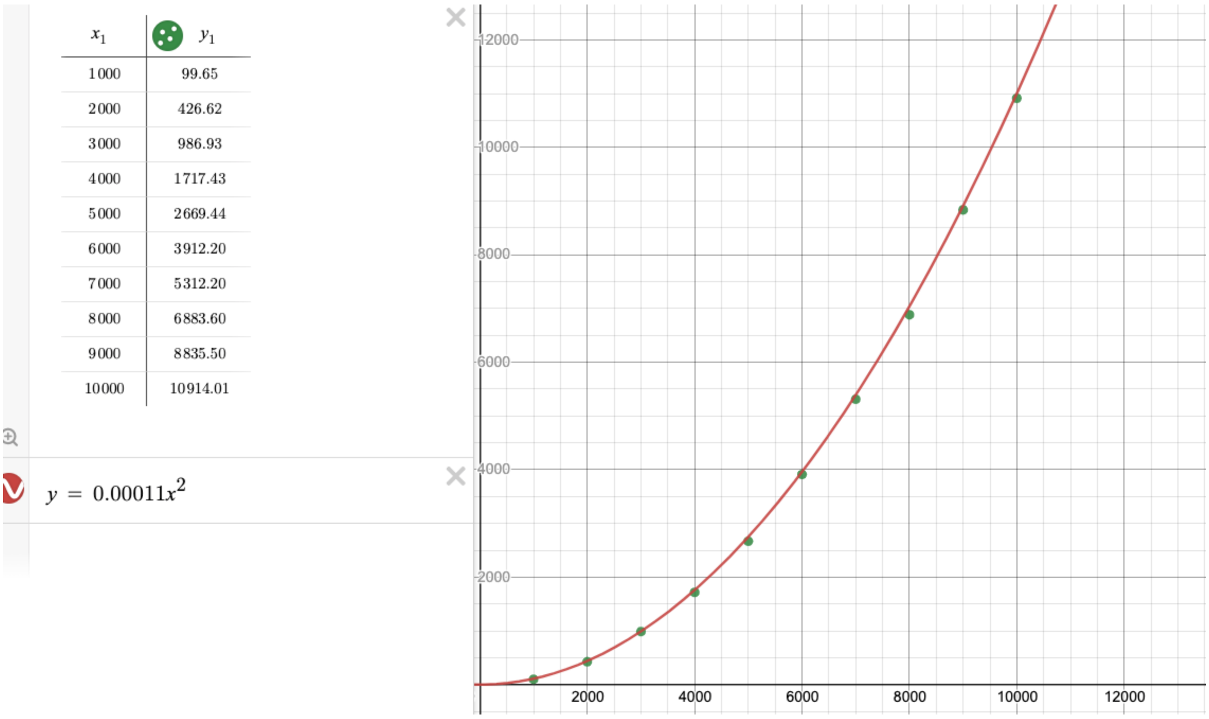
5000	41.09
6000	50.13
7000	59.31
8000	68.54
9000	77.34
10000	87.71

Average case:

Array Size	Running Times (MS)
1000	5.63
2000	24.05
3000	18.66
4000	25.81
5000	33.05
6000	40.2
7000	47.28
8000	54.97
9000	62.05
10000	70.2

B. Plot data and fit a curve:

Insertion sort (Average case):



