CSC 3110 – Homework 2

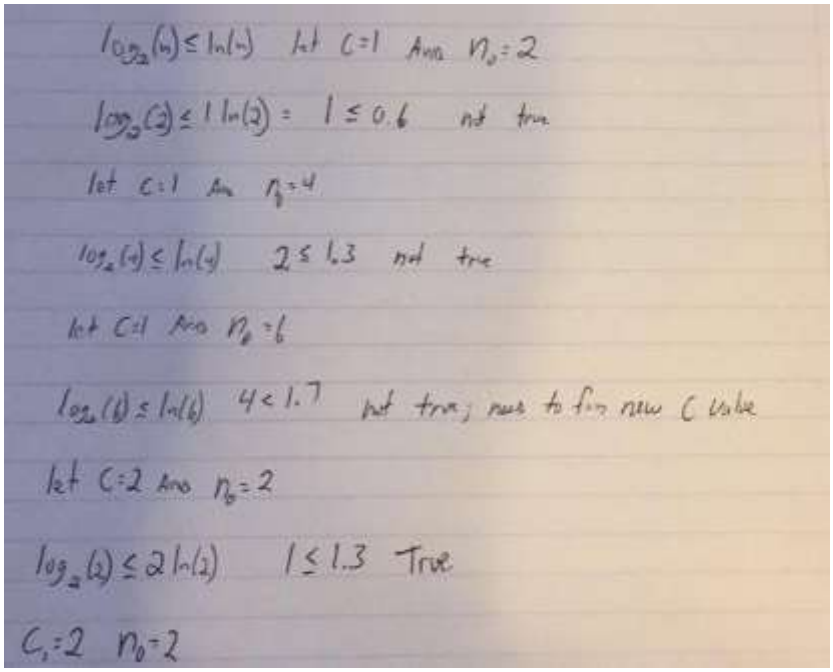
1. Find larger order of growth
   a. $\lim_{n\to\infty} (\log_2 n)^2 / (\log_2 n^2)$. This can be rewritten as

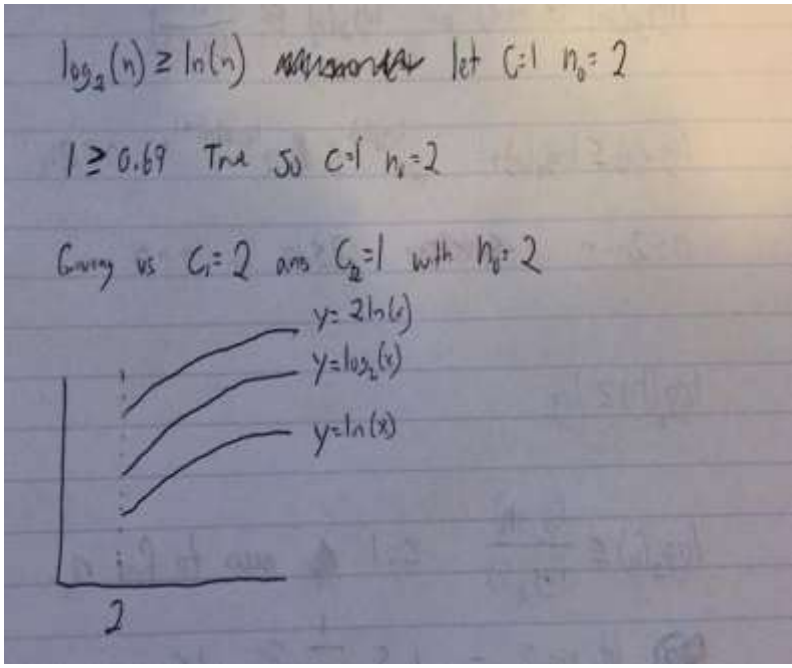
      $\lim_{n\to\infty} n^2/(2\log_2 n)$. Both approach infinity as n gets large but how we know how orders of growth for functions are ranked we know n^2 will always approach infinity faster than nlog(n). This means $(\log_2 n)^2$ has a larger order of growth between the two functions. Or it can be said $(\log_2 n)^2 \in \Omega(\log_2 n^2)$.

