Charles Kolozsvary Period 2 Algorithms

What is the value of the variable "count", as a function of "n", after running the following code fragment?

What is the order of growth? Justify your answer. Formal proof is not necessary. You can be as creative as you want to "show" your conclusion.

```
int count = 0;
for (int i = 0; i < n; i++)
for (int j = i+1; j < n; j++)
for (int k = j+1; k < n; k++)
count++;
```

Answers:

The value of the variable "count" after running the code fragment is:

$$count = \frac{N(N-1)(N-2)}{6}$$

$$\text{If N = 11, count} = \frac{11(10)(9)}{6} = 165$$

The order of growth for this code fragment is N^3 since the order of growth is directly proportional to the instances of the most frequent operation (count++;).

Yet the order of growth is not $\frac{N(N-1)(N-2)}{6}$ verbatim. This is the case because order of growth models use tilde notation, where all coefficients and lower order terms are removed.

If we expand our expression for the value of variable count for a given N we have the number of count operations $= N^3/6 - N^2/2 + N/2$.

When we remove all the lower order terms we are left with $N^3/6$, and when we remove coefficients all that remains is N^3.

What about all the other operations that occur in this code snippet? (I hear you asking)

Well, every other operation present like variable declaration and (int count, int i, int j, etc.) assignment statements (count = 0, i = 0, j = 0, etc.) happen much less frequently than the number of increment operations (count++).

If we declare a certain function f(N) such that f(N) describes the time complexity of the code snippet including the frequency of every operation, and we declare a function g(N) which is only described by the most frequent operation (count++;) we find that the $\lim_{N\to\infty} \frac{f(N)}{g(N)} = 1$ which tells us that what really matters for order of growth is the most frequent operation which is count++, and the order of growth is therefore N^3.

Also the order of growth is most concerned with the GENERAL behavior of the time complexity as a function of N not the actual specific amount of time required for each individual run time

analysis. Regardless of what N is in this code snippet the rate at which the time complexity will increase in relation to N will always resemble the proportionality found in N vs N^3. Even if the function of time complexity versus N is nearly $N^3/6$ in this case, the order of growth doesn't care, it cares about GENERAL behaviour which is N^3.

For all of these reasons, the order of growth for this code snippet is N^3.