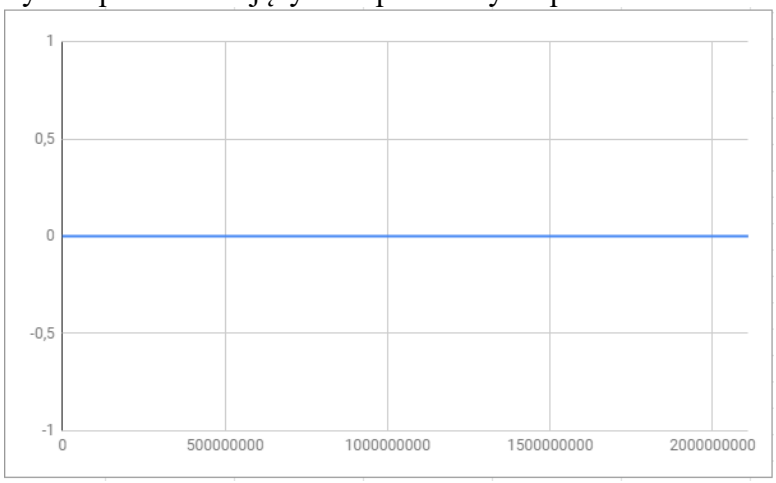
Piotr Waszak K37

W projekcie numer 1, mamy za zadanie przeprowadzić badanie złożności wyszukiwania liniowego i binarnego liczb całkowitych. Obie analizy zostaną poddane instrumentacji, oraz określimy ich złożoność na pesymistyczną i średnią.

Wyszukiwanie Liniowe

(1)Jako pierwszą przeprowadze analize wyszukiwania liniowego przed instrumentacją, w badaniu pesymistycznym. Oczywiście kod źródłowy znajduję się na githubie jako Program.cs. Pisanie kodu rozpocząłem od utworzenia tablicy na określoną długość mniejszą od liczby 2^{28} (na początku zamierzałem zrobić długosc za pomocą metody „Math.Pow(2,28)", jednak niestety trwałoby to zbyt długo, dlatego zastosowałem krótszą tablice)
Przypisałem więc losowa liczbe do każdego elemntu tablicy. Liczby posortowałem.
Następnie stworzyłem funkcje, która ma zadanie sprawdzić pokolei czy gdzieś wśród wylosowanych liczb, znajduję się taka którą szukam: „ `isPresent(tab, 202313)`". Czas pomiędzy przeszukiwaniem tablic odmierzałem metodą stopwatch(). Na końcu, aby zapisać rezultaty użyłem funkcji „`StreamWriter sw`" , która zapisuje numer tablicy, wartość tablicy, oraz czas w jakim została ona porównana z liczbą wyszukiwaną.
Wykres przedstawiający czas potrzebny na porównanie każdej następnej liczby



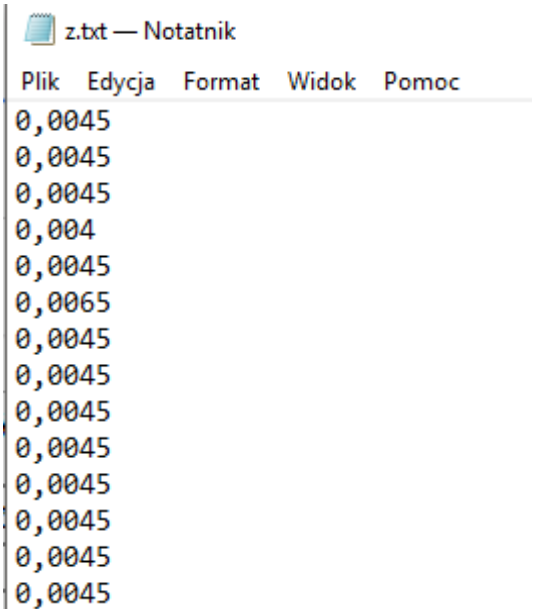
Poniżej 3 wyrwkowych wyników:

1002	2796861	0

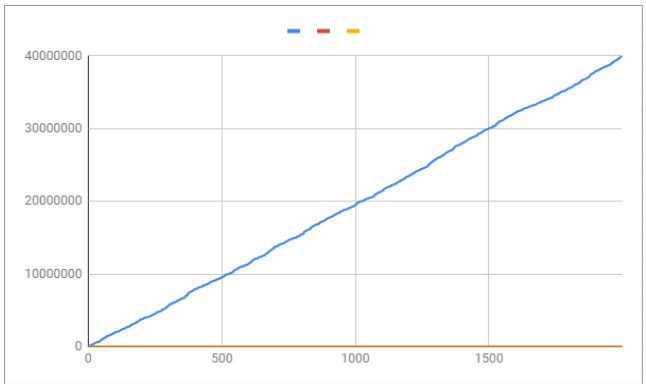
1416	1271796658	0

1843	1960140343	0

(2)W drugim kodzie dokonałem kilku zmian z tego względu, że w pierwszych rekordach pojawiały mi się same zera. Na początku żeby tego uniknąć dodałem minimalną i maksymalną wartość jaka może być wylosowana, niestety jednak to nie pomogło, więc stworzyłem pętle foreach, w której mogłem przedstawić pełen zakres liczb losowych, oraz dokonać pomiarów czasu.
Po doknaniu pomiaru dla każdego sprawdzenia, czy liczba jest zgodna z wyszukiwana, zrobiłem pomiar średniej za pomocą `double ElapsedSeconds = ElapsedTime * (1.0 / tab.Length); .`
Kod na githubie jest pod nazwa wyszukiwanielinavg.cs



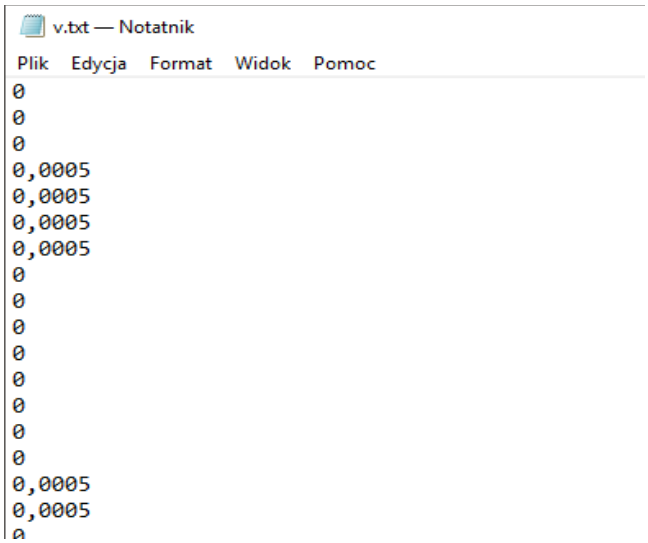
(3)Trzeci kod lininstrumentacja.cs, dokonujemy sprawdzenia ile liczb jest przyrównanych do naszej liczby podczas wyszukiwania. Dodałem więc w naszej metodzie licznik: „OpIncrement”.
Dodatkowo został zmierzony czas każdego przyrównania.Wykres przedstawiający wzrost naszego licznika na każdej następnej tablicy.



Przykładowe rekordy.

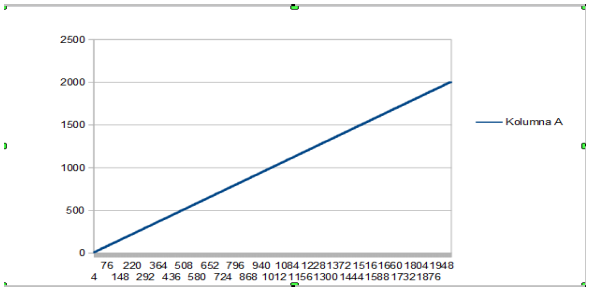
4	110403	5	0
571	11013146	572	0
994	19338240	995	0
1671	33262153	1672	0

(4)Ostatni kod to wyszukiwanie liniowe z instrumentacja i jej średnia pod nazwa lininstrumentacjaavg.cs na githubie. W zasadzie to do poprzedniego kodu dodałem tylko liczenie średniej, oraz odmierzałem czas mojego licznika.