   b. $\lim_{n\to\infty} 3^{3n+1}/3^{3n}$. This can be rewritten as

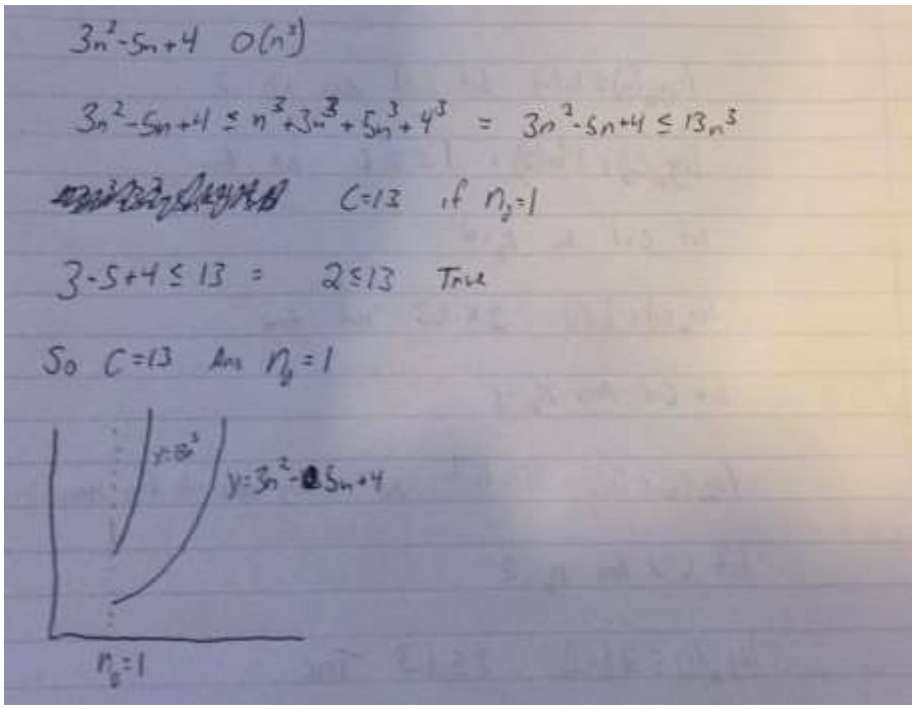
      $\lim_{n\to\infty} 3^n/3^n$. By looking at this we can see that both functions grow at the exact same pace. No matter how large n get, one does not grow at a faster or slower rate than the other. Or it can be said $3^{3n+1} \in \Theta(3^{3n})$.

