1. For each of the following program fragments, give an analysis of the (worst case) running time in Big-O notation. (a).

```
sum = 0;
for ( i = 0; i < n; ++i )
    ++sum;
```

Proof. For each element, we perform one operation. Trivially, we do n work, thus this program fragment runs in $O(n)$. □

(b).

```
sum = 0;
for ( i = 0; i < n; ++i )
    for ( j = 0; j < n; ++j )
        ++sum;
```

Proof. For each element, we perform n operations. Thus, we perform $n \cdot n$ operations total, thus this program fragment runs in $O(n^2)$. □

(c).

```
sum = 0;
for ( i = 0; i < n; ++i )
    for ( j = 0; j < n * n; ++j )
        ++sum;
```

Proof. For each element, we will perform n^2 operations. Thus, we perform in total $n \cdot n^2$ operation. Thus, this program fragment runs in $O(n^3)$. □

(d).

```
sum = 0;
for ( i = 0; i < n; ++i )
    for ( j = 0; j < i; ++j )
        ++sum;
```

Proof. For each $i = 0 \cdots n-1$, we perform i work. In other words, we perform $0 + 1 + 2 + \cdots + n - 1$ work. The amount of work we perform is equivalent to

$$\begin{aligned}
\sum_{i=0}^{n-1} i &= \frac{(n-1)((n-1)+1)}{2} \\
&= \frac{n(n-1)}{2} \\
&= \frac{n^2 - n}{2}
\end{aligned}$$

Thus, our program operates in $O(n^2)$. □

(e).

```
sum = 0;
for ( i = 0; i < n; ++i )
    for ( j = 0; j < i * i; ++j )
        for ( k = 0; k < j; ++k )
            ++sum;
```

Proof. For each $i = 0 \cdots n-1$, we perform i^2 work, and for each $j = 0 \cdots i^2 - 1$ we do j work.

$$\sum_{i=0}^{n-1} \sum_{j=0}^{i^2-1} j \tag{1}$$

We observe that our most inner loop performs $0 + 1 + \cdots + i^2 - 1$ work, which equals

$$\sum_{j=0}^{i^2-1} j = \frac{(i^2-1)((i^2-1)+1)}{2}$$

and our double summation (1) becomes

$$\begin{aligned}\sum_{i=0}^{n-1} \frac{(i^2-1)((i^2-1)+1)}{2} &= \sum_{i=0}^{n-1} \frac{(i^2-1)(i^2)}{2} \\ &= \sum_{i=0}^{n-1} \frac{i^4-i^2}{2}\end{aligned}$$

Theorem: Faulhaber's Formula: For $n, k, a \in \mathbb{N}$,

$$\sum_{k=1}^{n} k^a = \frac{1}{a+1} \sum_{j=0}^{a} (-1)^j \binom{a+1}{j} B_j n^{a+1-j} \tag{2}$$

where B_j denotes the j^{th} Bernoulli number. Using (2), we apply the following identities.

$$\sum_{i=0}^{n-1} \frac{i^4-i^2}{2} = \frac{1}{2}(\sum_{i=0}^{n-1} i^4 - \sum_{i=0}^{n-1} i^2)$$

$$= \frac{1}{2}\left(\frac{(n-1)((n-1)+1)(2(n-1)+1)(3(n-1)^2+3(n-1)-1)}{30} - \frac{(n-1)(n-1)+1)(2(n-1)+1)}{6}\right)$$

$$\vdots$$

$$= \frac{n(n-1)(2n-1)(n-2)(n+1)}{20}$$

$$= \frac{2n^5 - 5n^4 + 5n^2 - 2n}{20}$$

Thus, our time complexity is $O(n^5)$.

□

(f).

```
sum = 0;
    for ( i = 1; i < n; ++i )
        for ( j = 1; j < i * i; ++j )
            if ( j % i == 0 )
            for ( k = 0; k < j; ++k )
                ++sum;
```

Proof. Let us thoroughly examine our program's execution steps. Suppose $n = 5$.

```
i = 1
    j = 1
i = 2
    j = 1, 2, 3
i = 3
    j = 1, 2, 3, 4, 5, 6, 7, 8
i = 4
    j = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
```

For $i = 1$, trivially, our program does no work, as the second loop does not execute. For $i = 2$, our program's second loop executes $i^2 - 1$ times, or 3 times. We observe that the condition $j \mod i = 0$ is satisfied once when j = 2, thus, our program does 2 work. For $i = 3$, the condition $j \mod i = 0$ is satisfied twice when $j = 3, 6$, and our program does $3 + 6 = 9$ work. For $i = 4$, the condition $j \mod i = 0$ is satisfied three times when $j = 4, 8, 12$, and our program does 24 work.

