Problem 1:

a) I'd recursively divide the original array into 2 (half) sub-arrays until the sub-arrays are small enough (containing 2 elements) so that the min and max values can be determined. Then I'd use a recursive approach to compare min and max values of each sub-array.

Pseudocode:

minMax(array, start, end, min, max)

If $(start\ =\ end)$

Do $min \leftarrow array[start]$

$max \leftarrow array[start]$

Else if $(start\ =\ end\ -\ 1)$

If $(array[start]\ <\ array[end])$

Do $min\ \leftarrow\ array[start]$

$max\ \leftarrow\ array[end]$

Else

do $min\ \leftarrow\ array[end]$

$max\ \leftarrow\ array[start]$

Else

Do $mid\ \leftarrow (start + end)/2$

$minMax(array,\ start,\ mid,\ leftMin,\ leftMax)$

$minMax(array,\ mid + 1,\ end,\ rightMin,\ rightMax)$

If $(leftMin < rightMin)$

Do $min \leftarrow leftMin$

Else

Do $min \leftarrow rightMin$

If $(leftMax < rightMax)$

Do $max \leftarrow rightMax$

Else

Do $max \leftarrow leftMax$

b) T(n) = 2T(n/2) + c

c) Using masters theorem:

$a\ = 2,\ b\ = 2,\ k = 0$

$log\ _2 2\ =\ 1 >\ k$

$\Rightarrow \Theta(n^{log\ _2 2}) = \Theta(n)$

I think the theoretical running time of the recursive min_and_max algorithm is the same as the iterative algorithm since the running time of the iterative algorithm is O(n).

Problem 2:

a)

merge3(array, start, firstThirdStart, secondThirdStart, end)

> *create tempArray*
>
> $a \leftarrow start$
>
> $b \leftarrow firstThirdStart$
>
> $c \leftarrow secondThirdStart$
>
> **While** $(a,\ b,\ and\ c\ smaller\ than\ firstThirdStart,\ secondThirdStart,\ and\ end)$
>
> > **if** $(array\ at\ a\ is\ smallest)$
> >
> > > *append element at a to tempArray*
> > >
> > > a ++
> >
> > **Else if** $(array\ at\ b\ is\ smallest)$
> >
> > > *append element at b to tempArray*
> > >
> > > b ++
> >
> > **Else if** $(array\ at\ c\ is\ smallest)$
> >
> > > *append element at c to tempArray*
> > >
> > > c ++
>
> **While** *first third and last third still have elements*
>
> > *add smallest elements to tempArray*
>
> **While** *first third and second third have elements*
>
> > *add smallest elements to tempArray*
>
> **While** *second third and last third have elements*
>
> > *add smallest elements to tempArray*
>
> **While** *first third has elements*
>
> > *add elements to tempArray*
> >
> > a ++
>
> **While** *second third has elements*
>
> > *add elements to tempArray*
> >
> > b ++
>
> **While** *last third has elements*
>
> > *add elements to tempArray*
> >
> > c ++
>
> $array \leftarrow tempArray$

mergeSort3(array, start, end)

$firstThirdStart \leftarrow (start + end)/3$

$secondThirdStart \leftarrow firstThirdStart + firstThirdStart$

$mergeSort3(array,\ start,\ firstThirdStart)$

$mergeSort3(array,\ firstThirdStart,\ secondThirdStart)$

$mergeSort3(array,\ secondThirdStart,\ end)$

$merge3(array,\ start,\ firstThirdStart,\ secondThirdStart,\ end)$

b) $T(n) = 3T(n/3)\ + O(n)$

c) $a = 3,\ b = 3,\ k = 1,\ p = 0$

$log\ _3 3 = 1 = k$

$\Rightarrow \theta(n^k log^{p+1} n) = \theta(nlogn)$

Problem 4:

b) Collect Running Times

Size n	Running Time (seconds)
100,000	0.04
200,000	0.07
300,000	0.09
400,000	0.08
500,000	0.08
600,000	0.10
700,000	0.11
800,000	0.13
900,000	0.15
1,000,000	0.17