2. Use the formal definitions of O, Ω, and Θ to prove the following
   a. $\text{Log}_2 n \in \Theta(\ln n)$.

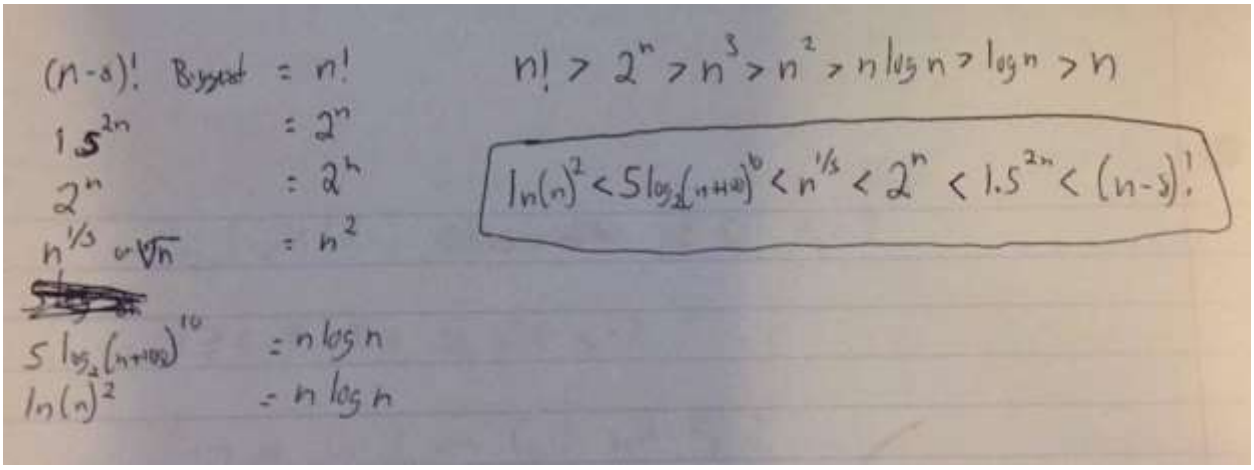




   b. $3n^2-5n+4 \in O(n^3)$

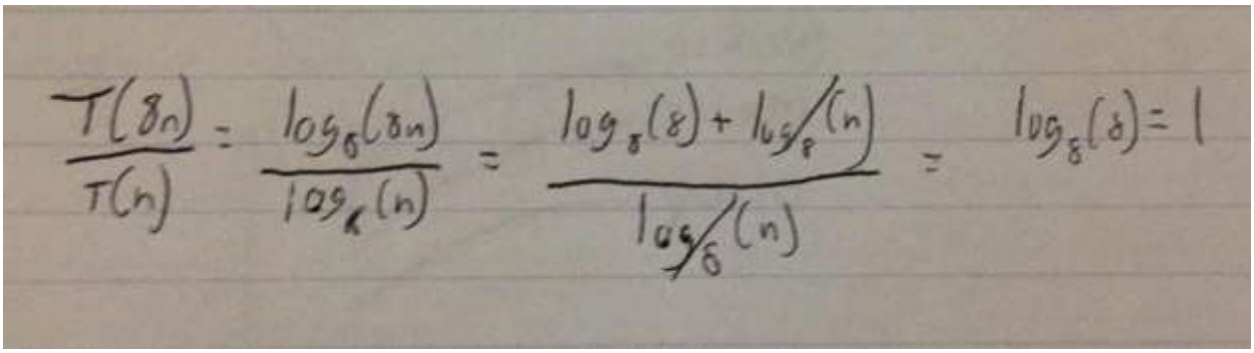


3. Order the following functions according to their order of growth from lowest to highest

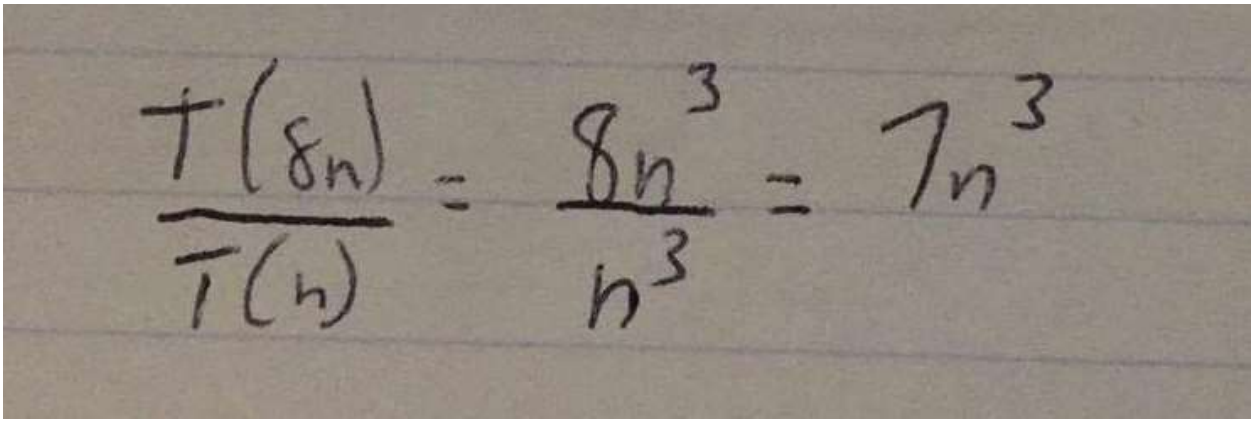


4. For the following functions, indicate how much the functions value will change if its argument is increased eight-fold

