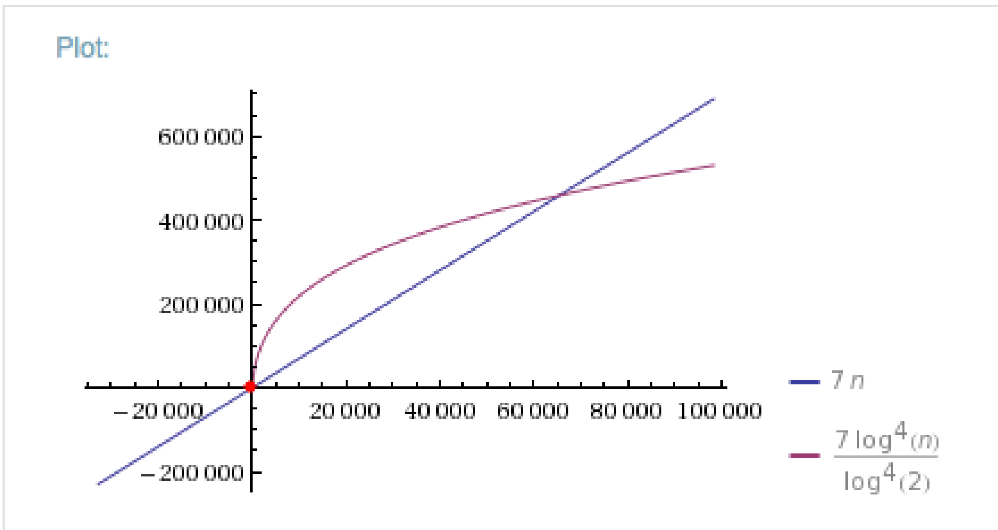
1. **Compare Functions**: (10 points) What is the smallest integer value of $n > 3$ such that an algorithm whose running time is $7n$ runs *slower than* an algorithm whose running time is $7(\log_2 n)^4$ on the same machine? Justify your answer. (Hint: You may write a program, draw a plot, or/and proof)

   http://www.wolframalpha.com/input/?i=7n+intersect+7((log2%5B+n%5D+)%5E4)

   so the number is 65536

2. **Pseudocode and Loop Invariant**: (15 points) textbook, Exercise2.1-3, p22, Searching Problem

```
Linear_Search (A, v)
        for i = 1 to A.length
                if A[i] == v
                        return i
        return NIL
```

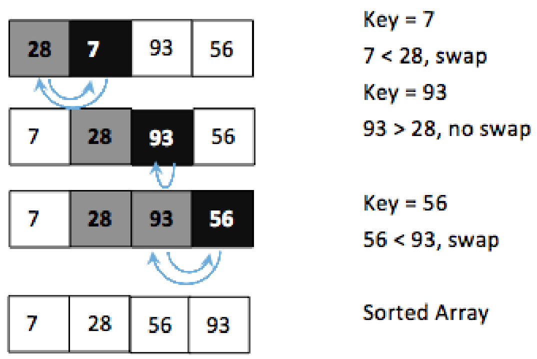
   At the start of the each iteration, loop invariant: v not ∈ {A[1], . . . , A[i-1]}.

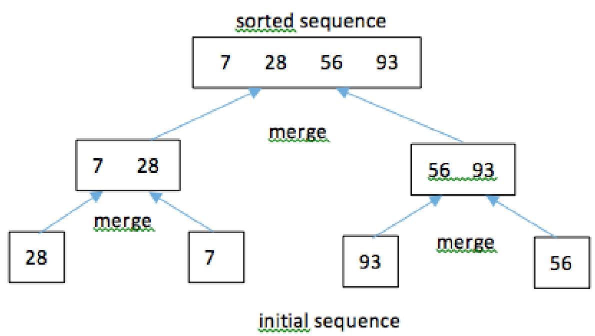
   • **Initialization**: At the beginning of the first iteration, we have i = 1, so the set is empty (no element).
   • **Maintenance**: The code haven't returned the value i yet, v would not in {A[1], ..., A[i-1]}, so the invariant is maintained by the loop.
   • **Termination**: Since the loop is a for loop over a finite sequence 0 ...len(A)-1, the loop will always terminate. If the algorithm finds v in the array A, we have A[i] = v, and the algorithm returns the index I is correct. Otherwise, the loop terminates after len(A) iterations, in which case the invariant states that

v not ∈ {A[1], . . . , A[len(A)]}, which is the whole array A, so we can guarantee that NIL, the value the algorithm returns, is correct.

