Homework #2

1. 𝑇(𝑛) = 3𝑇(𝑛 −1) +1
2. a.) Ternary search first checks to see if the right index is greater than or equal 1. If it were less than 1, that would mean that the search element was not found, and -1 would be returned. Next, it finds the 2 midpoints that split the data into approximate thirds. It checks the 2 midpoints for equality on the search element. If found, it returns the index of the search element. If the search element is less than the first midpoint, or in other words, in the first third part of the sorted array, ternary search is called again on the first third part of the sorted array. If the search element is greater than the second midpoint, or in other words, in the last third part of the sorted array, ternary search is called again on the last part of the sorted array. If the search element is in neither the first or last thirds of the sorted array, then ternary search is called on the middle part of the sorted array.

PSEUDOCODE

ternarySearch(array, Lindex, Rindex, x)  
 if Rindex >= 1  
 mid1 = Lindex + (Rindex - Lindex) / 3  
 mid2 = Rindex - (Rindex - Lindex) / 3  
 if array[mid1] == x  
 return mid1  
 if array[mid2] == x  
 return mid2  
 if x < array[mid1]  
 return ternarySearch(array, Lindex, mid1 - 1, x)  
 else if x > array[mid2]  
 return ternarySearch(array, mid2 + 1, Rindex, x)  
 else  
 return ternarySearch(array, mid1 + 1, mid2 - 1, x)

return -1

b.)

c.) Master Method:  
 a = 1, b = 3,

Case 2:

The running times of Binary and Ternary search are both .

1. a.) If the array size is only 1, then return the 1 element as the max and min. If the array size is 2, then make one comparison to see which element is bigger and smaller. If the array is greater than 2, then recursively split the array in half to see what the max and min are from each half. Compare the 2 halves min and max to see the absolute min and max and return the max, min pair.  
   The following pseudocode based on the code from: <https://www.geeksforgeeks.org/maximum-and-minimum-in-an-array/>

PSEUDOCODE

min\_and\_max(array, Lindex, Rindex)

if arraySize == 1

return element as (max, min)

else if arraySize == 2

if array[0] > array[1]

max = array[0]

min = array[1]

else

max = array[1]

min = array[0]

return (max, min)

else

min\_and\_max(array, Lindex, (Lindex+Rindex/2))

min\_and\_max(array, (Lindex+Rindex/2), Lindex)

compare 2 max’s to see which is bigger

compare 2 min’s to see which is smaller

return (max, min)

b.)

c.) Master Method:  
 a = 2, b = 2,

Case 1:

The running times of recursive min\_and\_max algorithm and an iterative algorithm for finding min and max are both .

1. a.)

If array is greater than 1, sort. Otherwise, no need to sort  
 Split the array into 4 quarters

Keep splitting arrays into quarters until only one or less element  
 in each quarter(recursion)

If there are elements not yet compared in the first two quarters,  
 compare and put lower/equal value in next spot in left array  
   
 If there are still elements in the q1 array  
 Put them in the next spot in left array  
   
 If there are still elements in the q2 array  
 Put them in the next spot in left array  
   
 If there are elements not yet compared in the last two quarters,  
 compare and put lower/equal value in next spot in right array  
  
 If there are still elements in the q3 array  
 Put them in the next spot in right array

There are still elements in the q4 array  
 Put them in the next spot in right array  
   
 If there are elements not yet compared in both halves, compare and  
 put lower/equal value in next spot in array

If there are still elements in the left array  
 Put them in the next spot in array  
   
 If there are still elements in the right array  
 Put them in the next spot in array

b.)