a. $\text{Log}_8(n)$

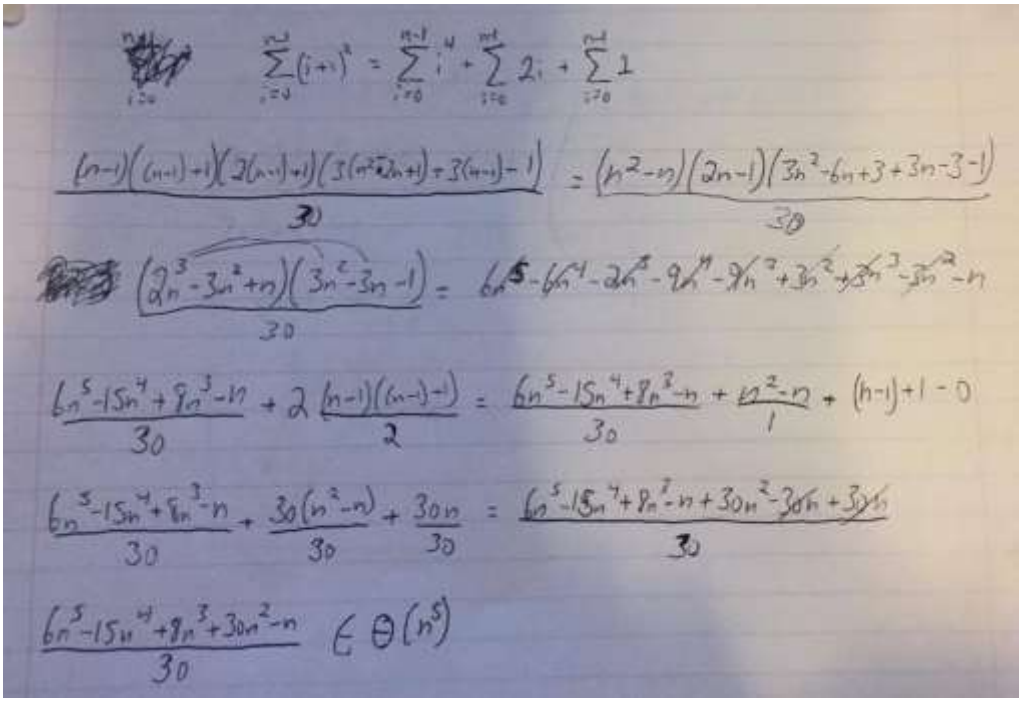


b. n^3

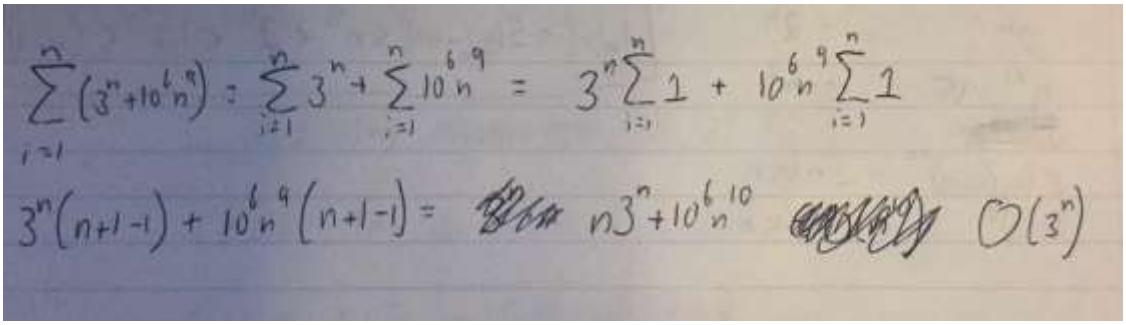


5. Find the order of growth of the following sums.

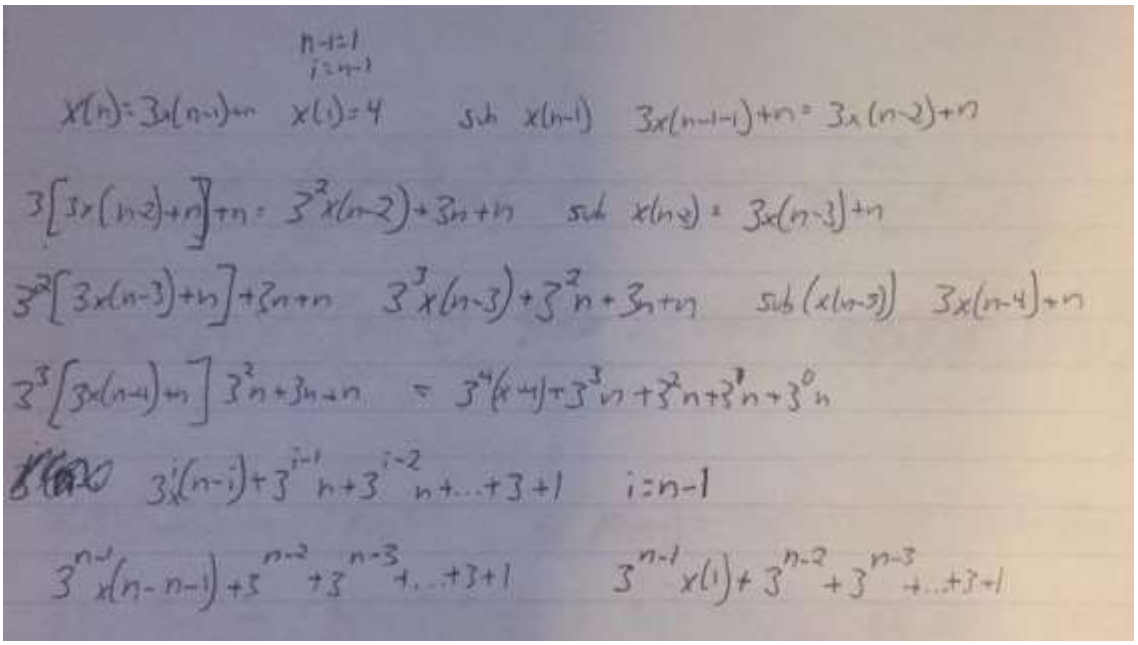
