# Juliusz Wasieleski Informatyka II rok, grupa 1

Algorytmy geometryczne, laboratorium 2 - sprawozdanie

# 1.Opis ćwiczenia

Na drugich zajęciach laboratoryjnych naszym zadaniem było wyznaczanie otoczki wypukłej dla zbiorów punktów. Do wykonania go używaliśmy algorytmów Grahama i Jarvisa:

**Algorytm Grahama:**

* + - 1. W zbiorze S wybieramy punkt p0 o najmniejszej współrzędnej y. Jeżeli jest kilka takich punktów, to wybieramy ten z nich, który ma najmniejszą współrzędną x
      2. Sortujemy pozostałe punkty ze względu na kąt, jaki tworzy wektor (p0,p) z dodatnim kierunkiem osi x. Jeśli kilka punktów tworzy ten sam kąt, usuwamy wszystkie z wyjątkiem najbardziej oddalonego od p0

Niech uzyskanym ciągiem będzie p1, p2,...p m

* + - 1. Do początkowo pustego stosu s wkładamy punkty p0, p1 , p2. t – indeks stosu; i ← 3
      2. while i<m do:

if pi leży na lewo od prostej (pt-1, pt)

then push(pi), i ← i+1

else pop(s)

* + - 1. Każdy wierzchołek, który został wyrzucony ze stosu, nie należy do otoczki, a po wykonaniu powyższej pętli, stos, który pozostał jest otoczką

**Koszt algorytmu:**

Operacje dominujące to porównywanie współrzędnych lub badanie położenia punktu względem prostej.

**O(n) + O(n logn) + O(1) + O(n-3) = O(n logn)**

szukanie sortowanie inicjalizacja krok 4  
minimum stosu

**Algorytm Jarvisa:**

1. Znajdź punkt i0 z S o najmniejszej współrzędnej y; i ← i0
2. Repeat

for j ≠ i do

znajdź punkt, dla którego kąt liczony przeciwnie do wskazówek zegara w   
 odniesieniu do ostatniej krawędzi otoczki jest najmniejszy

niech k będzie indeksem punktu z najmniejszym kątem zwróć (p i, p k) jako  
 krawędź otoczki i ← k

until i = i0

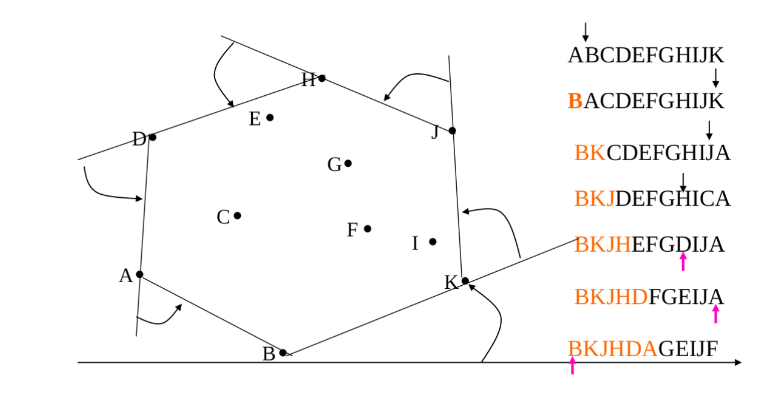
**Złożoność obliczeniowa:**

**O(n2)**

Gdy liczba wierzchołków otoczki jest ograniczona przez stałą **k**, jego złożoność jest rzędu

**O(k\*n)**

rys 1.



# 2. Środowisko, biblioteki, założenia oraz użyte narzędzia

Ćwiczenie wykonałem w Jupyter Notebook i napisałem w języku Python. Aby policzyć wyznaczniki korzystałem z bibliotek *pandas* oraz *numpy.*

Do rysowania wykresów użyłem biblioteki *matplotlib*  oraz *seaborn*, która obrazowała rozkład danych w postaci graficznej. Za wartość ε przyjąłem 10-12.

Do graficznej reprezentacji punktów oraz otoczek przyjąłem konwencję:

- na czerwono wygenerowane punkty

- na niebiesko otoczka

Do graficznej reprezentacji czasów wyznaczania otoczki przyjąłem konwencję:

- na niebiesko czasy algorytmu Grahama

- na pomarańczowo czasy algorytmu Jarvisa

Wszystkie obliczenia prowadziłem na komputerze Lenovo Y50-70 z systemem Windows 10 Pro w wersji 10.0.19045, procesor Intel Core i7-4720HQ 2.60GHz, 2601 MHz, rdzenie: 4, procesory logiczne: 8.

# 3. Plan i sposób wykonania ćwiczenia

Na początku należało wygenerować zbiory punktów o współrzędnych rzeczywistych typu float:  
a) zawierający 100 losowo wygenerowanych punktów o współrzędnych z przedziału [-100, 100]

b) zawierający 100 losowo wygenerowanych punktów leżących na okręgu o środku (0,0) i promieniu R=10

c) zawierający 100 losowo wygenerowanych punktów leżących na bokach prostokąta o wierzchołkach (-10, 10), (-10,-10), (10,-10), (10,10)

d) zawierający wierzchołki kwadratu (0, 0), (10, 0), (10, 10), (0, 10) oraz punkty wygenerowane losowo w sposób następujący: po 25 punktów na dwóch bokach kwadratu leżących na osiach i po 20 punktów na przekątnych kwadratu.

