1. For each of the following code segments, find a function \( f(n) \) with the property that the number of statements executed by the segment is \( O(f(n)) \). Your functions \( f(n) \) should be as simple as possible.

(a) \[
\text{sum} = 0; \\
\text{for (int i = 0; i < n; i++)} \\
\quad \text{sum++;}
\]

\( f(n) = n \).

(b) \[
\text{sum} = 0; \\
\text{for (int i = 0; i < n; i++)} \\
\quad \text{for (int j = 0; j < n; j++)} \\
\quad \quad \text{sum++;}
\]

The inner loop is \( \Theta(n) \). So \( f(n) = n^2 \).

(c) \[
\text{sum} = 0; \\
\text{for (int i = 0; i < n; i++)} \\
\quad \text{for (int j = 0; j < n*n; j++)} \\
\quad \quad \text{sum++;}
\]

The inner loop is \( \Theta(n^2) \). So \( f(n) = n^3 \).

(d) \[
\text{sum} = 0; \\
\text{for (int i = 0; i < n; i++)} \\
\quad \text{for (int j = 0; j < i; j++)} \\
\quad \quad \text{sum++;}
\]

The inner loop is \( \Theta(i) \). So the entire code segment has runtime proportional to

\[
\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2},
\]

and \( f(n) = n^2 \).

(e) \[
\text{sum} = 0; \\
\text{for (int i = 0; i < n; i++)} \\
\quad \text{for (int j = 0; j < i*i; j++)} \\
\quad \quad \text{for (int k = 0; k < j; k++)} \\
\quad \quad \quad \text{sum++;}
\]
The innermost loop is $\Theta(j)$. So the for j loop is
\[
\Theta \left( \sum_{j=0}^{i^2-1} j \right) = \Theta \left( \frac{i^2(i^2 - 1)}{2} \right),
\]
which is the same as $\Theta(i^4)$. So by the fact that
\[
\sum_{t=1}^{n} t^k = O(t^{k+1}),
\]
we can conclude that $f(n) = n^5$.

(f) \[
\text{sum = 0;}
\text{for (int i = 1; i <= n; i++)}
\text{for (int j = 1; j <= i*i; j++)}
\text{if (j % i == 0)}
\text{for (int k = 0; k < j; k++)}
\text{sum++;}
\]
The innermost loop will only be executed when $j$ is a multiple of $i$, which will occur precisely $i$ times. The remaining passes through the for j loop will simply execute the test and increment $j$. So we can rewrite the code so that it has the same runtime (up to big-Theta) as follows.
\[
\text{sum = 0;}
\text{for (int i = 1; i <= n; i++)}
\text{for (int j = 1; j <= i; j++)}
\text{j_prime = i*j;}
\text{for (int k = 0; k < j_prime; k++)}
\text{sum++;}
\text{Code that has runtime i-1;}
\]
The runtime of the body of the for j loop is $\Theta(i + j_{\text{prime}}) = \Theta(i + ij)$. So the runtime of the for j loop is
\[
\Theta \left( \sum_{j=1}^{i} (i + ij) \right) = \Theta \left( i^2 + \frac{i^2(i + 1)}{2} \right),
\]
which is $\Theta(i^3)$. So by the result on sums of powers of $i$, we see that $f(n) = n^4$.

2. Problem 5.2, page 161. Define the degree of a node as the number of its nonempty children. Prove by induction that the number of degree 2 nodes in any binary tree is one less than the number of leaves. (You can assume that the tree is nonempty.)

We use induction on $n$, the number of nodes in the binary tree. If $T$ is any binary tree, we use the notation $l(T)$ and $d(T)$ to denote the number of leaves of $T$ and the number of degree 2 nodes of $T$, respectively. So we wish to show that for any nonempty binary tree $T$ $l(T) = d(T) + 1$. 
Base case. If $T$ is a binary tree with 1 node, then $T$ consists of a single leaf, and no other nodes. In particular, it has no degree 2 nodes. So in this case $l(T) = d(T) + 1$.