a. $\sum_{i=0}^{n-1}(i^2+1)^2$



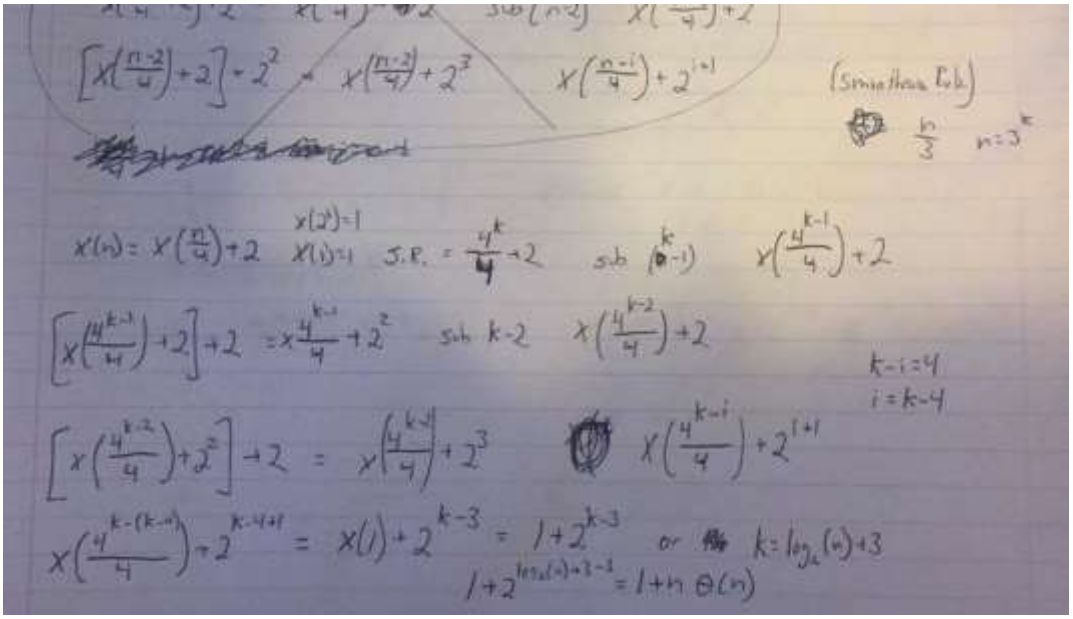
b. $\sum_{i=0}^{n}(3^n + 10^6 n^9)$



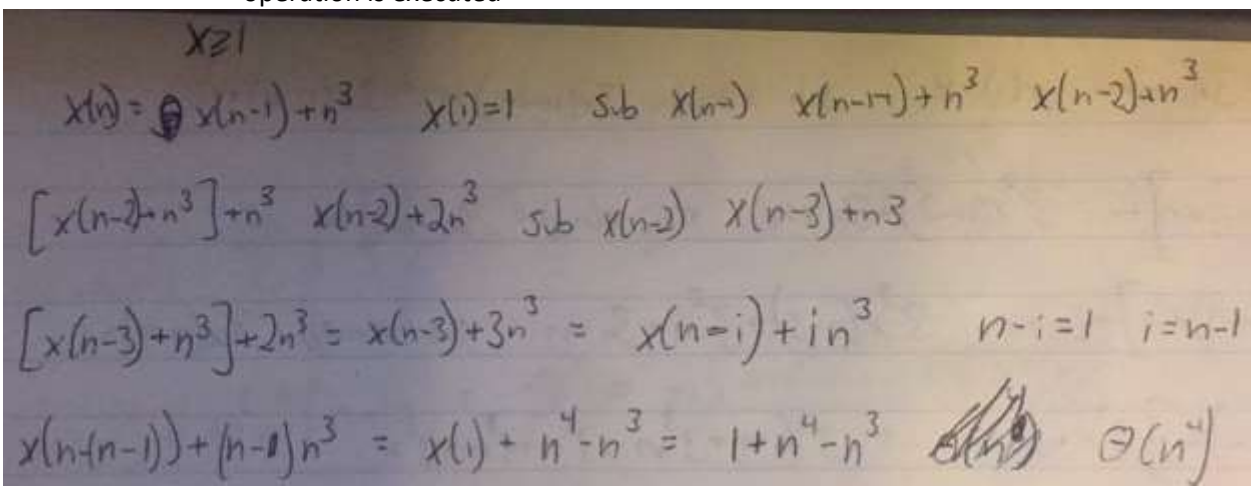
6. Solve the following recurrence relations:
   a. x(n) = 3x(n − 1) + n for n > 1, x(1) = 4



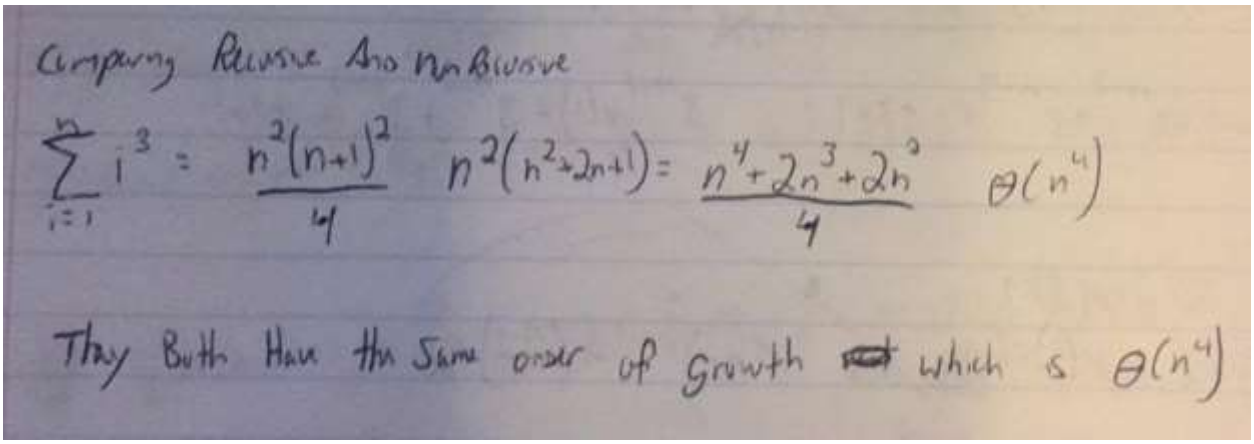
   b. x(n) = x(n/4) + 2 for n > 1, x(1) = 1 (Hint: solve for n = 4k)



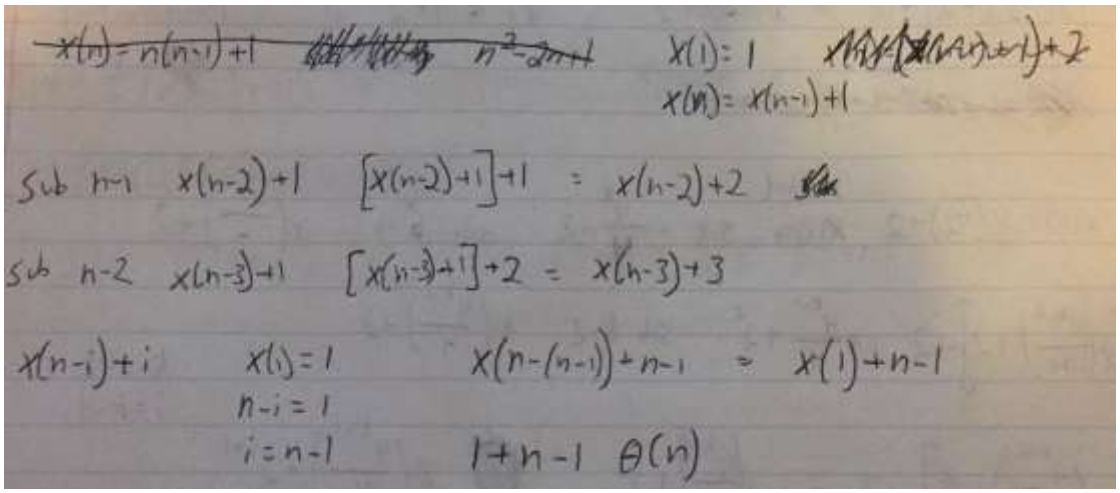
7. Consider the algorithm
    a. Set up and solve a recurrence relation for the number of times the algorithm's basic operation is executed



    b. How does this algorithm compare with the straightforward nonrecursive algorithm for computing this function?