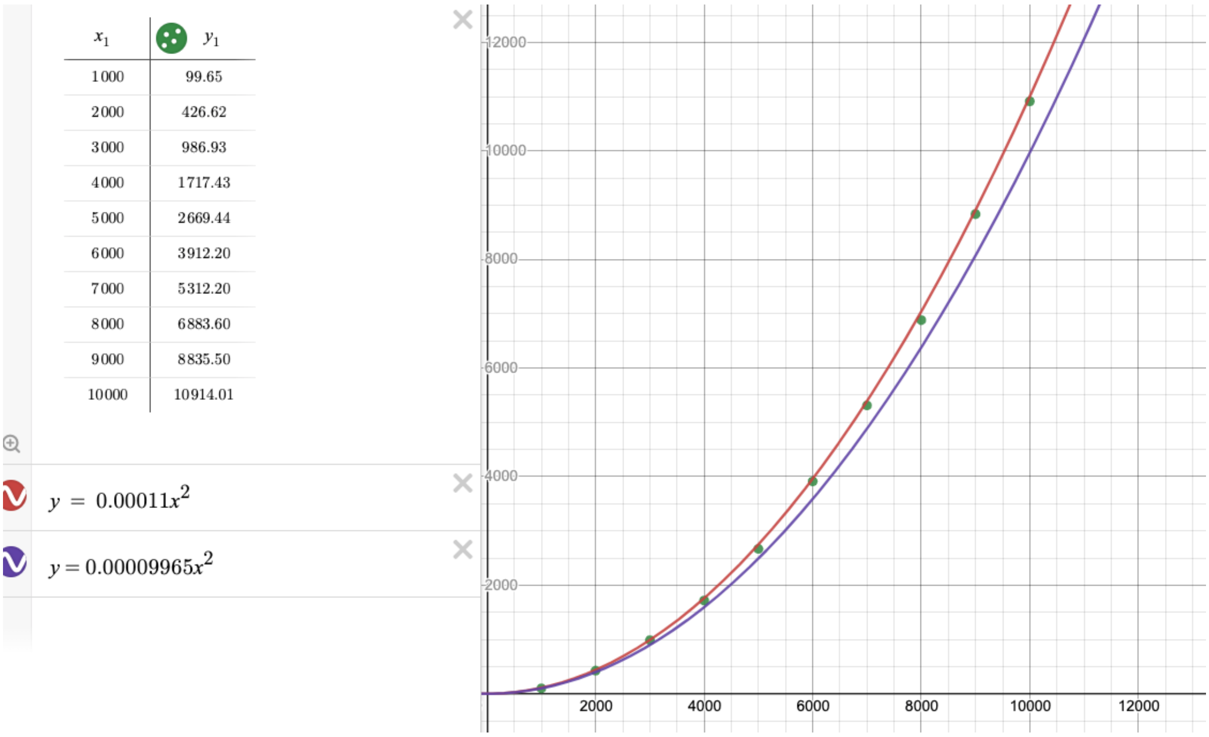
x_1	y_1
1000	99.65
2000	426.62
3000	986.93
4000	1717.43
5000	2669.44
6000	3912.20
7000	5312.20
8000	6883.60
9000	8835.50
10000	10914.01

$y = 0.00011x^2$

Parabola is the best fits this insertion sort plot. Equation: y=0.00011x^2

set the theoretical function is y = ax^2. Make the first point as the theoretical point. So, I get y = 0.00009965x^2

Theoretical (purple line)-Experimental (red line):



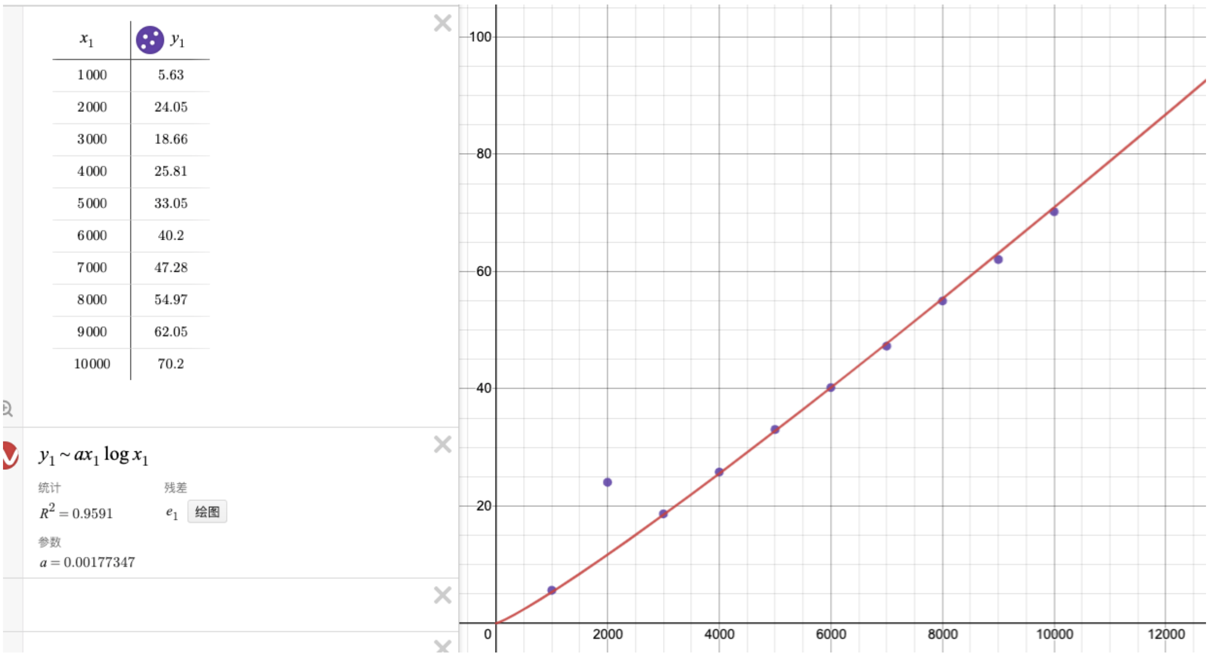
x_1	y_1
1000	99.65
2000	426.62
3000	986.93
4000	1717.43
5000	2669.44
6000	3912.20
7000	5312.20
8000	6883.60
9000	8835.50
10000	10914.01

