Michael Smith

Problem #1

A) Divide array to 2 parts and compare max & min of both to get max & min of whole thing

```
MaxMin (arr, arr_size)
  if arr_size = 1:
    return elem as both max & min
  elseif arr_size = 2
    compare to determine min & max
    return max & min
  else
    recur for max & min of left half
    recur for righ half
    one finds max comparison
    one finds min comparison
    return max & min
```

B) $T(n) = T(\text{floor}(n/2)) + T(\text{ceil}(n/2)) + 2$

C) $T(2) = 1 \quad T(1) = 0$

$T(n) = 2T(n/2) + 2$

$T(n) = 3n/2 - 2$

* uses n steps to get max/min cause it traverses the array linearly

Problem #2

A)

```
Mergesort3 (A[0..n-1
  if n ≤ 1
    return A[0...n-1]
  Let k = ⌈n/3⌉ & m = ⌈2n/3⌉
    return merge3(mergesort3)

merge3 (L0, L1, L3)
  return merge(L0, merge(L1, L3))
```

B)

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

C)

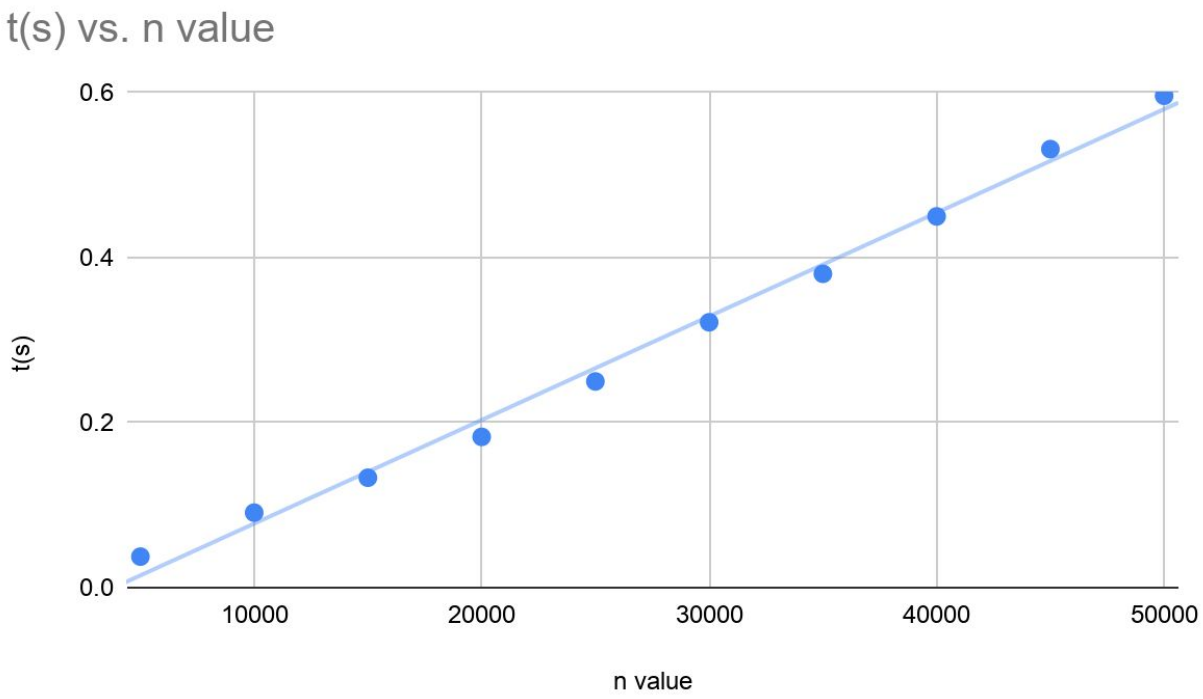
$$T(n) = O(n \log n)$$

* Solution of recurrence in part B

Problem #4:
B.

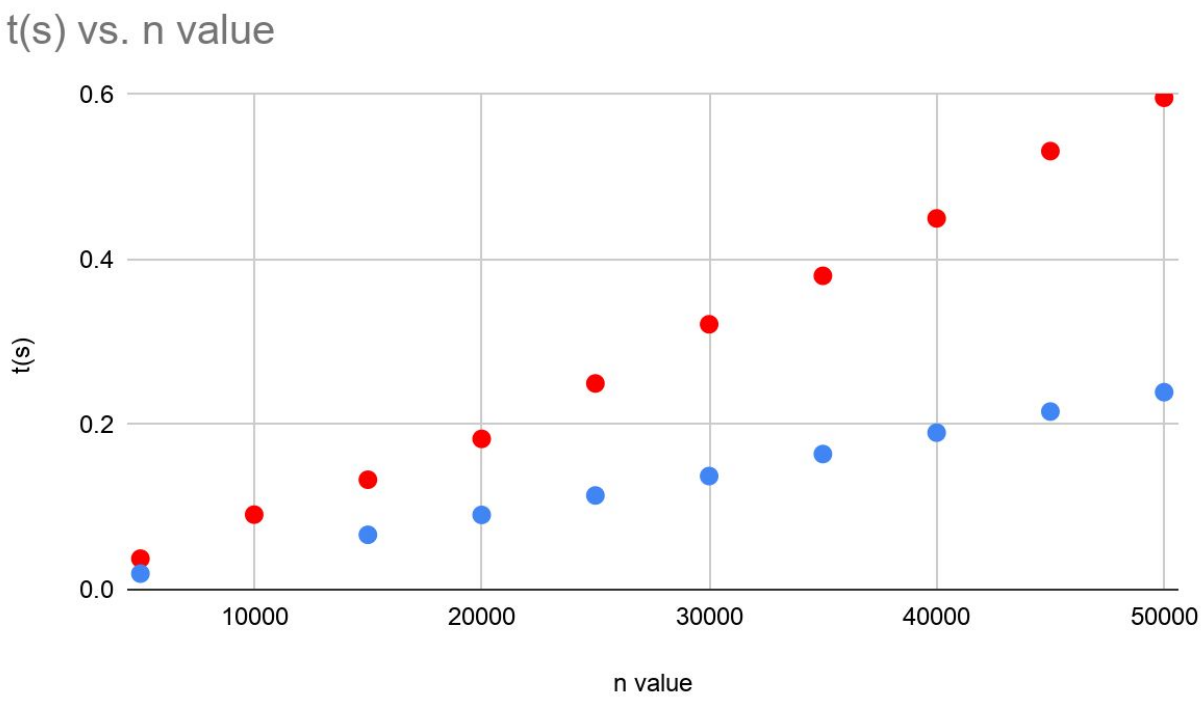
N value	t(s)
5000	0.03757095337
10000	0.09090304375
15000	0.1331989765
20000	0.1827030182
25000	0.2499029636
30000	0.3215858936
35000	0.3803420067
40000	0.4499289989
45000	0.5314919949
50000	0.5961170197

C.



Judging from my data and the graph, The equation it mostly follows would be a polynomial curve just like the mergesort from assignment 1. Possible nlog(n).

E.



This graph shows red being the mergesort3 and blue as the mergesort from assignment 1. When comparing the two I was really surprised because I figured mergesort3 would be faster. I averaged five separate time collections for each sorting technique and it remained the same. When comparing the two I see that they both seem to follow a similar curve. The only thing that I could see making the difference is the base of the nlog(n) for each curve. Mergesort3 has a log base of three compared to mergesorts log base 2.