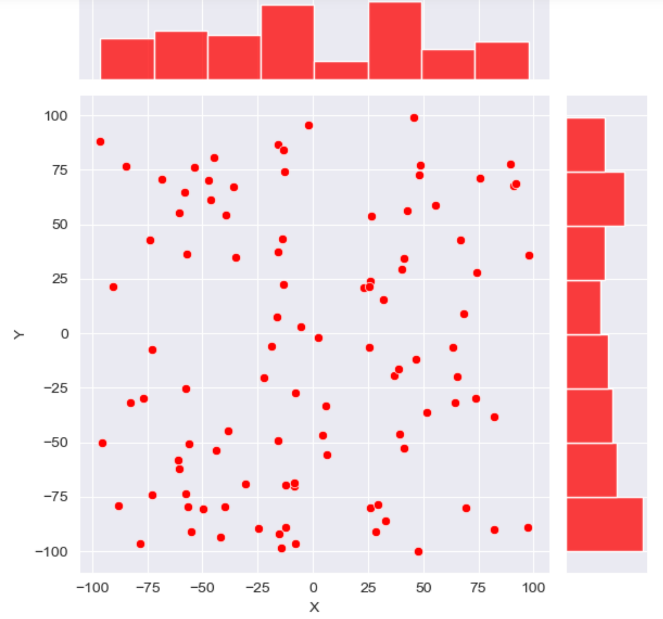
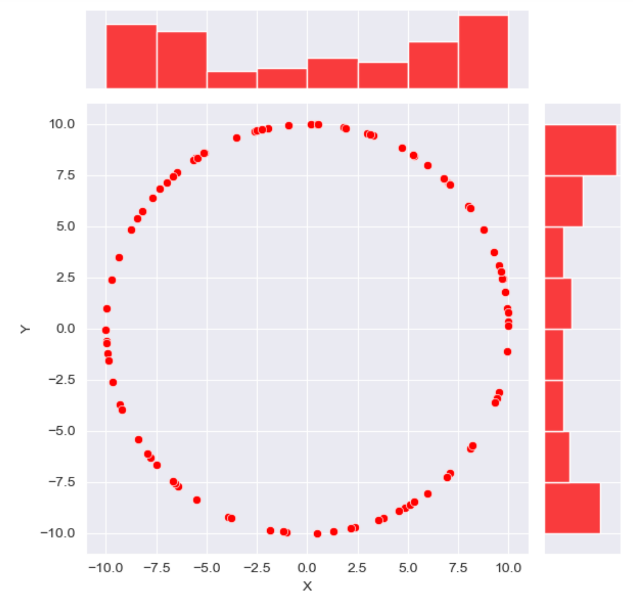
Kolejnym krokiem było zaimplementowanie funkcji wyznaczających otoczki wypukłe, algorytmów Grahama oraz Jarvisa.

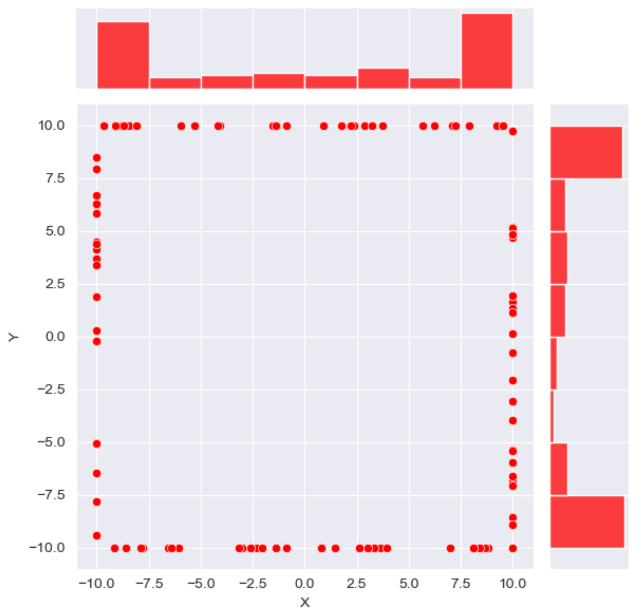
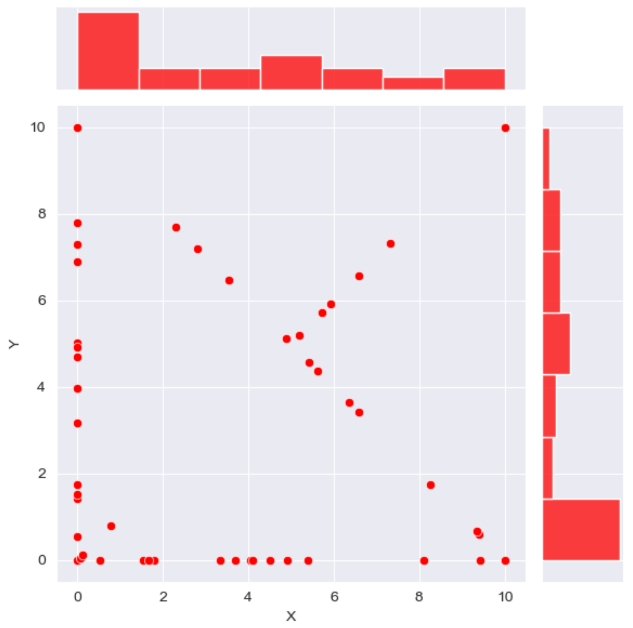
Następnie przedstawiłem graficznie uzyskane otoczki oraz sprawdziłem różnice w czasie wykonania obu algorytmów dla różnych zbiorów punktów, o różnej wielkości

# 4. Wykonanie ćwiczenia

## 4.1 Wygenerowanie punktów

Do wygenerowania punktów użyłem funkcji *uniform* z biblioteki *random*, następnie przedstawiłem na wykresach otrzymane zbiory punktów:

a) Zbiór A: b) Zbiór B:  
     
(rys. 2) (rys.3)

c) Zbiór C: d) Zbiór D:  
   
(rys.4) (rys.5)

Dla wszystkich wykresów został użyty wykres punktowy (scatter plot). Wykresy słupkowe u góry wykresów oraz z boku pokazują dystrybucję w danych obszarach odpowiednio x-ów u góry oraz y-ów z prawego boku. Również osie są podpisane u dołu ‘X’ oraz z lewego boku ‘Y’.

## 4.2 Funkcje wyznaczające zbiory zadanej wielkości

Aby wykonać powyższe zadanie zaimplementowałem funkcję:

a) *data\_a\_parameters(num\_of\_points, ranges)* - generowanie zbioru A ale o zadanej liczności oraz zadanych granicach

b) *data\_b\_parameters(num\_of\_points, middle\_point, radius)* - generowanie zbioru B ale o zadanej liczności, zadanym środku okręgu i zadanym promieniu