$y = 0.00011x^2$

$y = 0.00009965x^2$

Therefore, when I use the first experimental point as my theoretical point find the theoretical function. We can see the theoretical time does not have a big different with experimental. Theoretical time < Experimental time

Merge Sort (Average case):



x_1	y_1
1000	5.63
2000	24.05
3000	18.66
4000	25.81
5000	33.05
6000	40.2
7000	47.28
8000	54.97
9000	62.05
10000	70.2

$y_1 \sim ax_1 \log x_1$

统计 $R^2 = 0.9591$

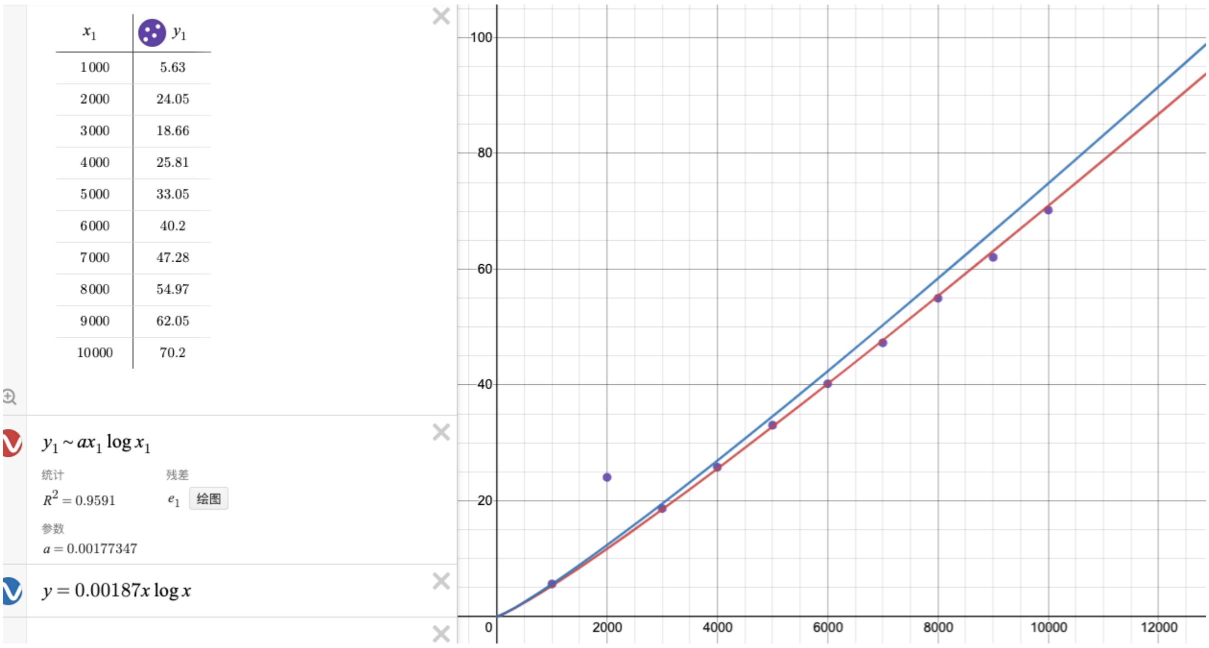
残差 e_1 绘图

参数 $a = 0.00177347$

The logarithm is the best fits this insertion sort plot. Equation: y= 0.00177xlogx

set the theoretical function is y = axlogx. Make the first point as the theoretical point. So, I get y = 0.00187xlogx

Theoretical (blue line)-Experimental (red line):



x_1	y_1
