1. Suppose we are comparing insertion sort and merge sort implementations on the same machine. For inputs of size n, insertion sort runs in 6*n^2 steps, while merge sort runs in 42*n*lg(n) steps. Does merge sort always prevail over insertion sort? If not, for which values of n does insertion sort beat merge sort? (2pts)

$$6*n^2 < 42*n*\log(n)$$
$$n < 7*\log(n)$$
$$n/7 < \log(n)$$
$$2^{(n/7)} < n$$

To find the values of n where insertion sort beats merge sort, I plotted the functions and found the intersections, which were n= 1.1 and 36.3. Since we look at this discretely, the required range of n is 2<=n<=36.

2. What is the time complexity of each code below? Show your work. (3pts)

(1)
```
count=1                          O(1)
while count<n                    O(log(n))
        count=count*3                    O(1)
```
T(n)= O(log(n))

(2)
```
i=0                              O(1)
for i<n                          O(n)
        j=0                              O(1)
        for j<n                          O(log(n))
                j=j*2                            O(1)
        i++                              O(1)
```
T(n)= O(n*log(n))

(3)
```
i=0                              O(1)
for i<n                          O(n)
        j=i                      O(1)
        for j<=i and j<n         O(n)
                j++
        i++
```
T(n)= O(n^2)

3. Examine each function in the left column. Determine whether the functions listed in the first row are their lower bound, upper bound, or asymptotic bound by filling in with the corresponding symbols. If multiple relations apply, choose the most rigorous one. The second row shows an example. (3pts)

	n	n^2	2^n
Example: n^2	Ω (lower bound)	Θ (same shape)	O (upper bound)
n^n	Ω	O (as it goes to infinity)	Ω (as it goes to infinity)
n*log(n^4)	Ω (mostly)	O (mostly)	O
(n-2)!	Ω (as it goes to infinity)	O (as it goes to infinity)	O (as it goes to infinity)
2^(log(n))	O (mostly)	O (mostly)	O

4. To deal with the case that one subarray is shorter than the other one, when we design MERGE (A, p, q, r) to merge L and R into a sorted array, at the end of each subarray, we add an extra element ∞ to serve as a sentinel. Rewrite MERGER without using sentinels. Instead, the rewritten algorithm should stop comparison when all elements from either L or R are all done and copy the remainder of the other array into A. (2pts)

```
MERGER(A, p, q, r)
        n = q-p+1
        m = r-q
        for i = 1 to n
                L[i] = A[p+i-1]
        for j=1 to m
                R[j] = A[q+j]
        if n<m
                x=2*n
        if n>m
                x=2*m
        i=1
        j=1
        for k=p to x
                if L[i] <= R[j]
                        A[k] = L[i]
                        i = i + 1
                else A[k] = R[j]
                        j = j + 1
        if n<m
                y=m-n
                while k<m
```

```
                A[k]=R[y]
                y++
                k++
if n>m
        y=n-m
        while k<n
                A[k]=L[y]
                y++
                k++
```