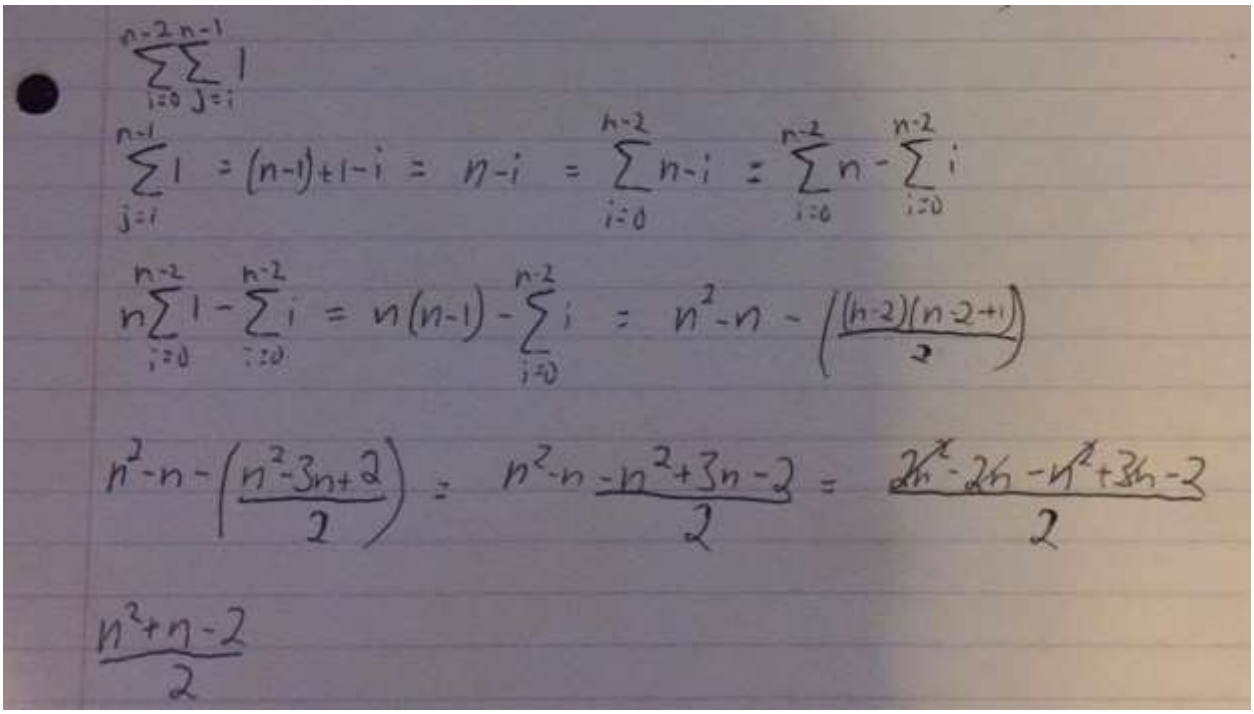


8. Consider the algorithm
    a. What does this algorithm compute?
        i. This takes in an array and recursively calls until the first element is assigned to temp. Then if temp is less than or equal to the last element in the list return temp otherwise return the last element
    b. Set up and solve a recurrence relation for the number of times the algorithms basic operation is executed.



9. Consider the algorithm
    a. What does this algorithm compute?
        i. Checks the symmetry of the matrix
    b. What is its basic operation
        i. Comparison in the inner most for loop

c. How many times is the basic operation executed



d. What is the efficiency class of this algorithm?
   i. $\Theta(n^2)$.
e. Are the best average and worst-case time efficiencies the same or different? Justify your answer
   i. They are the same because there are no breaks anywhere in the code. The full code will run no matter the input size

10. The range of a finite nonempty set of n real numbers S is defined as the difference between the largest and smallest elements of S. For each representation of S given below, describe in English an algorithm to compute the range. Indicate the time efficiency classes of these algorithms use the appropriate notation
    a. An unsorted array - $\Theta(n)$
       i. Traverse the array comparing each to the value of "small", if its smaller then assign it. Then repeat that task instead looking for the largest value. Then return the difference of them.

```
int large=0
int small=0
for i =0 to n-1
   if(A[i]<=small)
         small=A[i]
for j=0 to n-1
   if(A[i]>=large)
         large=A[i]
return large-small
```

    b. Sorted Array

i. Since the array is sorted we already know where the smallest and largest values are. Simply assign them and return them.
(ascending order) – O(1)

```
Int large = A[n-1]
Int small = A[0]
```

(descending order) – O(1)

```
Int large = A[0]
Int small = A[n-1]
```

c. Sorted singly linked list – O(n)
   i. Since we know its sorted the same approach as the above method can be applied. The very first node will be assigned to small or large depending on ascending or descending sorting and the list will traverse until it reaches the last node (next==NULL). Once there, the value of that will be assigned to small or large again depending on ascending or descending order

```
Node *temp=this
Small=this.vaule
While(temp->next!=NULL)
   Temp=temp->next
Large=temp.value
```

d. Binary search tree – $O(n^2)$
   i. BST are sorted as a tree data structure with the right child greater than itself and the left child less than itself. So, to find the max value of the tree we'd want to traverse the right child of the tree as long as the next right child is not equal to NULL. If it is we know we've found the largest value in the tree. To find the smallest value we'd want to traverse the left child as long as the next left child isn't NULL. If it is we've found the smallest value.

```
Node *temp=this
While(temp->right!=NULL)
   Temp=temp->right
Large=temp->value
Temp=this
While(temp->left!=NULL)
   Temp=temp->left
Small=temp.value
Return large-small
```