Induction hypothesis. Suppose then that $n_0$ is a positive integer with the property that if $T$ is any binary tree with $n = n_0(\geq 1)$ nodes, then $l(T) = d(T) + 1$.

Induction step. Suppose now that $T$ is a binary tree with $n = n_0 + 1 \geq 2$ nodes. Then (as we saw in class) $T$ has a leaf $L$. By deleting $L$ from $T$, we get a new tree $T'$ with $n = n_0 \geq 1$ nodes. So by the induction hypothesis, $l(T') = d(T') + 1$. We consider two cases:

(a) $L$ has no sibling in $T$, and
(b) $L$ has a sibling in $T$.

Note that since $T$ has at least two nodes, $L$ is not the root of $T$, and hence $L$ has a parent $P$ in $T$. In case (a), $P$ has degree 1 in $T$, and hence it’s a leaf of $T'$. Thus, $l(T) = l(T')$ and $d(T) = d(T')$. So in this case, we have $l(T) = d(T) + 1$.

In case (b), $P$ has degree 2 in $T$, and hence degree 1 in $T'$. Thus $l(T) = l(T') + 1$ and $d(T) = d(T') + 1$. So by the induction hypothesis, we have that

$$l(T) = l(T') + 1 = d(T') + 1 + 1 = d(T) + 1.$$  

3. Problem 5.5, page 162. Write a function named search that takes as input a binary tree (not a BST!) and a value $K$, and returns true if value $K$ appears in the tree and false otherwise. Your function should have the following prototype.

```java
boolean search(BinNode rt, int K)

private boolean search(BinNode currNode, int key) {
    if (currNode == null)
        return false;
    Elem currElem = (Elem) currNode.element();
    if (currElem.key() == key)
        return true;
    boolean hasKey = search(currNode.left(), key);
    if (hasKey)
        return true;
    else
        return search(currNode.right(), key);
} // search
```

4. Problem 5.7, page 162. Define the internal path length for a tree as the sum of the depths of all internal nodes, while the external path length is the sum of the depths of all the leaves. Prove by induction that if $T$ is a full binary tree with $n$ internal nodes, $I$ is $T$’s internal path length, and $E$ is $T$’s external path length, then $E = I + 2n$ for all $n \geq 0$.

We use induction on $n$, the number of internal nodes in the binary tree. If $T$ is any binary tree, we use the notation $I(T)$ and $E(T)$ to denote the internal and external path lengths of $T$, respectively.
Base case. If a binary tree $T$ has no internal nodes, it can either be empty, or it can have a single node, the root. In either case, $E(T) = I(T) = 0$, and we see that for this case $E(T) = I(T) + 2n$.

Induction hypothesis. Suppose then that $n_0$ is a nonegative integer with the property that any full binary tree $T$ with $n = n_0 (\geq 0)$ internal nodes satisfies the formula $E(T) = I(T) + 2n$.

Induction step. Suppose now that $T$ is a full binary tree with $n = n_0 + 1 \geq 1$ internal nodes. As discussed in class, $T$ contains at least one internal node $P$, both of whose children, $L$ and $L'$, are leaves. Consider the tree $T'$ obtained from $T$ by deleting $L$ and $L'$ from $T$. Then, in $T'$ the node $P$ is a leaf. So $T'$ contains $n = n_0$ internal nodes. Furthermore, if $Q$ is any node of $T'$ different from $P$, the children of $Q$ in $T'$ are the same as the children of $Q$ in $T$. Hence $T'$ is full, and we can apply the induction hypothesis to conclude that $E(T') = I(T') + 2n_0$.

If $d(Q)$ denotes the depth of a node of $T$, we know that $d(L) = d(L') = d(P) + 1$. Thus, we have that

$$E(T) = E(T') - d(P) + 2d(L)$$
$$I(T) = I(T') + d(P).$$

So we have that

$$E(T) = I(T') + 2n_0 - d(P) + 2d(L)$$
$$= I(T) - d(P) + 2n_0 - d(P) + 2d(L)$$
$$= I(T) + 2n_0 - 2d(P) + 2d(P) + 2$$
$$= I(T) + 2(n_0 + 1).$$