|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N choose k | Method 1 | Method 2 | Method 3 | Method 4 | Method 5 |
| 5 choose 3 | 1420600 | 460000 | 447700 | 23600 | 11500 |
| 10 choose 5 | 1579200 | 479300 | 442800 | 34300 | 12200 |
| 15 choose 8 | 3176600 | 1428100 | 1108200 | 34400 | 16200 |
| 20 choose 10 | 2794500 | 1184100 | 3330800 | 42200 | 26400 |
| 50 choose 25 | 2857700 | 1208000 | Died here | 391400 | 227300 |
| 80 choose 40 | Died here | 4536000 | // | 882800 | 2165900 |
| 100 choose 50 | // | 3217200 | // | 478100 | 305000 |
| 150 choose 75 | // | Died here | // | 3116700 | 5667700 |
| 200 choose 100 | // | // | // | 1229900 | 1340700 |
| 300 choose 150 | // | // | // | 4139000 | 2586100 |
| 500 choose 250 | // | // | // | 11648500 | 11292700 |
| 1000 / 500 | // | // | // | 14304400 | 13142000 |
| 2000 / 1000 | // | // | // | 39350000 | 23090600 |
| 5000/2500 | // | // | // | 323432100 | 123955200 |
| 10000 / 5000 | // | // | // | overflow | overflow |

Mark Meade \*all time In nanoseconds

They are both 100% accurate up to their overflow limit. They are also both very efficient when compared to the first 3 methods. The calculations aren’t incorrect due to overflow limits, but the code errors out. Id prefer the last method (method 5) because it seems to be the most efficient when calculating binomial coefficients. The only drawback that applies to these methods is that they are bounded by Java’s array[][] function. Once that gets too large, the program can’t function and errors out. In comparison to the other 3 methods however, these functions are more efficient, and are able to work with larger inputs.