Problem 1. Give the asymptotic bounds for T(n):

$$T(n) = 3T(n-1) + 1$$

We will solve by iteration:

$T(n) = 3T(n-1) + 1$

$T(n-1) = 3T(n-2) + 1$

$T(n) = 9T(n-2) + 2$

$T(n-2) = 3T(n-3) + 1$

$T(n) = 27T(n-3) + 3$

.

.*We continue subtracting* 1 *from* n, *until it* "bottoms out" at some base case T(1)

.

$T(n) = 3^n T(1) + c$

$\therefore T(n) = O(3^n + c) = O(3^n)$

Problem 2. Ternary search

a. Verbally describe and write pseudo-doe for ternary search

The ternary search algorithm will take as arguments: an array, a low index, a high index, and a target. The algorithm will then calculate two midpoints, the first midpoint representing the first third and the second midpoint the last third. The array from the first midpoint to the second midpoint will represent the middle third of the array.

The algorithm compare will determine which third to search by comparing the target value to value of the element stored at the mid points. It will recursively call itself, until it reach it's base case of an array length of one. The index

Pseudo Code:

```
Int ternarySearch(Array, low, high, target){
  if low == high  //This is the base case
    if Array[low] == target return 1
    else return 0

  else
    mid1 = low + (high – low)/3      //first third
    mid2 = high – (high – low)/3     //last third

   if target <= Array[mid1]
     ternarySearch(Array, low, mid1, target)
   else if target >= Array[mid2]
     ternarySearch(Array, mid2, high, target)
   else
     ternarySearch(Array, mid1, mid2, target)
```

b. Give the Recurrence
The four comparisons will take 4 units of time, and calculations for the 2 mid points take 2 units of time:

$$T(n) = T\left(\frac{n}{3}\right) + 4 = T\left(\frac{n}{3}\right) + c$$

c. Solve the Recurrence
Solve using the Master Method:

$a = 1, b = 3 => n^{log_3 1} = 1 \therefore 1 = f(n) = c$

$Case$ 2:

$T(n) = \theta(n^{log_3 1}\log(n)) = \theta(n^o logn) = \theta(logn)$

Problem 3. Design and analyze a divide and conquer algorithm that determines the minimum and maximum value in an unsorted list.
Note: The Algorithm I used was a slightly modified version from the Text book example of the divide and conquer algorithm.

a) Verbally describe and write pseudocode for min_and_max algorithm

The min_and_max algorithm will recursively call itself, dividing the array in half during each call, until it reaches a base case. The base case is when the array is a sub-array of length one. When the sub-array is length one, it is both minimum and maximum.

Each recursion will compare the left max and min, the right max and min, and the max and min of the crossing.

The Algorithm requires a subroutine which calculates the max and min crossing sub array between the left and right sub-arrays.

```
//Subroutine the calculate min and max crossing

Min and Max Crossing(A[], low, mid, high){
   left-sum-max = -infinity
   left-sum-min = inifinity

   max-left-index = mid
   min-left-index = mid
   sum = 0

   for i <- mid to low{
      sum += A[i]
      if sum > left-sum-max{
         left-sum-max = sum
         left-max-index = i
      }

      if sum < left-sum-min{
```

```
        left-sum-min = sum
        left-min-index =
    }
  }

  return(left-sum-max, left-sum-min, right-sum-max, right-sum-min,
        left-sum-max + right-sum-max, left-sum-min + right-sum-min)

  right-sum-max = -inifinity
  right-sum-min = inifinity

  max-right-index = mid
  min-right-index = mid
  sum = 0

  for i <- mid to high{
    sum += A[i]
    if sum > right-sum-max{
        right-sum-max = sum
        max-right-index = i
    }

    if sum < right-sum-min{
        right-sum-min = sum
        min-right-index = i
    }
```

In order to determine the running time, we need to determine the number of iterations in each loop.

$$Left\ loop\text{:}\ \ (mid - low + 1)$$
$$Right\ loop\text{:}\ (high - mid)$$
$$(mid - low + 1) + (migh - mid) = high - low + 1 = n$$
$$\therefore \theta(n)$$

```
//Min and Max Algorithm

Find Min and Max(A[], low, high){

  //Base Case is when there is only one element
  if low == high{
    return(low, high, A[low], low, high, A[low])

  }else{
    mid = (low + high)/2
    min = infinity
    max = -infinity

    //return values for the minimum
    ret_left_min_index = 0
    ret_right_min_index = 0
```

```
ret_min = 0

//return values for the maximum
ret_left_max_index = 0
ret_right_max_index = 0
ret_max = 0

min-left-low = 0
min-left-high = 0
min-left-sum = 0

max-left-low = 0
max-left-high = 0
max-left-sum = 0


min-right-low = 0
min-right-high = 0
min-right-sum = 0

max-right-low = 0
max-right-high = 0
max-right-sum = 0

min-cross-low = 0
min-cross-high = 0
min-cross-sum = 0

max-cross-low = 0
max-cross-high = 0
max-cross-sum = 0

(min-left-low, min-left-high, min-left-sum, max-left-low, max-left-high, max-left-sum) =
   Find Min and Max(A[], low, mid)


(min-right-low, min-right-high, min-right-sum, max-right-low, max-right-high, max-right-sum) =
   Find Min and Max(A[], mid, high)

(min-cross-low, min-cross-high, min-cross-sum, max-cross-low, max-cross-high, max-cross-sum) =
   Find Min and Max(A[], low, high)

if max-left-sum >= max-right-sum and max-left-sum >= max-cross-sum{
   ret_left_max_index = max-left-low
   ret_right_max_index = max-left-high
   ret_max = max-left-sum
}else if max-right-sum >= left-max-sum and max-right-sum >= max-cross-sum{
   ret_left_max_index = max-right-low
   ret_right_max_index = max-right-high
   ret_max = max-right-sum
```

```
}else{
   ret_left_max_index = max-cross-low
   ret_right_max_index = max-cross-high
   ret_max = max-cross-sum
}

if min-left-sum >= min-right-sum and min-left-sum >= min-cross-sum{
   ret_left_min_index = min-left-low
   ret_right_min_index = min-left-high
   ret_min = min-left-sum
}else if min-right-sum >= left-min-sum and min-right-sum >= min-cross-sum{
   ret_left_min_index = min-right-low
   ret_right_min_index = min-right-high
   ret_min = min-right-sum
}else{
   ret_left_min_index = min-cross-low
   ret_right_min_index = min-cross-high
   ret_min = min-cross-sum
}
```