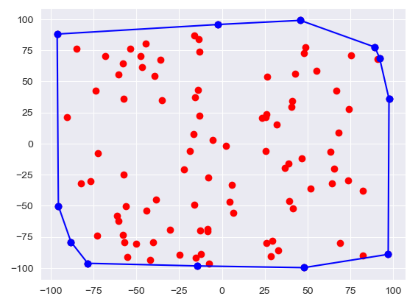
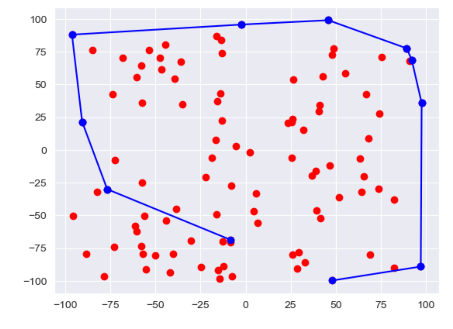
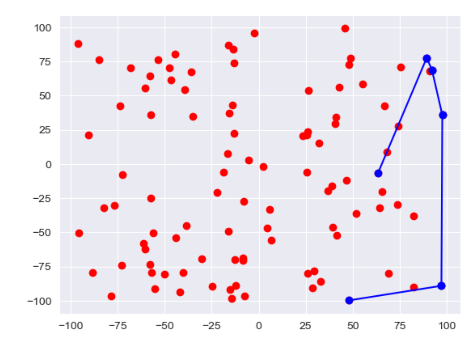
c) *data\_c\_parameters(num\_of\_points, leftLower, rightUpper)* - generowanie zbioru C ale o zadanej liczności oraz zadanych wierzchołkach kwadratu

d) *data\_d\_parameters(points, axis\_points, diagonal\_points)* - generowanie zbioru D ale o zadanej liczności, wierzchołkach kwadratu oraz liczby punktów na przekoątnych

## 4.3 Wyznaczenie otoczek dla zbiorów z podpunktu 4.1

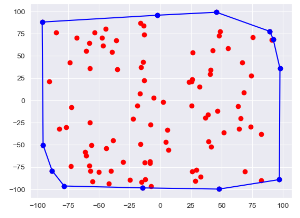
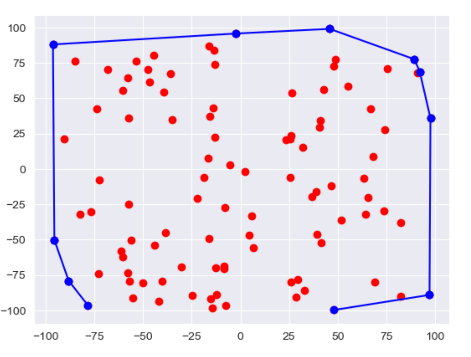
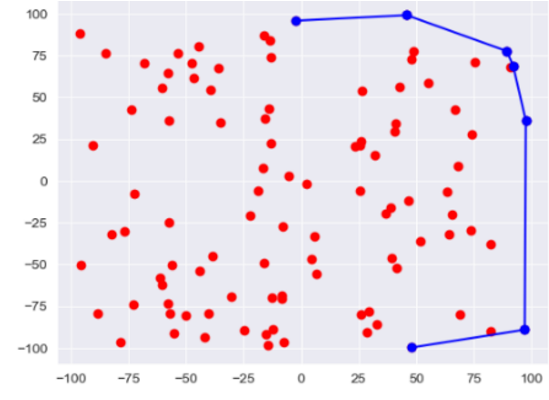
Algorytmy Grahama oraz Jarvisa realizują odpowiednio funkcja *graham* oraz *jarvis.* W algorytmie Grahama zaimplementowałem moje sortowanie oparte na algorytmie *QuickSort*, ale zmodyfikowałem funkcję procedurę *partition*, tak żeby sortowała punkty ze względu na kąt jaki dany punkt tworzy z prostą łączącą minimalny punkt i pivot. Kąt wyznaczam przy użyciu funkcji orient, w której zaimplementowałem wyznacznik 3x3. Dodatkowo podczas sortowania, jeśli dany punkt leży na prostej łączącej punkt minimalny i pivot sprawdzam odległości w sensie metryki Euklidesowej. Z faktu sortowania korzystam później w 4 kroku algorytmu Grahama i nie wykonuję niepotrzebnych porównań. W algorytmie Jarvisa, podczas szukania kolejnego punktu do otoczki, sprawdzam dystanse między punktami jeśli wyznacznik jest równy zero. Robię tak, ponieważ wtedy mogę uniknąć dodawania do otoczki punktów, które leżą na jej brzegu, ale nie są wierzchołkiem. W ten sposób zmniejszam stałą k, a co za tym idzie moja implementacja jest szybsza.Wszystkie kroki wyznaczania otoczek umieszczone są w jupyter notebooku, a poniżej umieszczam przykładowe stadia działania tych algorytmów:

### Zbiór A: 1.Algorytm Grahama:



(rys.6) (rys.7) (rys.8)

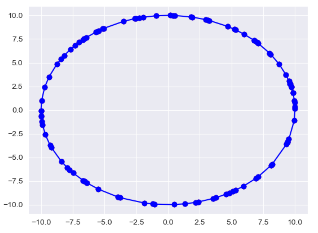
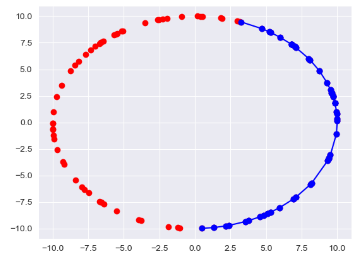
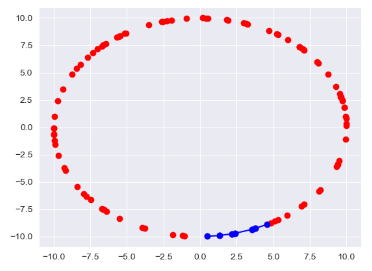
2.Algorytm Jarvisa



(rys.9) (rys.10) (rys.11)

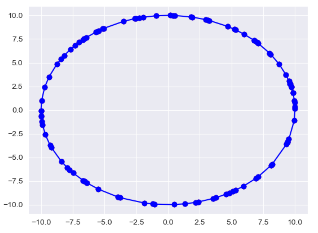
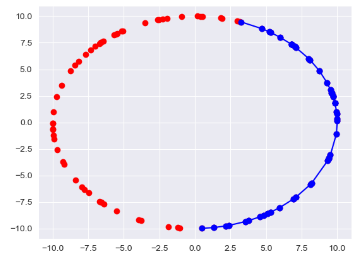
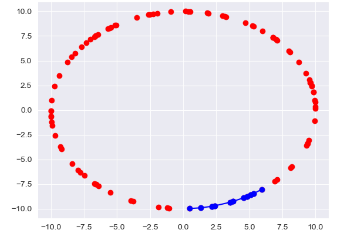
### Zbiór B:

1.Algorytm Grahama:



(rys.12) (rys.13) (rys.14)

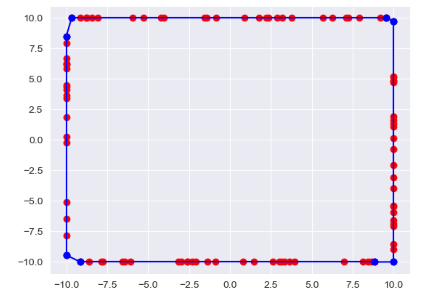
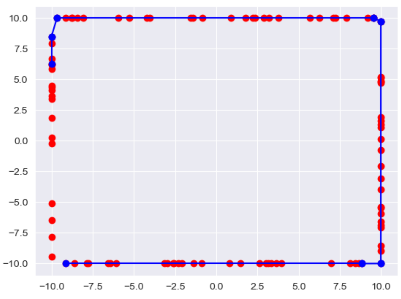
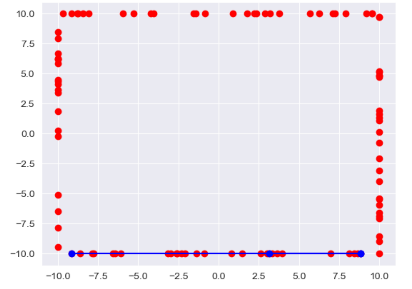
2.Algorytm Jarvisa:



(rys.15) (rys.16) (rys.17)

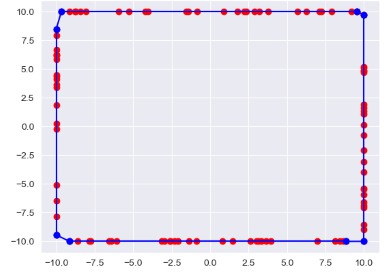
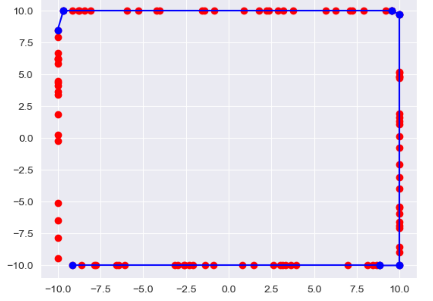
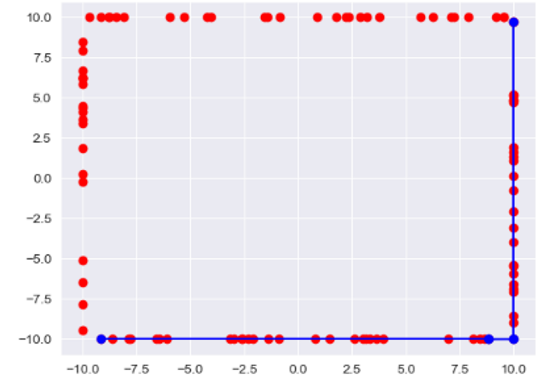
### Zbiór C:

1.Algorytm Grahama:



(rys.18) (rys.19) (rys.20)

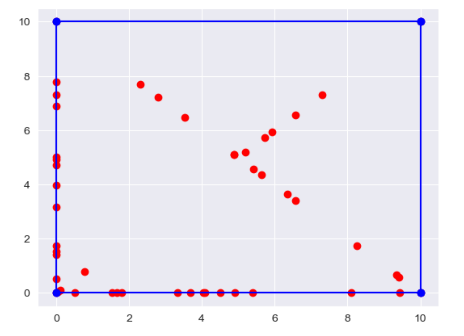
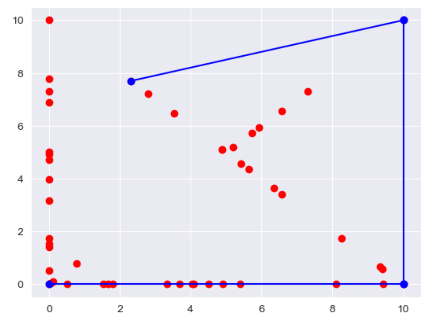
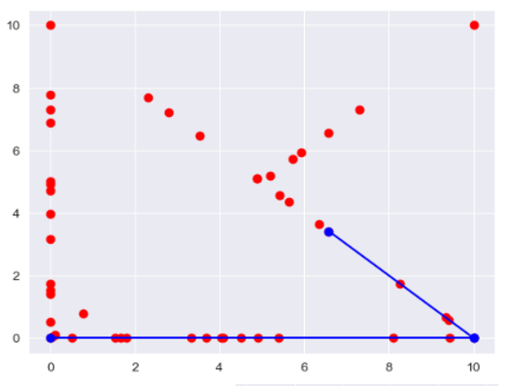
2.Algorytm Jarvisa:



(rys.21) (rys.22) (rys.23)

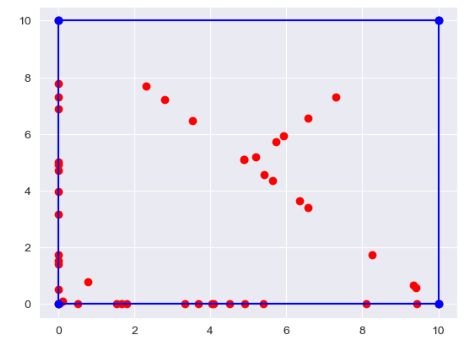
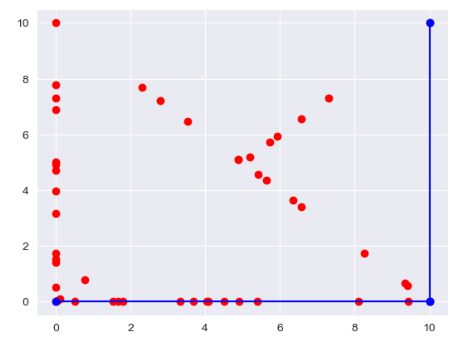
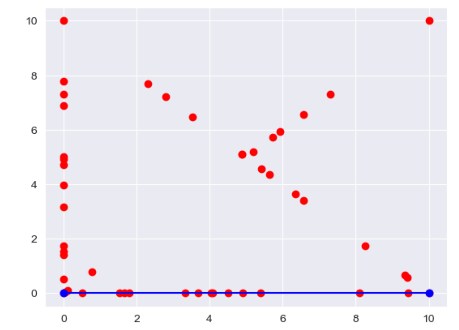
### Zbiór D:

1.Algorytm Grahama:



(rys.24) (rys.25) (rys.26)

1. Algorytm Jarvisa:



(rys.27) (rys.28) (rys.29)

5. Porównanie czasu działania algorytmów

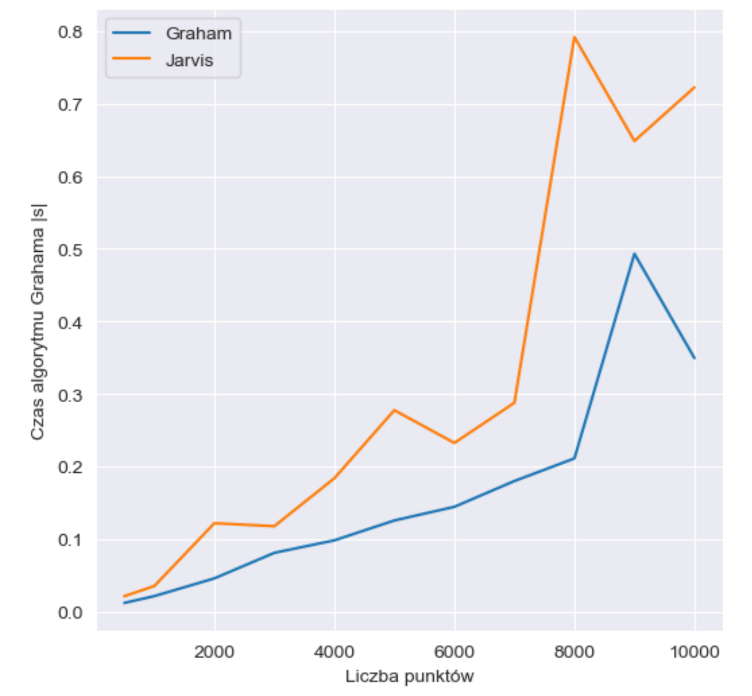
Do porównania czasu działania algorytmów użyłem funkcji count\_time, w której za samo porównywanie czasów odpowiadała funkcja perf\_counter z biblioteki time. Następnie z pobranych danych stworzyłem tabelkę która porównuje czasy algorytmów. W tych tabelach wyświetlam ilość punktów oraz ich zakres, następnie czas działania algorytmu Grahama w sekundach i czas działania algorytmu Jarvisa, na koniec wypisywałem, który algorytm jest szybszy i o ile czasu. Pod tabelami umieściłem wykresy obrazujące graficznie czasy wykonania algorytmów.

5.1 Porównanie czasu działania algorytmów bez flag

a) Zbiór A, losowe punkty z przedziału

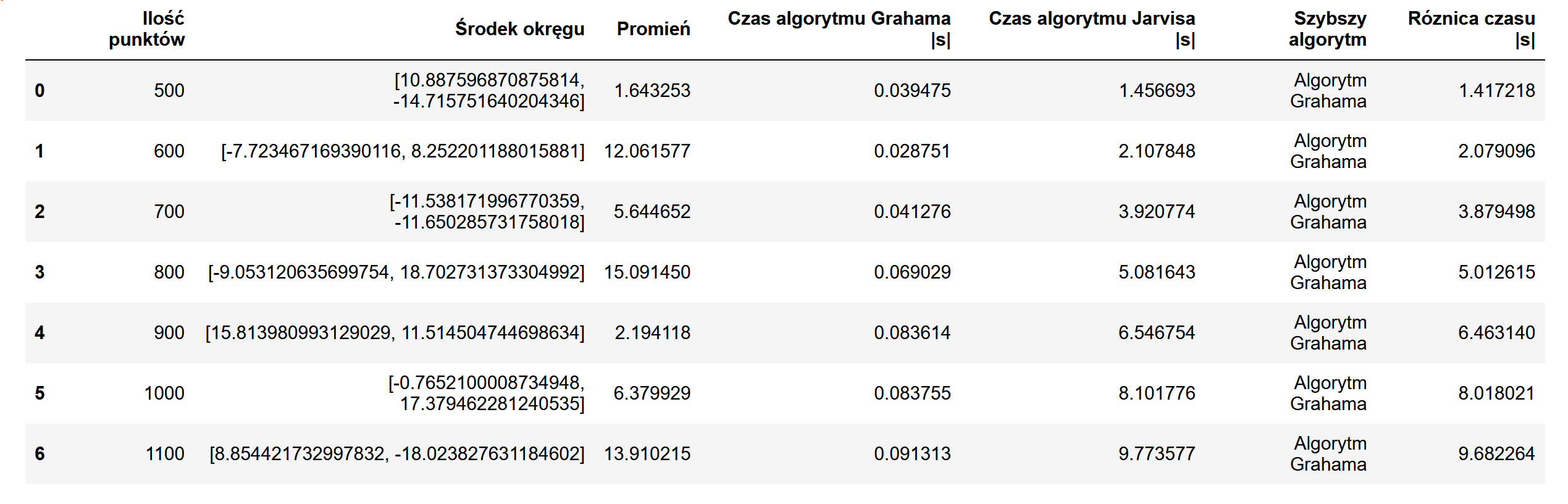


Tab 1.

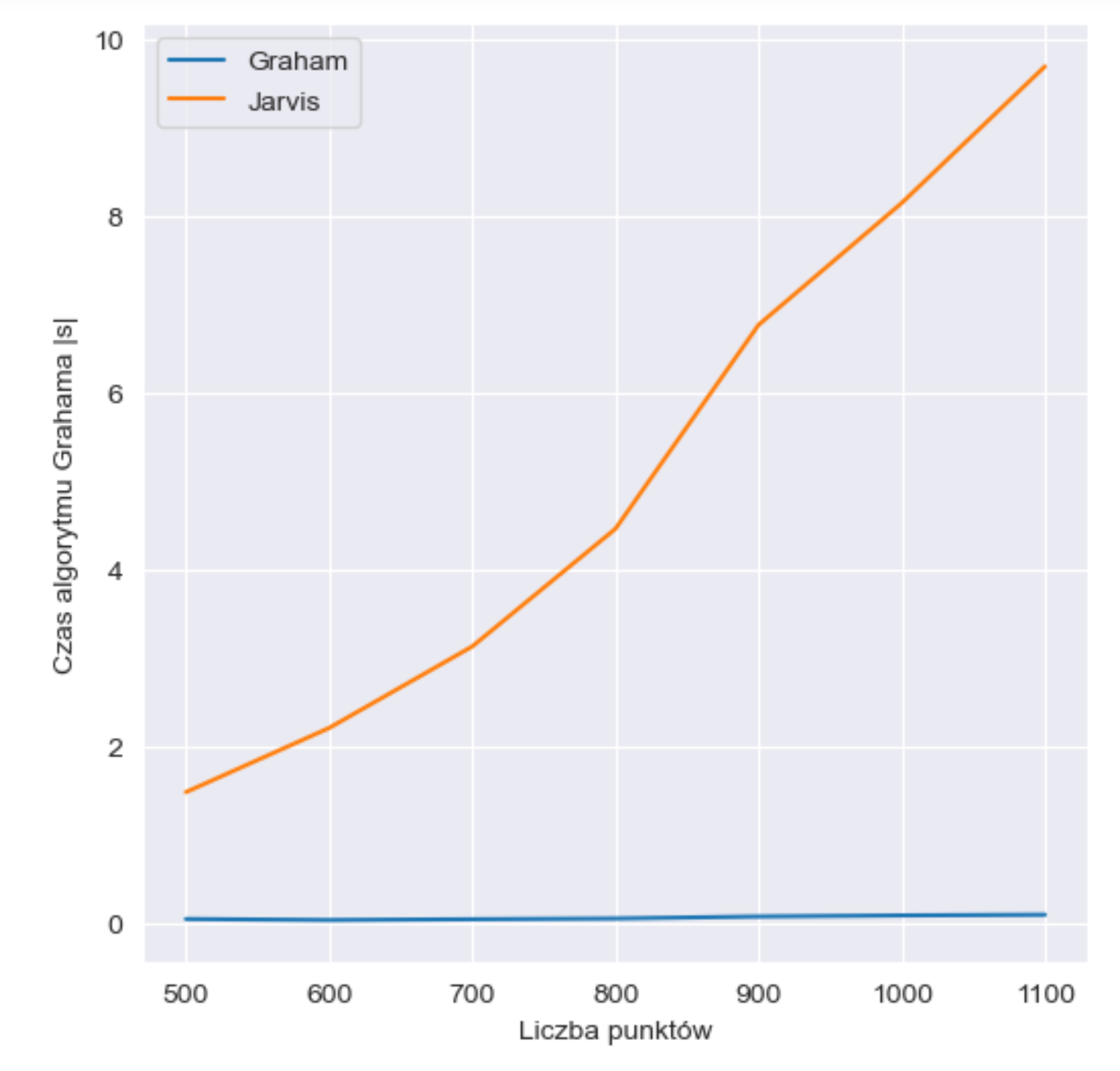


Rys. 30

b) Zbiór B, losowe punkty leżące na okręgu

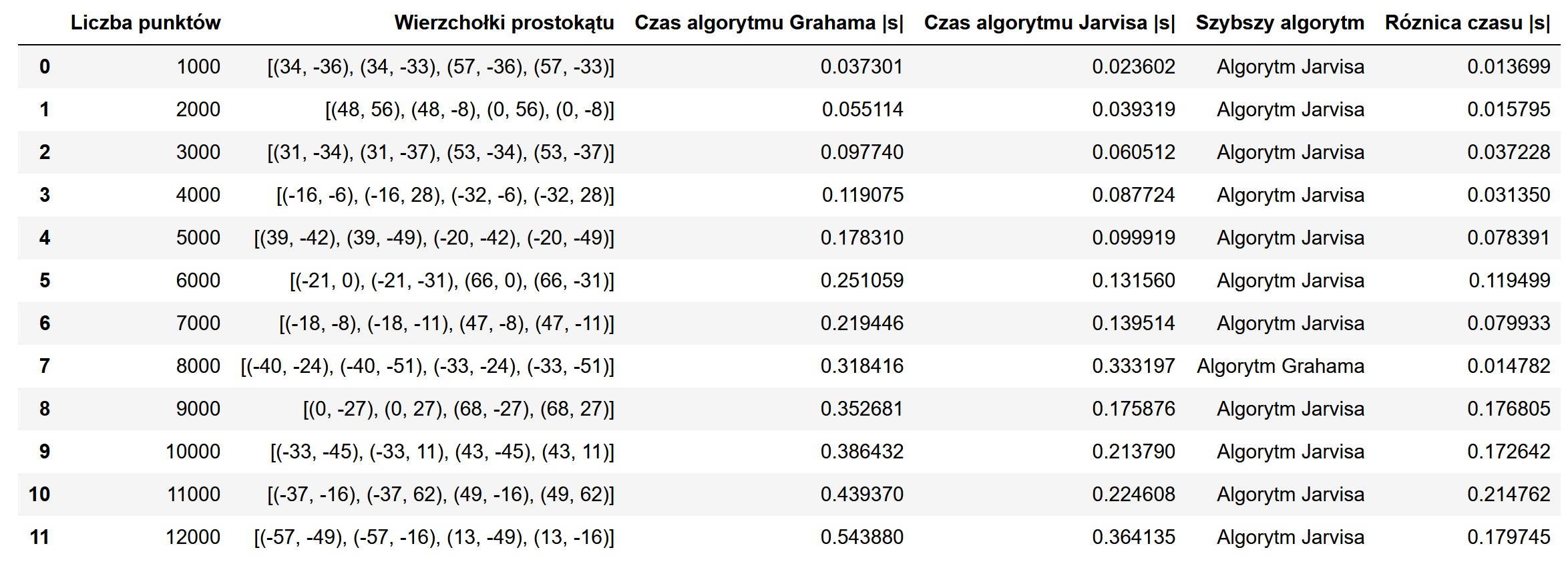


Tab 2.