1000	5.63
2000	24.05
3000	18.66
4000	25.81
5000	33.05
6000	40.2
7000	47.28
8000	54.97
9000	62.05
10000	70.2

$y_1 \sim ax_1 \log x_1$

统计 $R^2 = 0.9591$

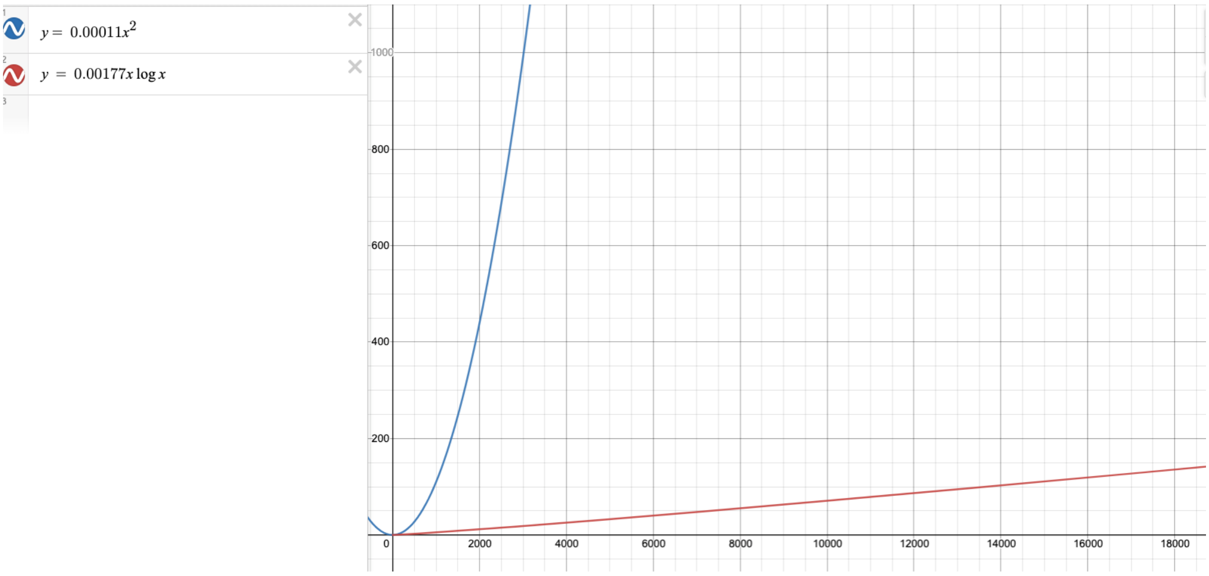
残差 e_1 绘图

参数 $a = 0.00177347$

$y = 0.00187x \log x$

Therefore, when I use the first experimental point as my theoretical point find the theoretical function. We can see the theoretical time does not have a big different with experimental. Theoretical time > Experimental time

C. Combine:



1 $y = 0.00011x^2$

2 $y = 0.00177x \log x$

3

Blue line: Insertion sort

Red line: Merge sort

D. Prediction:

Insertion sort: T(n) = 0.00011n^2

Merge sort: T(n) = 0.00177nlogn

Array of size n = 500000:

Insertion sort: T (500000) = 27500000ms About 7.6 hours

Merge sort: T (500000) = 5044ms About 5 second