b. Give the Recurrence

The base case of the "min and max" algorithm will take constant time, c. The function will recursively call itself while length of the array, n, is n > 2. Each recurrence call will split the array into two halves.

The Algorithm for min anx max crossing is linear, as was shown in a).

$$T(n) = \begin{cases} \theta(1) & if\ n = 1 \\ 2T(n/2) + \theta(n) & if\ n > 1 \end{cases}$$

c. Solve the recurrence

Solve using the master method:

$$2\left(\frac{n}{2}\right) + \theta(n)$$
$$a = 2, b = 2 => n^{log_2 2} = n^1 = \theta(n)$$
$$Therefore\ this\ is\ Case\ 2:$$
$$T(n) = \theta(n^{log_b a} logn) = \theta(n^{log_2 2} logn) = \theta(n^1 logn) = \theta(nlogn)$$

Comparison to an iterative method:

An iterative method, brute force method would require nested for loops. One loop iterating through each element, and the other iterating through subarray lengths from 1 to arr.length. Depending on the implementation. Running time will be $\theta(n^2)\ or\ \theta(n^3)$

$$\theta(nlogn) < \theta(n^3)$$

Problem 4: 4 Way Merge Sort

a. Pseudo Code

```
Mergesort(A[], start, end)
  If start < end
    Delta = end – start
    N1 = start + delta/4
    N2 = start + 2 * delta / 4
    N3 = start + 3 * delta / 4
    Mergesort(A[], start, N1)
    Mergesort(A[], N1 + 1, N2)
    Mergesort(A[], N2 + 1, N3)
    Mergesort(A[], N3 + 1, end)
    Merge4(A[], start, N1, N2, N3, end)
```

b. State the recurrence for the number of comparisons:

$$'start < end' comparison\ is\ c$$

$$assigning\ values\ to\ four\ variable - delta, N1, N2, N3, N4 = 4c$$

$$Four\ subproblem\ of\ length \frac{n}{4}:\ 4T(\frac{n}{4})$$

$$T(n) = 4T\left(\frac{n}{4}\right) + \theta(n)$$

Solve using the Master Method:

$$a = 4, b = 4 => n^{log_4 4} = n^1; f(n) = \theta(n^{log_b a})$$

$$\therefore Case\ 2$$

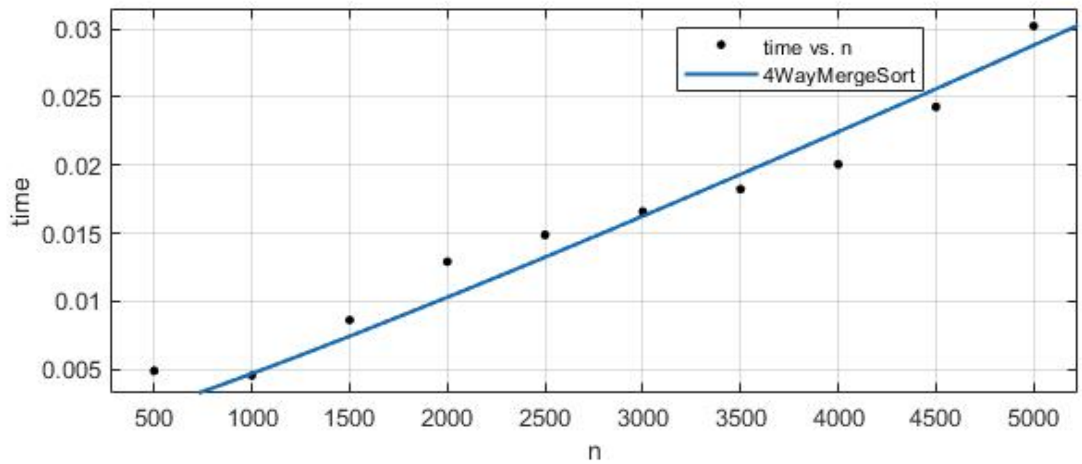
$$T(n) = \theta(n^{log_b a} logn) = \theta(n^{log_4 4} logn) = \theta(n * log * n)$$

Problem 5:

c. Collect running times:

length	mergeTime	inserTime	4wayMerge
500	0.002678	0.015104	0.00487
1000	0.005489	0.11206	0.004529
1500	0.008693	0.352841	0.00861
2000	0.011938	0.803589	0.012912
2500	0.015271	1.598262	0.014871
3000	0.018694	2.771835	0.016578
3500	0.021815	4.445526	0.018234
4000	0.02566	7.195229	0.020069
4500	0.028218	9.333934	0.024286
5000	0.031701	13.480052	0.030235

d. Plot data and fit a curve:



$$f(n) = a * n * \log(n)$$
$$a = 6.767e - 07$$

e. Combined Plot: