| PRINCE SULTAN UNIVERSITY |
| --- |
| Project-Prime-Generator-241 |
| Course CS285 |
|  |
| **Abdulrhman Mamdouh**  **Abdulmlik Al-Shanqiti** |
| **11/30/2024** |

|  |
| --- |

**Table of Contents**

[1. The Prime Numbers Formulas 2](#_gjdgxs)

[2. .Methods explanation 2](#_30j0zll)

[3. Results 5](#_1fob9te)

[4. References 5](#_2et92p0)

# The Prime Numbers Formulas

* **Formula 1:**
  + Expression:
  + 4⋅(n5−133n4+6729n3−158379n2+1720294n−6823316)4⋅(n5−133n4+6729n3−158379n2+1720294n−6823316)
  + Explanation:
  + This polynomial expression is designed to generate numbers that may include prime values for specific positive integers nn. The polynomial has coefficients tuned to ensure that outputs avoid trivial divisors for many nn. However, as with all such formulas, not every output will necessarily be prime, and the reliability diminishes as nn grows.
* **Formula 2:**
  + Expression:
  + 36⋅(n6−126n5+6217n4−153066n3+1987786n2−13055316n+34747236)36⋅(n6−126n5+6217n4−153066n3+1987786n2−13055316n+34747236)
  + Explanation:
  + This is a degree-6 polynomial multiplied by 36. Similar to Formula 2, it generates large numbers that could potentially be prime for specific values of nn. The high degree and carefully chosen coefficients help to structure outputs around prime-friendly values. However, prime verification is still necessary.
* **Formula 3:**
  + Expression:
  + n4−97n3+3294n2−45458n+213589n4−97n3+3294n2−45458n+213589
  + Explanation:
  + This degree-4 polynomial is simpler compared to the previous two and is not multiplied by a constant. The coefficients are fine-tuned to minimize the likelihood of divisors for small nn. While it might occasionally yield primes, it does not guarantee primes for all nn.
* **Formula 4:**
  + Expression:
  + n5−99n4+3588n3−56822n2+348272n−286397n5−99n4+3588n3−56822n2+348272n−286397
  + Explanation:
  + This is a fifth-degree polynomial. The careful arrangement of coefficients aims to produce outputs that align with prime conditions for smaller values of nn. However, like the others, it generates composite numbers as well for many inputs.
* **Formula 5:**
  + Expression:
  + −66n3+3845n2−60897n+251831−66n3+3845n2−60897n+251831
  + Explanation:
  + A cubic polynomial whose coefficients are designed to tune the output toward values that can be prime for specific nn. The negative leading coefficient causes the polynomial to decrease for larger nn, which might shift its prime-generating tendencies.
* **Formula 6:**
  + Expression:
  + 36n2−810n+275336n2−810n+2753
  + Explanation:
  + A quadratic polynomial designed to yield numbers that may be prime for certain nn. The choice of coefficients, such as 36 and 2753, reflects an attempt to align outputs with prime numbers. However, it is relatively simplistic and will frequently produce composites.
* **Formula 7:**
  + Expression:
  + 3n3−183n2+3318n−187573n3−183n2+3318n−18757
  + Explanation:
  + Another cubic polynomial where coefficients have been selected to bias the output toward prime numbers for certain nn. The cubic form suggests that outputs will grow quickly as nn increases, making direct prime verification essential.
* **Formula 8:**
  + Expression:
  + 47n2−1701n+1018147n2−1701n+10181
  + Explanation:
  + A second-degree polynomial where coefficients are tuned to minimize common divisors of outputs. This one likely has better success at generating primes for smaller nn, but it is not a guarantee of primality.
* **Formula 9:**
  + Expression:
  + 103n2−4707n+50383103n2−4707n+50383
  + Explanation:
  + Another quadratic polynomial, similar in structure to Formula 9 but with larger coefficients. This one is tuned to generate larger numbers, potentially prime, for specific nn. The choice of 103 as the leading coefficient suggests careful tuning for prime congruences.
* **Formula 10:**
  + Expression:
  + n2−n+7
  + Explanation:
  + The formula n2−n+41n2−n+41 generates prime numbers efficiently for small nn due to the careful selection of coefficients and the rapid growth of quadratic terms. However, it is not an absolute prime generator and requires verification for outputs when nn becomes large. Despite its limitations, this formula remains a classic example of mathematical ingenuity in the study of primes.