Wyszukiwanie Binarne

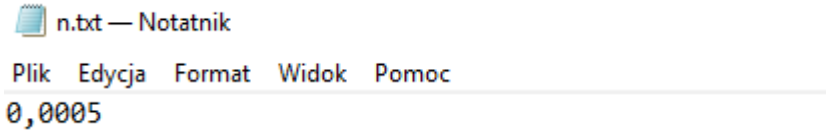
(1) Zmiany kodu w stosunku do wyszukiwania zaszły przede wszystkim w funkcji IsPresent. Funkcja ma za zadanie podzielić tablice na pół, a następnie jeśli wyszukiwana liczba jest większa od środkowej wartości tablicy iść w prawo, jeśli natomiast wartość jest mniejsza ma za zadanie deinkrementować punkty tablicy. Plik na githubie: binarne.cs .Wykres poniżej.



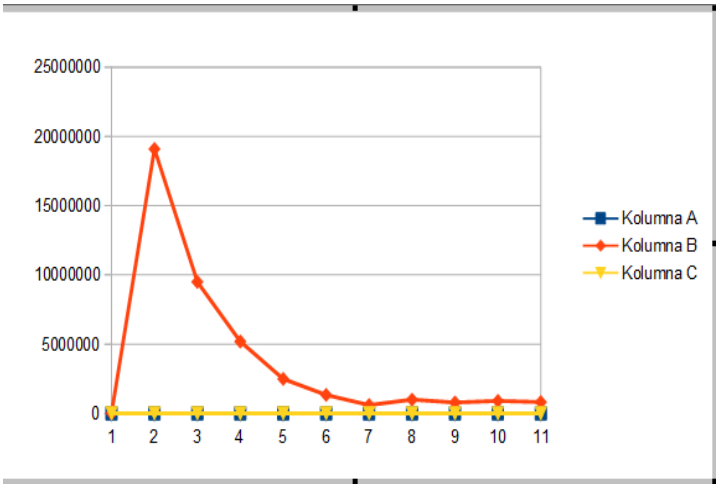
Przykładowe rekordy:

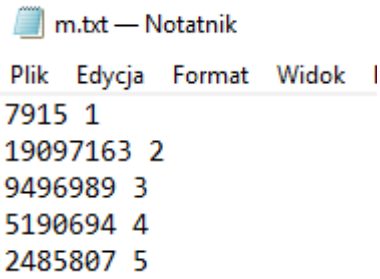
291	6154216	-1
1318	26426710	-1
1701	34276948	-1

(2) Wyszukiwanie binarne ze średnią. Plik z kodem znajduje się na githubie pod nazwą binarneavg.cs . Został zmierzony średni czas sprawdzenia wyszukiwań.



(3) Wyszukiwanie binarne z instrumentacja, kod znajduje się na github o nazwie binarneinstrumentacja.cs. W pętli while wstawiłem licznik couter. Moja wyszukiwana liczba była mniejsza od `srodek = (lewo + prawo) / 2;` więc warunek szedł w lewą strone.





(4) Wyszukiwanie binarne z instrumentacja średnia. Kod znajduje się na githubie o nazwie binarneinstrumentacjaavg.cs. Została policzona średnia wyszukiwań z instrumentacja.

Badanie miało na celu pokazać szybkość czasową, oraz użycia zaosobów doo wyszukiwania logicznego. Wyszukiwanie binarne przy większych liczbach jest szybsze, od wyszukiwania liniowego, ponieważ zaczyna sprawdzać połowe rekordów, w przeciwieństwie do liniowego, które musi sprawdzić wszystkie rekordy.