Chapter-3:

R-3.8: Order the following functions by asymptotic growth rate:

1. **2^10**
2. **4n**
3. **2^n**
4. **N^3**
5. **2logn**
6. **Nlogn**
7. **N^2+10n**
8. **3n+100logn**
9. **4nlogn+2n**

C-3.36: Describe an efficient algorithm for finding the ten largest elements in a sequence of size n. What is the running time of your algorithm?

**Recursively call a function that loops through an array and every time an element is found it’s added to a new array that holds the 10 largest elements. Once 10 elements are in the array any time an element that is larger than an element found in the new array then that element is replaced. The function calls stop once the original array is empty. What is left will be the largest 10 elements in an array.**

**The running time would be Ologn**

C-3.45: A sequence S contains n − 1 unique integers in the range [0,n − 1], that is, there is one number from this range that is not in S. Design an O(n)- time algorithm for finding that number. You are only allowed to use O(1) additional space besides the sequence S itself.

**Set up an array that contains every number in the range and loop through the original sequence of numbers matching each element in the range. When the number in the range isn’t found in the array you can halt the loop and you found the number.**

Chapter-4:

R-4.1: Describe a recursive algorithm for finding the maximum element in a sequence, S, of n elements. What is your running time and space usage?

**Call a function that reads an element at an index in the sequence and if it isn’t found recursively call the function again with the next indexed element. Once it is found then the function calls are done.**

R-4.3: Draw the recursion trace for the computation of power(2,18), using the repeated squaring algorithm, as implemented in Code Fragment 4.12

**There isn’t any recursion in the function.**

C-4.19: Write a short recursive Python function that rearranges a sequence of integer values so that all the even values appear before all the odd values.

def even\_to\_odd(data, eData, oData, index):  
 if index == len(data):  
 data = eData + oData  
 return  
  
 value = data[index] % 2  
 if value == 0:  
 eData.append(data[index])  
 else:  
 oData.append(data[index])  
  
 even\_to\_odd(data, eData, oData, index+1)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 data = [0,5,8,1,2,3,11,29,48,65,97,1321,11,54,21]  
 eData = []  
 oData = []  
 even\_to\_odd(data, eData, oData, 0)