# .Methods explanation



1. **main Method**

* This is the entry point of the program.
* Prompts the user to choose between two options:
  + **Option 1:** View statistics about the functions (like most primes, largest prime, etc.).
  + **Option 2:** View detailed analysis of a specific function.
* Handles input validation and loops for specific function testing.

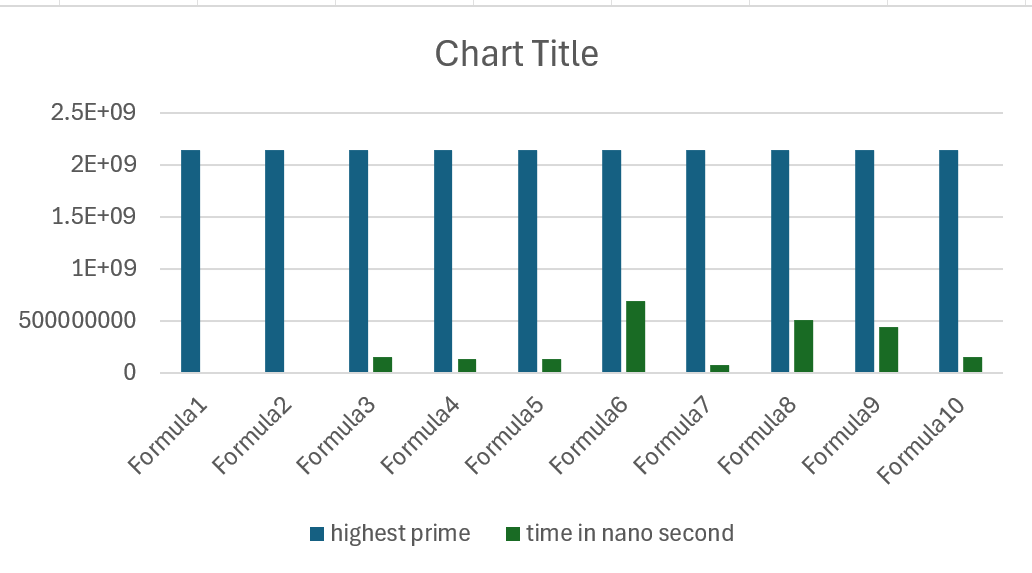
1. **firstNonPrimeGenerator Method**
   * Identifies the first formula (function) that generates a non-prime number.
   * Iterates over all functions (indexed from 0 to 9) and tests them with values of n from 0 to 9999.
   * As soon as a non-prime value is found, it prints the function, the input n, and the non-prime value.
2. **TestFunc Method**
   * Analyzes a specific function chosen by the user.
   * Tests the function with values of n from 0 to 9999.
   * Outputs:
     + The highest prime number generated by the function.
     + The total count of prime numbers.
     + The time taken to perform the calculations (in nanoseconds).
3. **checkprime Method**
   * Determines if a given number (x) is prime.
   * Uses a square root optimization to reduce the number of divisions needed.
   * Returns true for prime numbers and false for non-prime numbers.
4. **functions Method**
   * Defines 11 polynomial functions indexed from 0 to 10.
   * Takes two inputs:
     + input: Specifies which function to evaluate.
     + n: The input variable for the function.
   * Returns the computed value of the chosen function for a given n.
5. **getFunctionName Method**
   * Provides the mathematical formula (as a string) for a given function index.
   * Used to display user-friendly information about the selected function.
6. **mostPrime Method**
   * Identifies the function that generates the largest number of prime numbers for values of n between 0 and 9999.
   * Tracks the function and its prime count and outputs the result.
7. **LargestPrime Method**
   * Finds the largest prime number generated by any of the functions for values of n between 0 and 9999.
   * Outputs the function and the largest prime value.
8. **fastest Method**
   * Measures the execution time for each function to test prime numbers for values of n from 0 to 9999.
   * Identifies and outputs the function that completes its execution in the least time.

