1) Write a boolean function that is given a binary tree and returns true if and only if the tree has an odd number of nodes. An empty tree is considered to have an even number of nodes.

Notes:
The function should have just one argument, a pointer to the root.
No global variables may be used.
No additional functions may be defined. You may not count the number of nodes.

2) Consider the following insertion sort algorithm.

```c
void insertion_sort(element a[], int n)
// Put a[0]..a[n-1] into ascending order by insertion sort.
{
    for (int k = 1; k < n; k++) {
        // At this point, a[0]..a[k-1] are already in order.
        // Insert a[k] where it belongs among a[0]..a[k].
        // You need to write code for this insertion as the body of the for-k loop.

    }//endfor k
}
```

a) Write the code for the body of the for-k loop to complete the insertion sort routine.
b) Count the precise best case and worst case number of “element comparisons” in your insertion sort routine. Your answers should be functions of $n$ in closed form. Note that “closed form” means that you must resolve all sigmas and …’s. Asymptotic answers (such as ones that use big-oh, big-theta, etc.) are not acceptable.
3) For each function with input argument \( n \), determine the asymptotic number of “fundamental operations” that will be executed. Note that \( fc \) and \( fd \) are recursive. Choose each answer from among the following. You do not need to explain your choices.

\[ \Theta(1) \quad \Theta(\log n) \quad \Theta(n) \quad \Theta(n \log n) \quad \Theta(n^2) \quad \Theta(n^2 \log n) \quad \Theta(n^3) \quad \Theta(2^n) \quad \Theta(n!) \]

a) 
void fa(int n) {
    for(i = n; i > 0; i = i/2)
       Perform 1 fundamental operation;
    //endfor i
}

b) 
void fb(int n) {
    for(i = 1; i <= n; i++) {
        for(j = 1; j < n; j++)
           Perform 1 fundamental operation;
        //endfor j
        for(k = 1; k <= i; k++)
           Perform 1 fundamental operation;
        //endfor k
    }//endfor i
}

c) 
void fc(int n) {
    if (n > 1){
        fc(n/2);
        fc(n/2);
        Perform n-1 fundamental operations;
    }//endif
}

d) 
void fd(int n) {
    if (n > 1){
        fd(n/2);
        Perform n-1 fundamental operations;
    }//endif
}