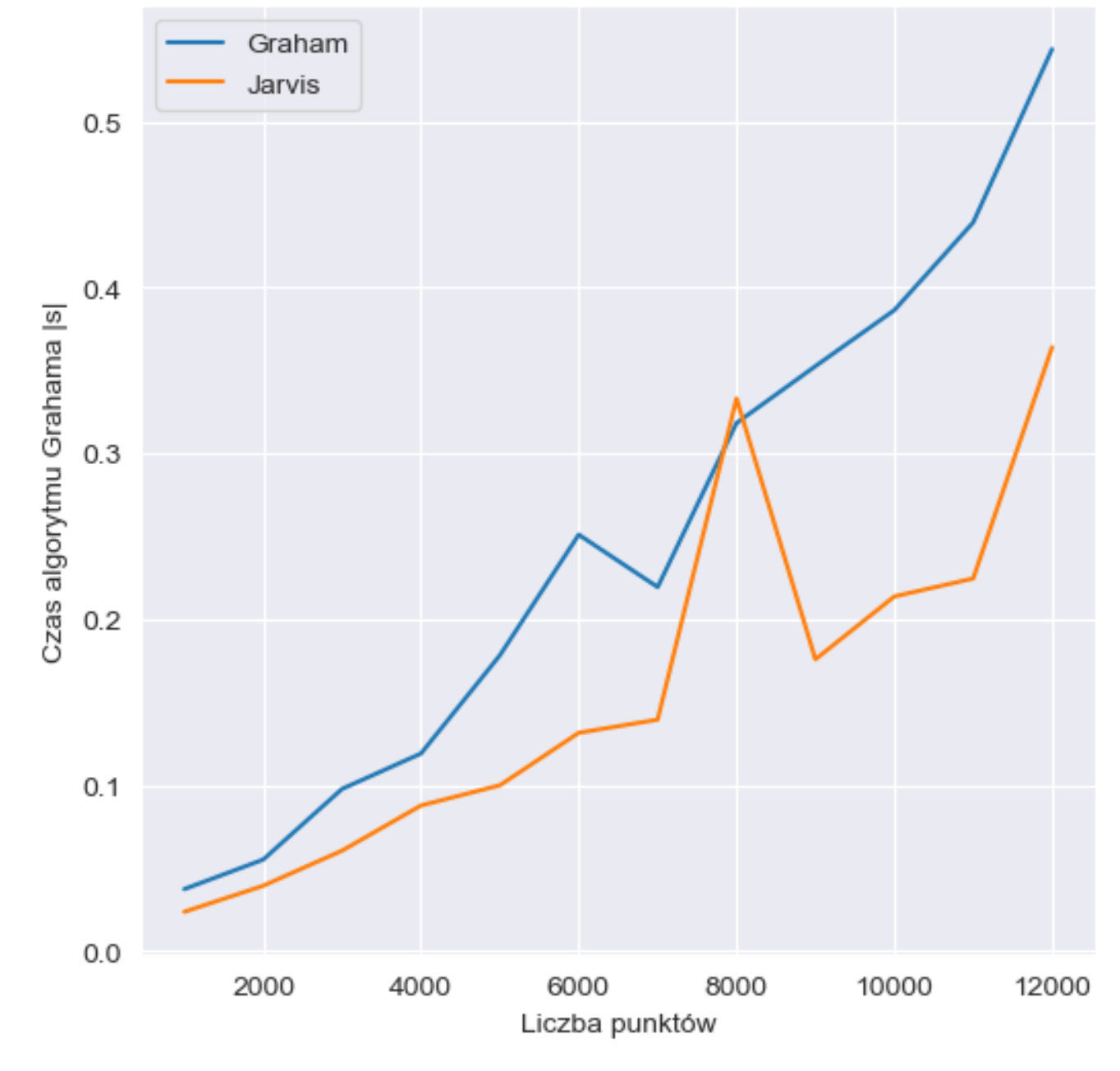


Rys. 31

c) Zbiór C, losowe punkty leżące na prostokącie

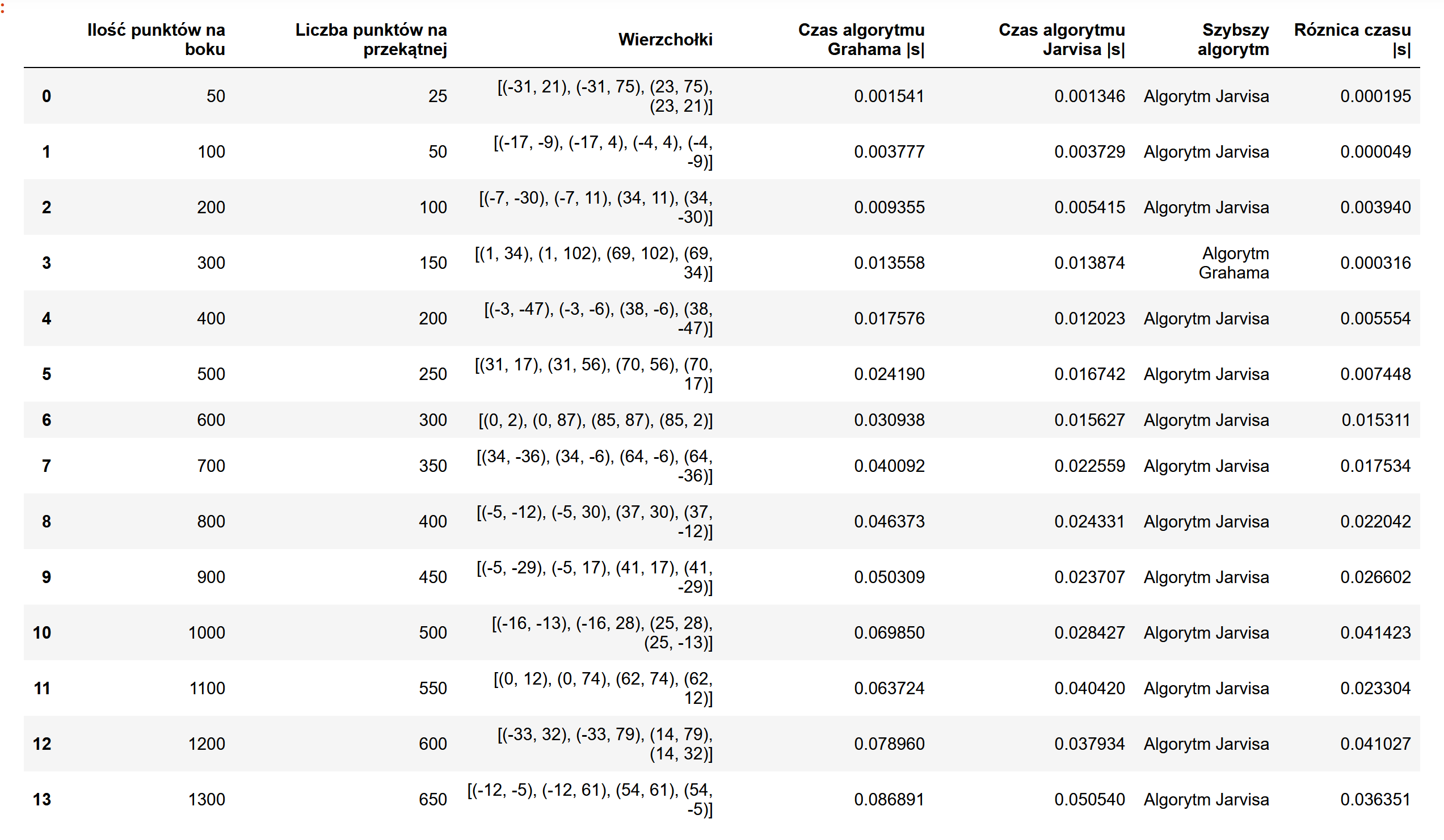


Tab. 3

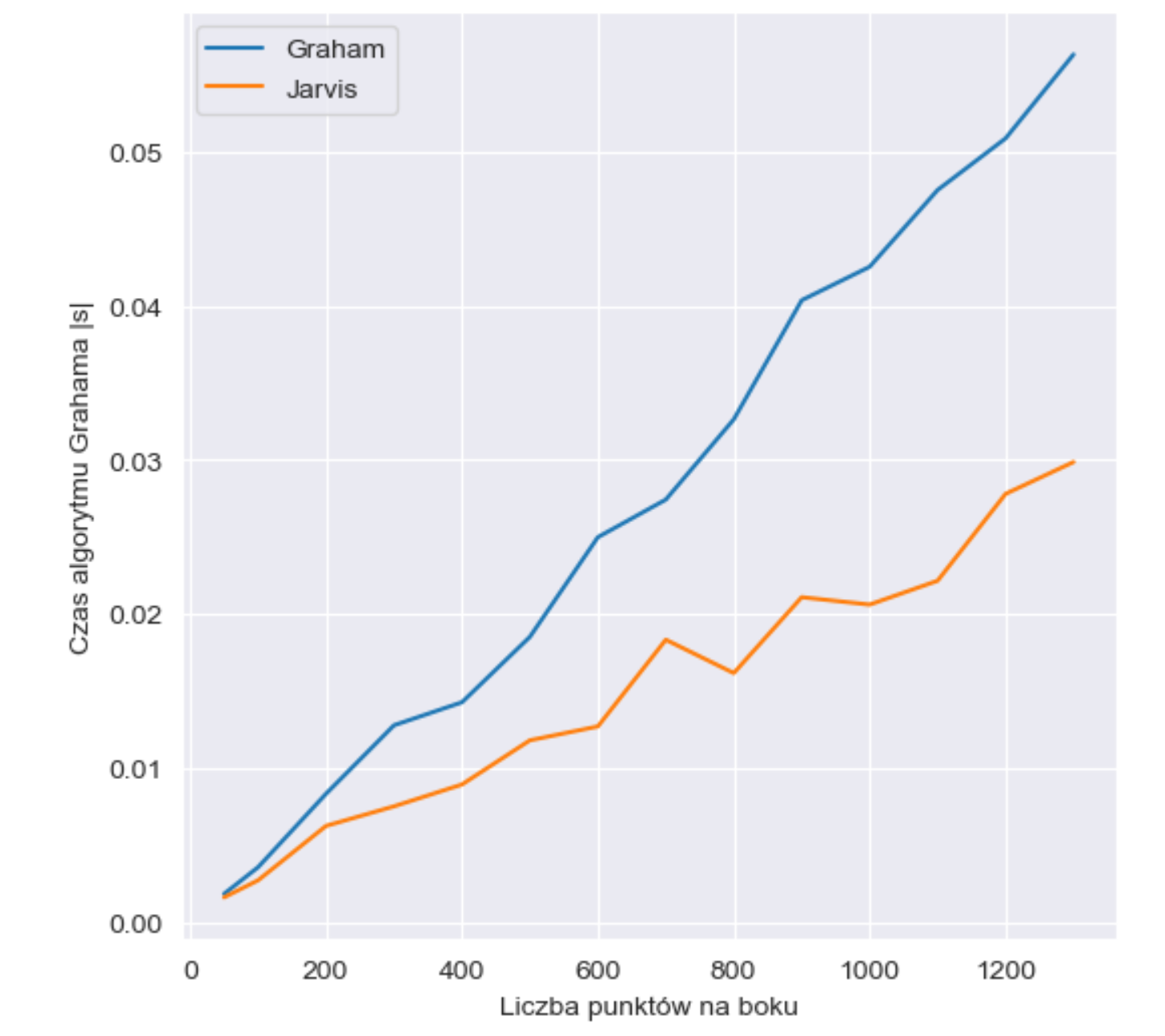


Rys. 32

d) zbiór D, losowe punkty leżące na dwóch bokach kwadratu i na przekątnych



Tab 4.



Rys. 33

# 7. Wnioski

Algorytm Grahama w moim przypadku okazał się szybszy od Jarvisa dla zbiorów A i B, natomiast dla zbiorów C i D rola się odwróciła. Zapewne jest to efektem charakterystyki zbiorów danych i mojej implementacji. Ze względu na to, że dla zbiorów C i D otoczka składała się z kilku punktów więc k<<n, gdzie k to ilość punktów na otoczce, a n to ilość punktów zbiorze. Wtedy złożoność O(nk) była lepsza niż O(nlogn). Natomiast w tych zbiorach dla mniejszych wartości n powinniśmy obserwować sytuację, w której algorytm Grahama jest szybszy. Nie dzieje się tak zapewne ze względów implementacyjnych, bardzo możliwe, że mój algorytm Grahama ma dużą stałą. Dla zbiorów A i B szybszy jest algorytm Grahama, gdyż w otoczce mamy więcej punktów niż w zbiorach C i D. Na podstawie wykresów 30-33 oraz tabelek 1-4 możemy stwierdzić że ich działanie zgadza się z ich przewidywaną złożonością.