Metody probabilistyczne i statystyka
Zadanie domowe 3
Illia Azler 239537

Zadanie 1:

Celem zadania było rozszerzenie symulacji z poprzedniej listy o eksperymentalne wyznaczanie maksymalnej liczby kul w urnie po wrzuceniu n kul do n urn (maximum load) w przypadku, gdy:

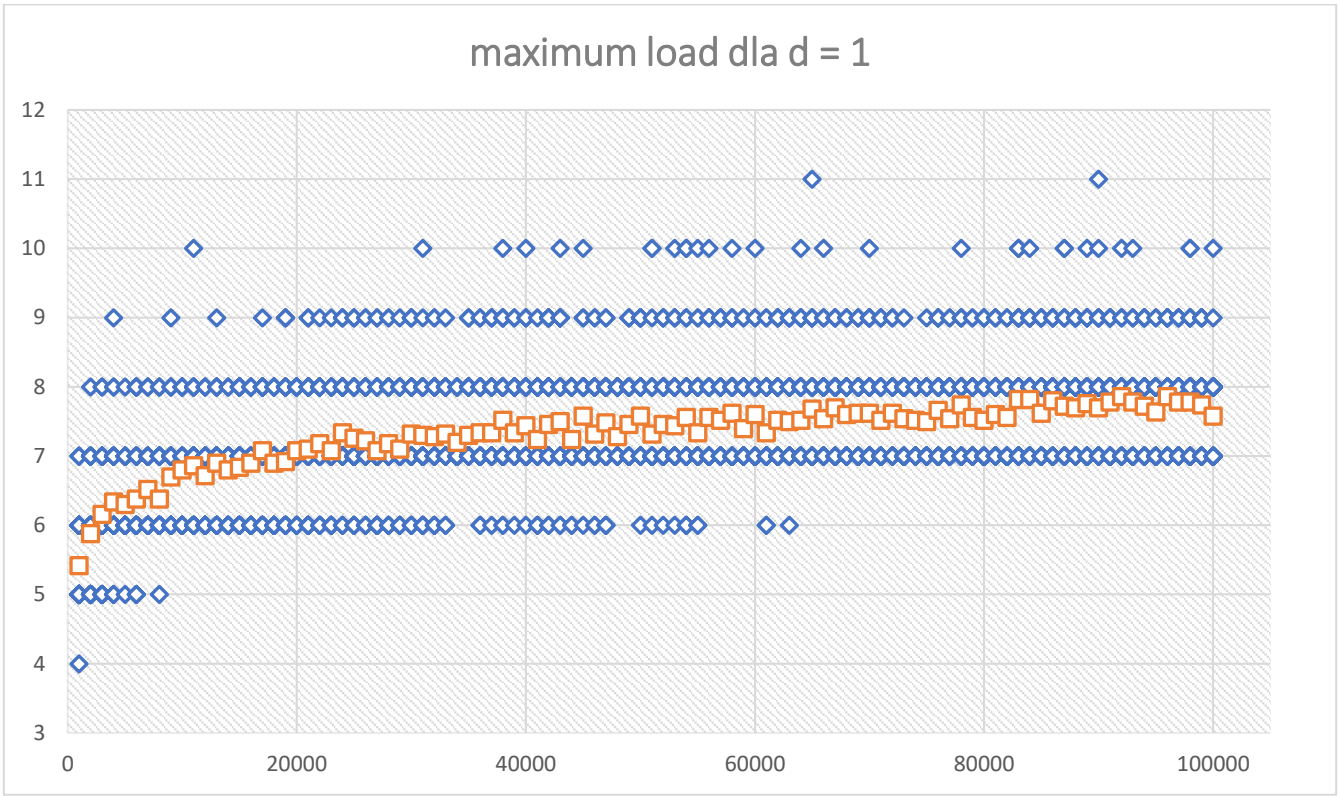
a) dla każdej kuli wybieramy niezależnie i jednostajnie losowo jedną z n urn, w której umieszczamy kulę;

b) dla każdej kuli wybieramy niezależnie i jednostajnie losowo z powtórzeniami d urn i umieszczamy kulę w najmniej zapełnionej z wybranych urn („remisy" rozstrzygamy w dowolny sposób).

Przeprowadzone zostały podobne eksperymenty jak na poprzedniej liście celem wyznaczenia $L_n^{(d)}$ - maksymalnego zapełnienia pojedynczej urny po wrzuceniu n kul – dla d = 1 (punkt zadania a)) oraz d = 2 (punkt zadania b)).

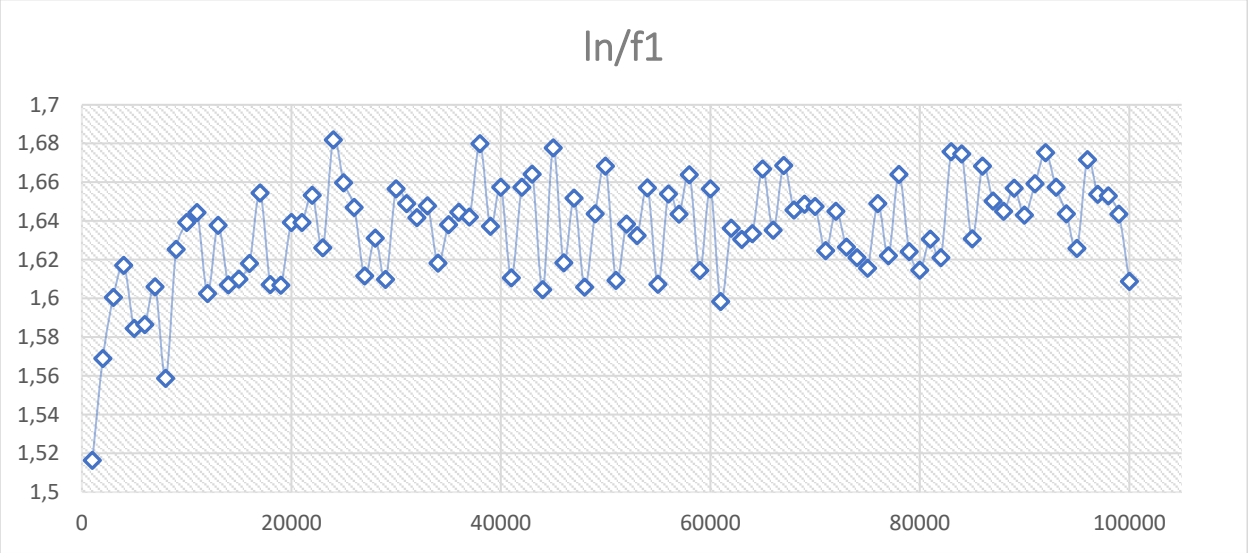
Implementacja programu w pliku **ppd_z3_1_kod_239537.py**

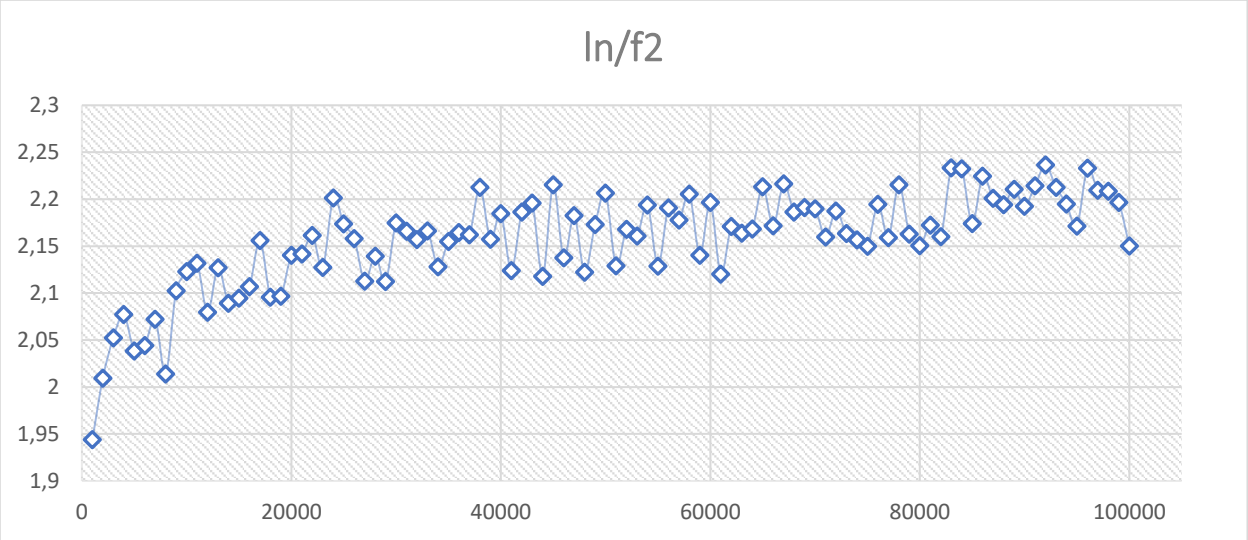
Test 1(a):



Wniosek: Im więcej kul wrzucimy do urny (oznaczane jako n), tym bardziej rośnie prawdopodobieństwo, że osiągnie ona maksymalną liczbę kul, przy czym ten wzrost jest bardzo szybki. To zjawisko wynika z tego, że szanse na to, aby kolejna kulka była maksymalną, maleją w miarę zwiększania liczby wrzutów.

Asymptotyka:





$$Ln(n) = O(e^n)$$

Zadanie 2:

Celem zadania była implementacja oraz testowanie działania algorytmu sortowania przez wstawianie INSERTIONSORT. W tym celu dla każdego n ∈ {100, 200, . . . , 10 000} wykonano po k = 50 niezależnych powtórzeń:

(a) generowania tablicy A[1..n] będącej losową permutacją zbioru liczb {1, . . . , n};

(b) sortowania wygenerowanej tablicy A.

(c) zapisywania statystyk obejmujących rozmiar danych n, liczbę wykonanych porównań między kluczami (elementami tablicy A) oraz liczbę przestawień kluczy.

Wykorzystałem pseudokod dla implementacji programu:

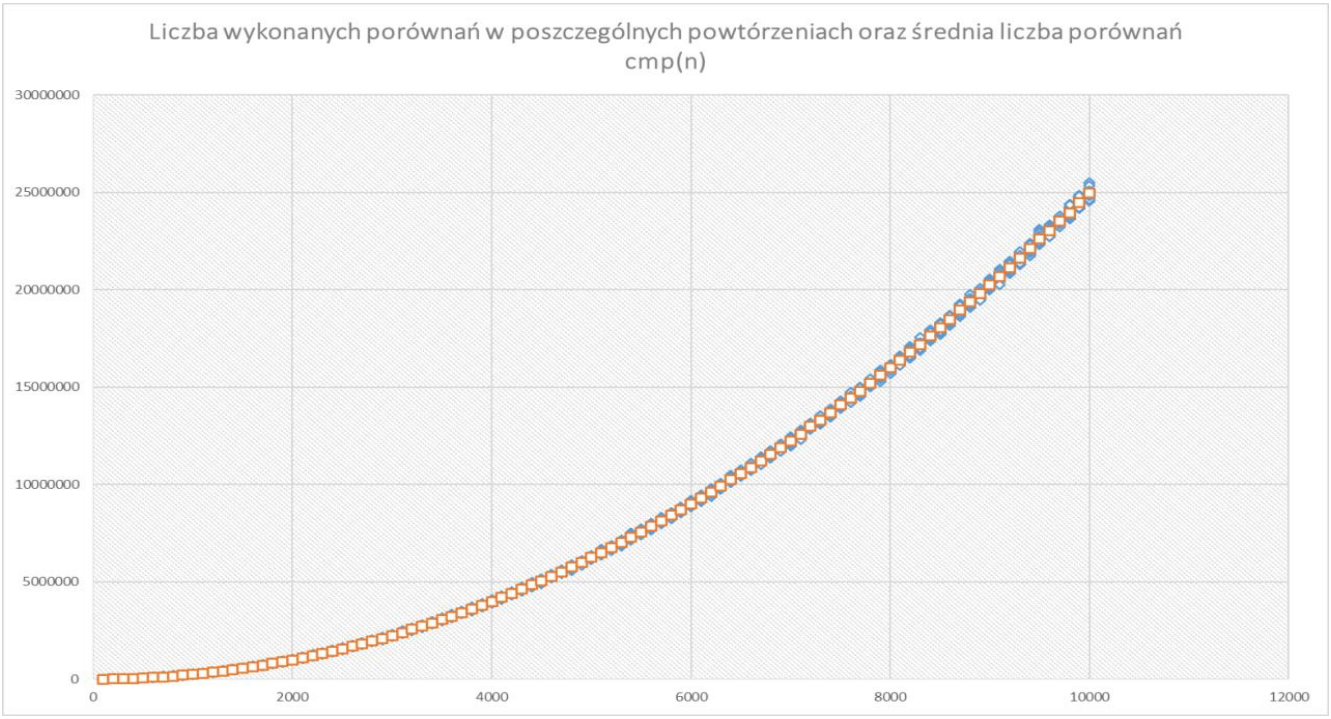
Algorytm 1 Algorytm sortowania przez wstawianie (por. rozdział 2.1 w [CLRS09]).

```
1:  procedure INSERTIONSORT(A[1..n])
2:      for j = 2 to n do
3:          key ← A[j]
4:          // Wstaw A[j] w posortowany podciąg A[1..j − 1].
5:          i ← j − 1
6:          while i > 0 and A[i] > key do
7:              A[i + 1] ← A[i]
8:              i ← i − 1
9:          end while
10:         A[i + 1] ← key
11:     end for
12: end procedure
```

Implementacja programu w pliku ppd_z3_2_kod_239537.py

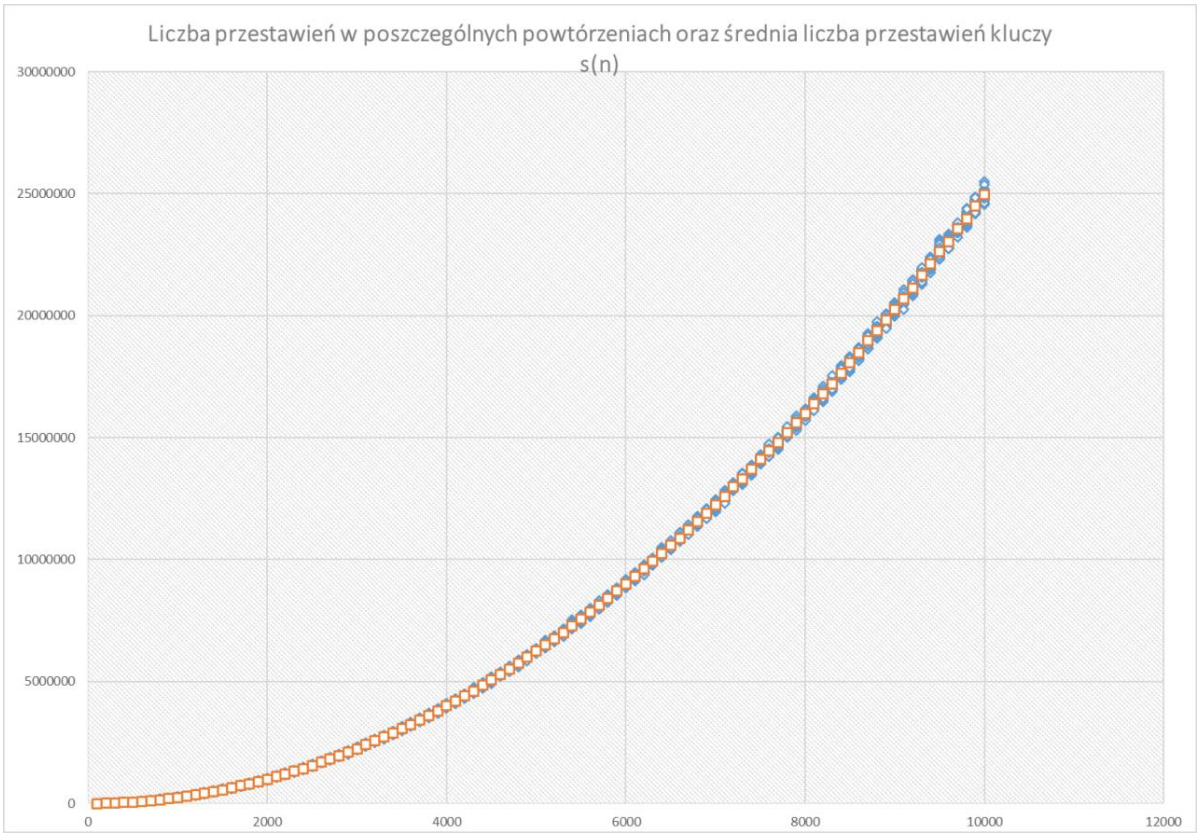
Prowadzone badania:

a) Liczba wykonanych porównań w poszczególnych powtórzeniach oraz średnia liczba porównań $cmp(n)$ jako funkcję n:



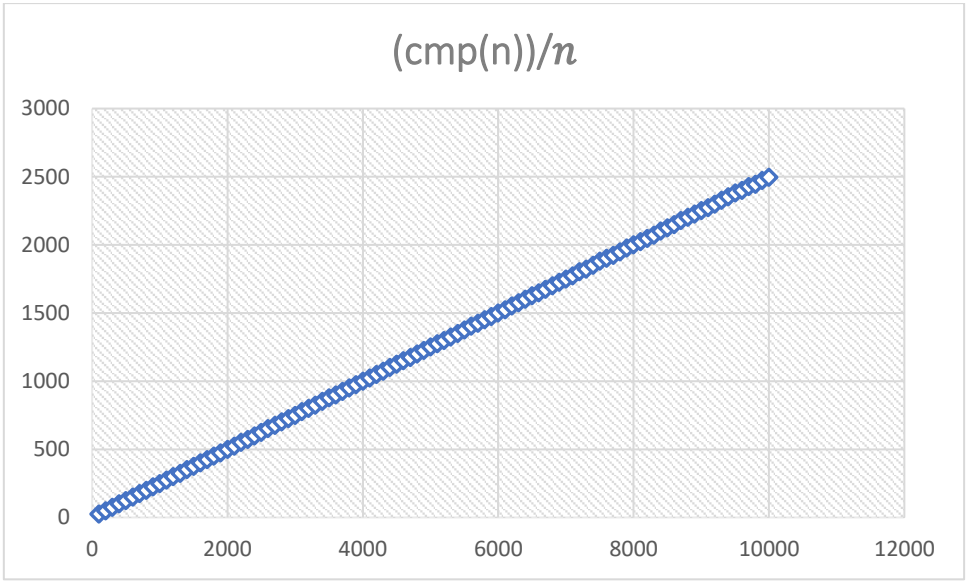
Wniosek: W miarę zwiększania się rozmiaru tablicy, liczba porównań w algorytmie sortowania przez wstawianie (Insertion Sort) rośnie proporcjonalnie, co sugeruje, że efektywność tego algorytmu maleje dla większych zbiorów danych. Ta obserwacja wynika z faktu, że zwiększająca się liczba porównań wpływa negatywnie na czas wykonania sortowania. W rezultacie, chociaż Insertion Sort może być skuteczny dla niewielkich tablic lub już częściowo posortowanych danych, jego liniowa zależność od liczby porównań sprawia, że staje się mniej atrakcyjny w kontekście efektywności dla dużych zbiorów danych. W takich przypadkach, bardziej zaawansowane algorytmy sortowania, takie jak sortowanie przez scalanie czy szybkie sortowanie, mogą być preferowane ze względu na ich lepszą wydajność i skuteczność.

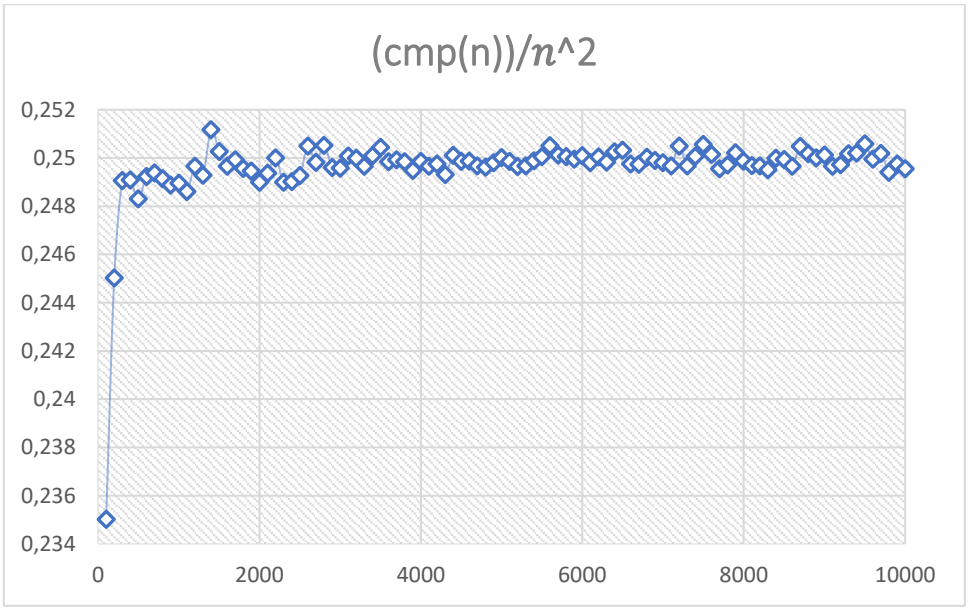
b) Liczba przestawień w poszczególnych powtórzeniach oraz średnia liczba przestawień kluczy $s(n)$ jako funkcję n:



Wniosek: W miarę wzrostu rozmiaru tablicy, liczba przestawień w algorytmie sortowania przez wstawianie (Insertion Sort) rośnie proporcjonalnie, co sugeruje, że dla większych zbiorów danych metoda ta może być bardziej kosztowna ze względu na zwiększającą się liczbę przestawień. Choć Insertion Sort może być skuteczny dla niewielkich tablic lub danych już częściowo posortowanych, dla dużych zbiorów jego zależność od liczby przestawień sprawia, że staje się mniej atrakcyjny pod względem wydajności.

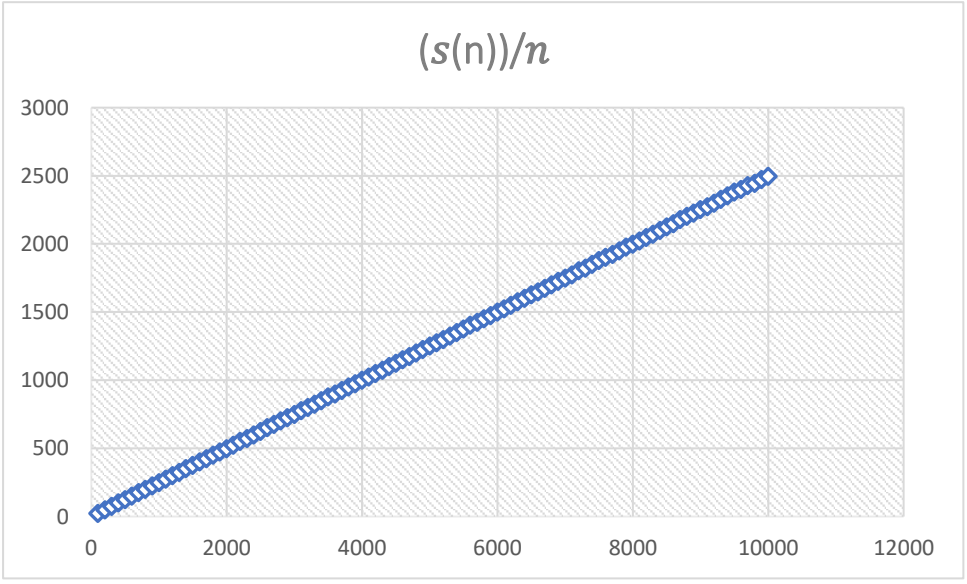
c) Iloraz $\frac{\mathrm{cmp(n)}}{\mathrm{n}}$ oraz $\frac{\mathrm{cmp(n)}}{\mathrm{n^2}}$ jako funkcje n:

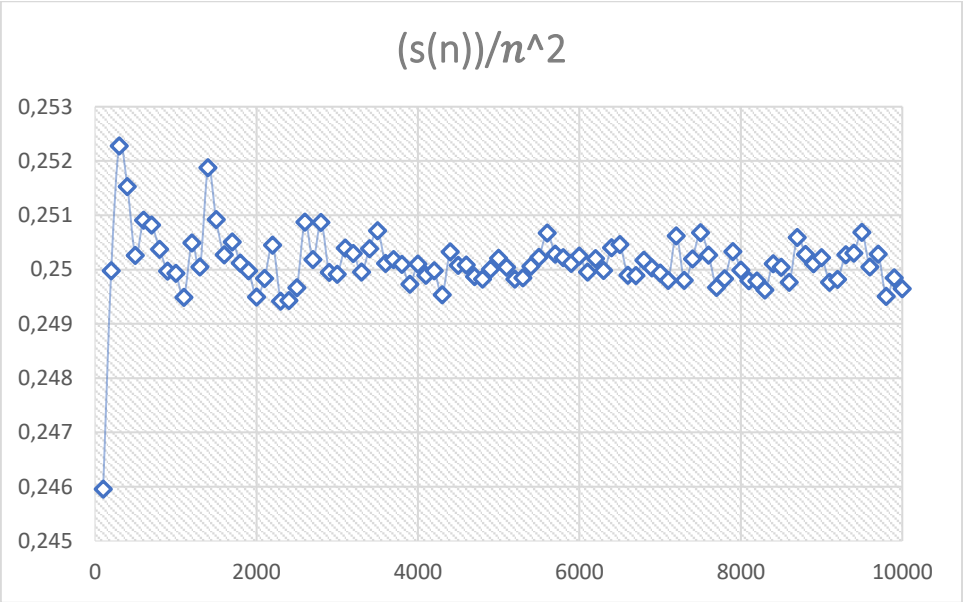




$$CMP(n) = O(n^2)$$

d) Iloraz $\frac{\mathrm{s(n)}}{\mathrm{n}}$ oraz $\frac{\mathrm{s(n)}}{\mathrm{n^2}}$ jako funkcje n:




(s(n))/n^2
0,253
0,252
0,251
0,25
0,249
0,248
0,247
0,246
0,245
0
2000
4000
6000
8000
10000

$$S(n) = O(n^2).$$