Therefore, the function linear search is correct by the loop invariant.

3. **Sorting Algorithms**: (20 points) Using textbook Figure 2.2 and Figure 2.4 as models to illustrate the operations of Insertion_Sort and Merge_Sort on the array A = <30, 7, 95, 56>





4. **Analysis**: (20 points) There is a mystery function called Mystery(n) and the pseudocode of the algorithm is shown as below. Please analyze the worst-case asymptotic execution time of this algorithm using the method we learn in the class. Express the execution time as a function of the input value n. Assume that $n = 3^k$ for some positive integer $k \geq 1$. Justify your answer.
Hint:
    (a) Draw a recursion tree to help with your analysis.
    (b) Appendix A may help with your calculation

```
Mystery(n)
    if n≤1
        return 1
    for i=1 to 5
        for j = 1 to n^2
            print "this is a recursive call."
```

```
        Mystery (n/3)
        Mystery (n/3)
        Mystery (n/3)
```

	Mystery (n)	cost	times
1	$if\ n \leq 1$	$c1$	1
2	$return\ 1$	$c2$	1
3	$for\ i = 1\ to\ 5$	$c3$	6
4	$for\ j = 1\ to\ n^2$	$c4$	$5(n^2 + 1)$
5	$print\ "Welcome\ to\ recursion!"$	$c5$	$5(n^2)$
6	$Mystery(n/3)$	$T(n/3)$	1
7	$Mystery(n/3)$	$T(n/3)$	1
8	$Mystery(n/3)$	$T(n/3)$	1

$T(n) = c1 + c2 + 6c3 + c4 * 5(n^2 + 1) + c5 * 5(n^2) + 3T(n/3) = cn^2 + 3T(n/3)$

This recursive solution then becomes.....

$$T(n) = cn^2 + \left(\frac{cn^2}{3}\right) + \left(\frac{cn^2}{9}\right) + \left(\frac{cn^2}{27}\right) + \cdots + \left(\frac{cn^2}{3^i}\right)$$

$$\leq cn^2 \sum_{i=0}^{\infty} \frac{1}{3^i}$$

The summation is geometric and converges to 3/2

$$\leq \frac{3}{2} cn^2$$

Thus, the worst-case asymptotic execution time of Mystery is $T(n) = \Theta(n^2)$

5 a

(5) Divide & Conquer

(a) (20 points) 2.3-5

Write pseudocode for binary search

Input: A is an array of values, "low" is the low point in the array, "high" is the high point in the array, and v is the value being sought.

Output: index in Array that holds the value, or NIL if not found

```
Binary-Search(A, v, low, high)
    if (low > high)
        return NIL
    mid = ⌊(low + high)/2⌋
    if A[mid] = v
        return mid
    else if A[mid] > v
        return Binary-Search(A, v, low, mid - 1)
    else
        return Binary-Search(A, v, mid + 1, high)
```

Because $T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(\frac{n}{2}) + \Theta(1), & n > 1 \end{cases}$

And because the algorithm creates a recursive binary tree that has n levels where at each level (if the root is level 0), there are 2^i nodes, then $n = 2^i$ and $i = \log_2 n$. Thus, if there n levels with $\log_2 n$ nodes, the time complexity in the worst case is $O(\log_2 n)$.

5 b.

```
MERGE-SORT S        // O(n log n)
for i = [1..n]
  if BINARY-SEARCH S (x-i) != NIL return true
end for
return false
```

since the for loop iterates n times, our time is:

$$\underbrace{n \log n}_{\text{merge sort}} + \underbrace{n}_{\text{for loop}} \underbrace{\log n}_{\text{binary search}} = \Theta(n \log n)$$