Table 1: Running Times of Merge Sort 3

c) Plot data and fit a curve

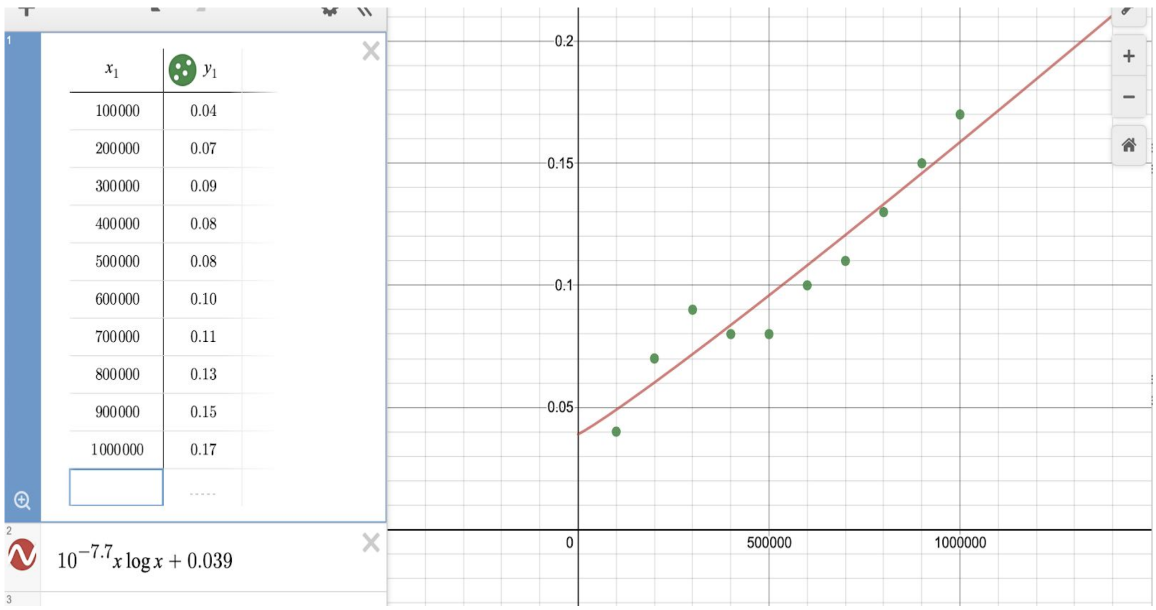

Figure 1: Graph for Running Time of Merge Sort 3

d)

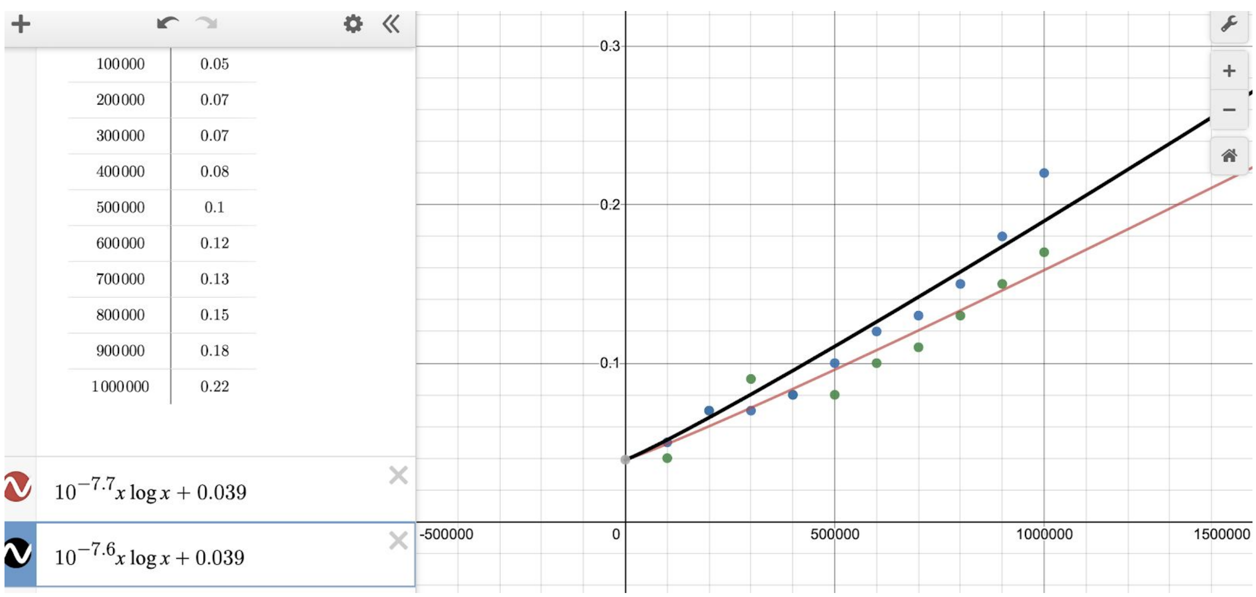

Figure 2: Combined Graph for MergeSort and MergeSort3

Based on the combined graph, MergeSort3 (red curve) seems to run a little bit faster since the red curve lies below the black curve as the input size increases. The MergeSort3 has time complexity of nlogn which matches my theoretical time complexity found in part c of problem 2.