PROJEKT 2 – PIKULA KAMIL, grupa 76

Liczby pierwsze

Liczby pierwsze wbrew pozorom są ciężkim orzechem do zgryzienia dla komputerów. Przy dużych wartościach nawet najprostszy algorytm nie działa zbyt szybko, czas oczekiwania na wynik końcowy (sprawdzający czy liczba jest pierwsza) się wydłuża, dlatego też moim zadaniem była implementacja algorytmu oraz zaproponowanej nowej wersji (bez zmian trzymających trzon projektu) tak, aby sprawdzanie liczby było bardziej efektywne.

Podstawowy kod przed instrumentacją:

```
bool IsPrime(BigInteger Num)
                {
                    if (Num < 2) return false;
                    else if (Num < 4) return true;
                    else if (Num % 2 == 0) return false;
                    else for (BigInteger u = 3; u < Num / 2; u += 2)
                            if (Num % u == 0) return false;
                    return true;
                }
```

Poprawiony/efektywniejszy kod przed instrumentacją:

```
bool IsPrimeFaster(BigInteger Num)
                {
                    if (Num < 2) return false;
                    else if (Num < 4) return true;
                    else if (Num % 2 == 0) return false;
                    else for (BigInteger u = 3; u * u < Num; u += 2)
                            if (Num % u == 0) return false;
                    return true;
                }
```

Kod źródłowy po instrumentacji znajduje się na moim repozytorium na Github – https://github.com/kamilpikula/prime-numbers-algorithm

Liczby (na których wykonywałem badania) oraz wyniki liczników zapętleń znajdują się w tabeli poniżej:

lp	**number**	**IsPrimeCounter**	**IsPrimeFasterCounter**
1	100913	25228	159
2	1009139	252284	502
3	10091401	2522850	1588
4	100914061	25228515	5023
5	1009140611	252285152	15883
6	10091406133	2522851533	50228
7	100914061337	25228515334	158835
8	1009140613399	252285153347	502280

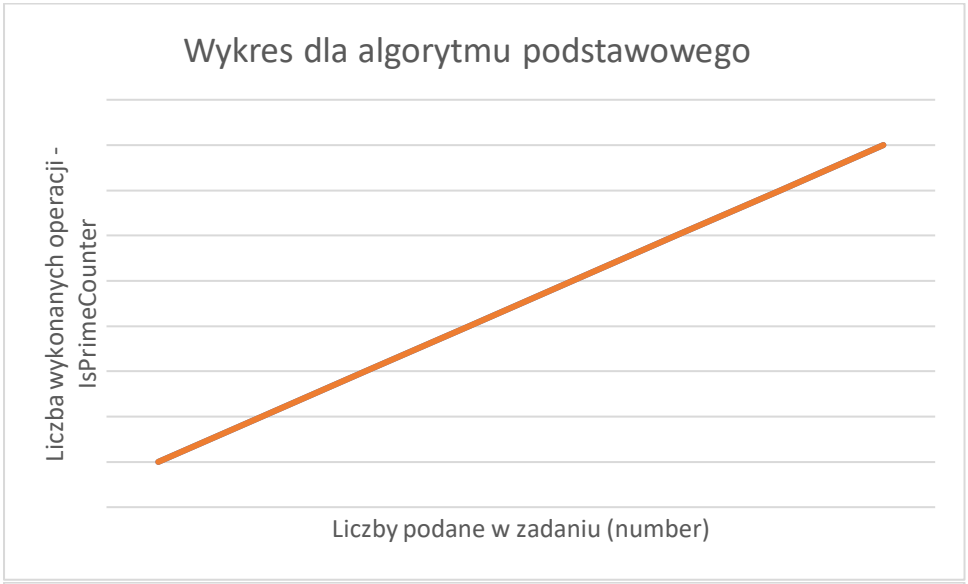
PS. Ostatnią liczbę wyliczyłem ze wzoru podanego na zajęciach (zaoszczędziłem sobie czasu oraz szkoda było mi sprzętu – w pracy myśleli, że kopię bitcoiny kiedy dzieliłem siódmą liczbę : -))

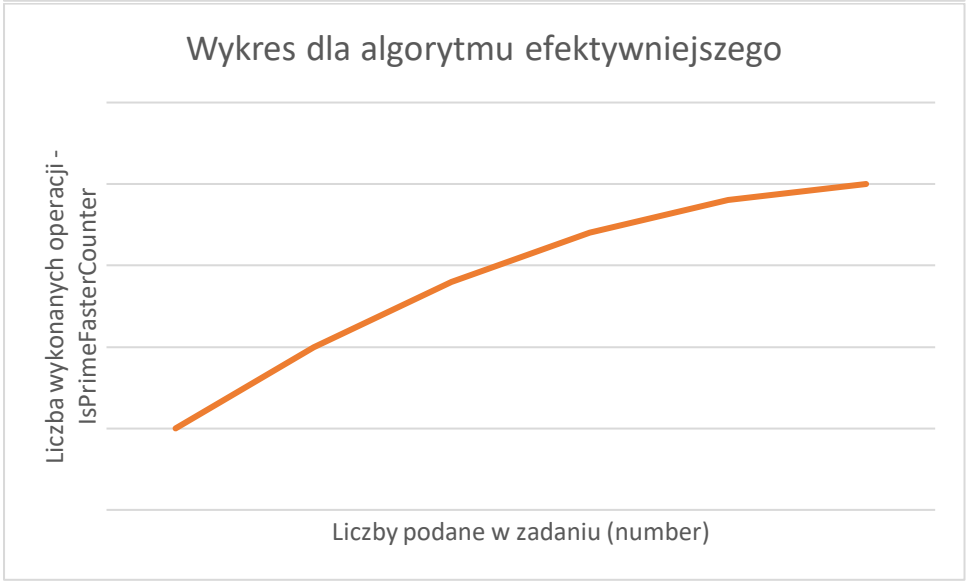
Wzór, który zastosowałem to:

1	wartość A	wartość B
2	wartość C	X

wartość B * wartość C / wartość A = X, w tym przypadku X była ósma liczba „IsPrimeCounter”, A – siódma liczba „number”, B – siódma liczba „isPrimeCounter”, a C – ósma liczba „number”. Jako, że algorytm *IsPrimeFaster* jest o wiele efektywniejszy – przy nim nie musiałem zastosowywać wzoru, gdyż bardzo szybko pokazał wartość dla ósmej wartości „IsPrimeFasterCounter”.

Wykresy:





Opis wykresów:
Niestety, mistrzem Excela nie jestem i wykonałem bez danych ze względu na to, że wykresy, które tworzyłem, źle mi się formatowały (z braku czasu nie zdążyłem poustawiać odpowiednio osi X i Y, dlatego też błędnie się wyświetlały). Z tego też względu podałem powyższe wykresy. Są one oczywiście prawidłowe. Pierwszy wykres jest funkcją liniową, drugi zaś – funkcją logarytmiczną. Jest to **notacja duże O (pierwiastek z n).**

Wnioski (ocena złożoności):
Algorytm podstawowy pod względem złożoności jest bardzo słabo – przy coraz większych wartościach liczb liczba iteracji wzrasta liniowo. Algorytm efektywniejszy jest o wiele bardziej zoptymalizowany, gdyż liczba operacji wzrasta logarytmicznie (przez co jest bardziej

optymalny). Liczba operacji w funkcji logarytmicznej nie wzrasta aż tak bardzo dynamicznie jak w funkcji liniowej.