It follows that for $i = 5$, our program will do $5 + 10 + 15 + 20 = 50$ work. For $i = 6$, our program will do $6 + 12 + 18 + 24 + 30 = 90$ work.

For each successive i, $j \mod i = 0$ is satisfied $i - 1$ times. This is intuitive, as between 1 and $i^2 - 1$, there are $i - 1$ multiples of i.

Let's generalize our solution. We observe that for each i, $j \mod i = 0$ is satisfied $i - 1$ times.

$$\begin{aligned}
i + 2i + 3i + \cdots + (i-1) \cdot i &= i \cdot (1 + 2 + 3 + \cdots + (i-1)) \\
&= i \cdot \left(\sum_{j=1}^{i-1} j \right) \\
&= i \cdot \left(\frac{(i-1)((i-1)+1)}{2} \right) \\
&= i \cdot \left(\frac{i^2 - i}{2} \right) \\
&= \frac{i^3 - i^2}{2}
\end{aligned}$$

We have then, the general solution to our program's time complexity equal to

$$\sum_{i=1}^{n-1} \frac{i^3 - i^2}{2} \tag{3}$$

Using (2), we can decompose (3) into

$$\begin{aligned}
\sum_{i=1}^{n-1} \frac{i^3 - i^2}{2} &= \frac{1}{2} \cdot \left(\sum_{i=1}^{n-1} i^3 - \sum_{i=1}^{n-1} i^2 \right) \\
&= \frac{1}{2} \left(\frac{(n-1)^2((n-1)+1)^2}{4} - \frac{(n-1)((n-1)+1)(2(n-1)+1)}{6} \right) \\
&\vdots \\
&= \frac{n(n-1)(3n-1)(n-2)}{24} \\
&= \frac{3n^4 - 10n^3 + 9n^2 - 2n}{24}
\end{aligned}$$

Thus, our program fragment runs in $O(n^4)$ $\square$

2. (1). Order the following 14 functions by growth rate: N, $\sqrt{N}$, $N^{1.5}$, N^2, $NlogN$, $NloglogN$, $Nlog^2N$, $Nlog(N^2)$, $\frac{2}{N}$, 2^N,$2^{\frac{2}{N}}$, 29, N^2logN, N^3.

From smallest growth rate to largest growth rate, we have:

$\frac{2}{N} < 29 < \sqrt{N} < N < NloglogN < NlogN \leq Nlog(N^2) < Nlog^2N < N^{1.5} < N^2 < N^2logN < N^3 < 2^{\frac{N}{2}} < 2^N$

(2). Which two of these functions grow at the same rate?

$Nlog(N^2) = 2Nlog(N)$. This is asymptotically equal to $NlogN$.

3. Suppose $T_1(n) = O(f(n))$ and $T_2(n) = O(f(n))$, prove that $T_1(n) + T_2(n) = O(f(n))$

Proof. For $T_1(n) = O(f(n))$, $T_2(n) = O(f(n))$, there exists c_1, n_1, c_2, $n_2 \in \mathbb{N}$ such that

$$T_1(n) \leq c_1 \cdot f(n) \tag{4}$$

$$T_2(n) \leq c_2 \cdot f(n) \tag{5}$$

for all $n \geq n_1$, $n \geq n_2$ respectively. Adding (4), (5), we have

$$\begin{aligned} T_1(n) + T_2(n) &\leq c_1 \cdot f(n) + c_2 \cdot f(n) \\ &\leq f(n)(c_1 + c_2) \end{aligned}$$

Let $c_3 = c_1 + c_2$, and $n_3 = \max(n_1, n_2)$, then

$$T_1(n) + T_2(n) \leq c_3 \cdot f(n) \tag{6}$$

for all $n \geq n_3$. This is precisely equivalent to saying $T_1(n) + T_2(n) = O(f(n))$.

□