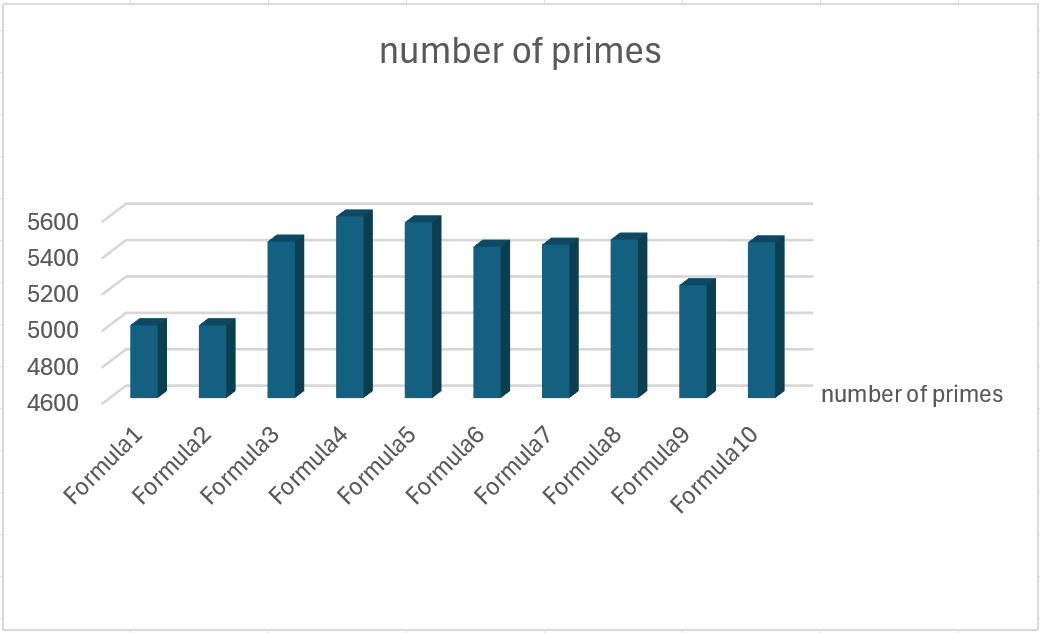
# Results

Write the results provided for all the 10000 numbers. Make sure to add the graphs, charts and table that summarizes the results (Time, first none prime,..etc)

|  | Formula1 | Formula2 | Formula3 | Formula4 | Formula5 | Formula6 | Formula7 | Formula8 | Formula9 | Formula |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| highest prime | 2147387920 | 2147387920 | 2147175965 | 2146369117 | 2145991105 | 2147069429 | 2147157051 | 2147085767 | 2146578299 | 2145596871 |
| number of primes | 5000 | 5000 | 5460 | 5598 | 5566 | 5432 | 5443 | 5471 | 5220 | 5456 |
| time | 213000 | 1816600 | 150975600 | 137857700 | 129867400 | 693276200 | 77098400 | 511119100 | 442441000 | 148982700 |
| Largest number of primes |  |  |  | TRUE |  |  |  |  |  |  |
| Largest prime | TRUE |  |  |  |  |  |  |  |  |  |
| least time |  | TRUE |  |  |  |  |  |  |  |  |
| first non-prime number | TRUE |  |  |  |  |  |  |  |  |  |

# 





# References

<https://mathworld.wolfram.com/Prime-GeneratingPolynomial.html>

<https://byjus.com/maths/prime-numbers/>

<https://www.cuemath.com/numbers/prime-numbers/>