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
3. Do początkowo pustego stosu s wkładamy punkty p0, p1 , p2. t – indeks stosu; i ← 3
4. while i<m do:
   if p_i leży na lewo od prostej (p_{t-1}, p_t)
   then push(p_i), i ← i+1
   else pop(s)
5. 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.

$$O(n) + O(n \log n) + O(1) + O(n-3) = O(n \log n)$$

szukanie minimum | sortowanie | inicjalizacja stosu | krok 4

Algorytm Jarvisa:

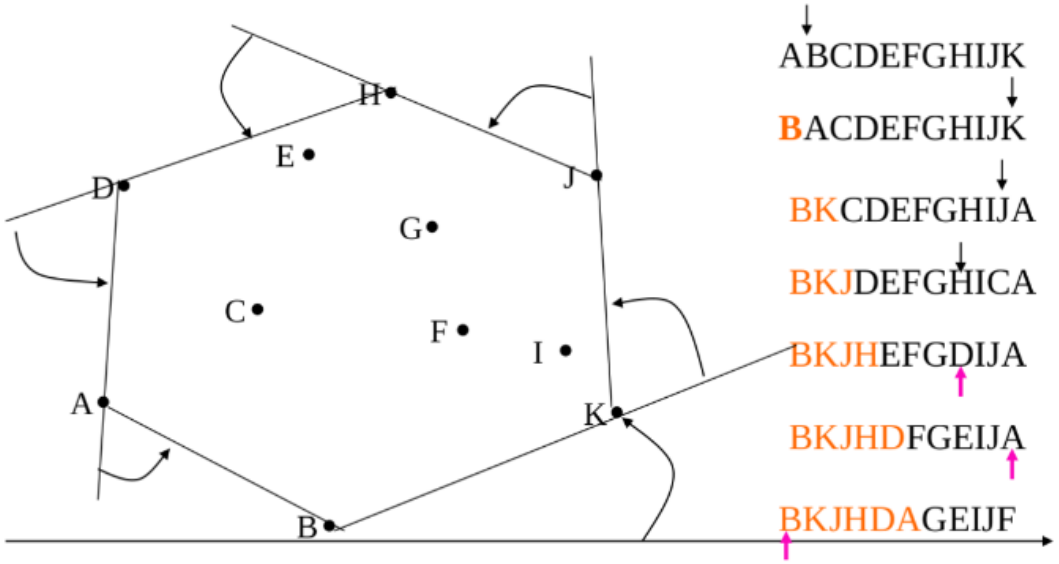
1. Znajdź punkt i0 z S o najmniejszej współrzędnej y; i ← i0
2. Repeat
   for j ≠ i do
   znajdź punkt, dla którego kąt liczony przeciwnie do wskazówek zegara w odniesieniu do ostatniej krawędzi otoczki jest najmniejszy
   niech k będzie indeksem punktu z najmniejszym kątem zwróć (p i, p k) jako krawędź otoczki i ← k
   until i = i0

Złożoność obliczeniowa:

$$O(n^2)$$

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.

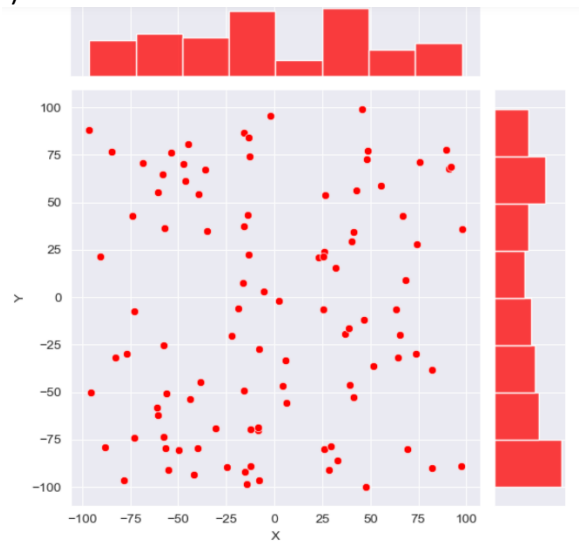
Następnie przedstawiłem graficznie uzyskane otoczki oraz sprawdziłem różnice w czasie wykonania obu algorytmów dla różnych zbiorów punktów, o różnej wielkości

4. Wykonanie ćwiczenia

4.1 Wygenerowanie punktów

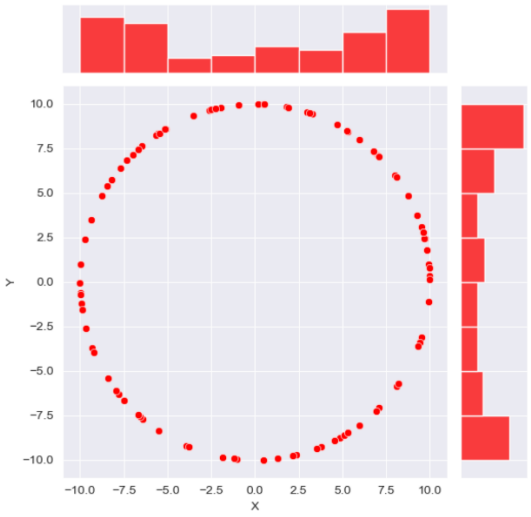
Do wygenerowania punktów użyłem funkcji *uniform* z biblioteki *random*, następnie przedstawiłem na wykresach otrzymane zbiory punktów:

a) Zbiór A:



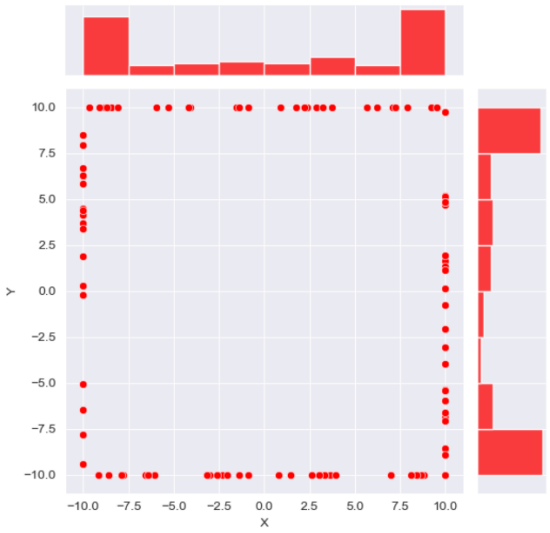
(rys. 2)

b) Zbiór B:



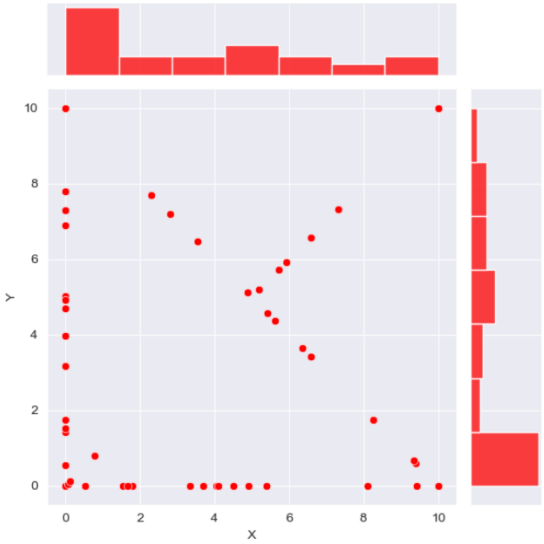
(rys.3)

c) Zbiór C:



(rys.4)

d) Zbiór D:



(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

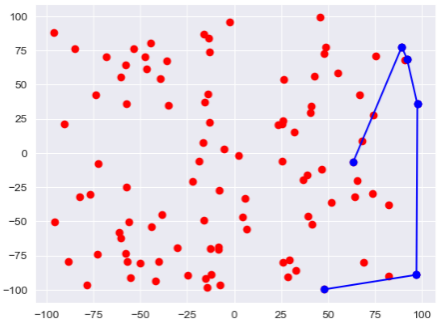
d) *data_d_parameters(points, axis_points, diagonal_points)* - generowanie zbioru D ale o zadanej liczności, wierzchołkach kwadratu oraz liczby punktów na przekoątnych

4.3 Wyznaczenie otoczek dla zbiorów z podpunktu 4.1

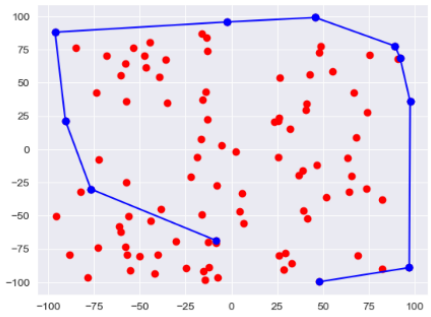
Algorytmy Grahama oraz Jarvisa realizują odpowiednio funkcja *graham* oraz *jarvis.* W algorytmie Grahama zaimplementowałem moje sortowanie oparte na algorytmie *QuickSort*, ale zmodyfikowałem funkcję procedurę *partition*, tak żeby sortowała punkty ze względu na kąt jaki dany punkt tworzy z prostą łączącą 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:

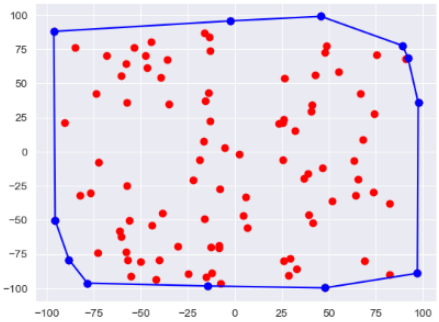
1.Algorytm Grahama:



(rys.6)

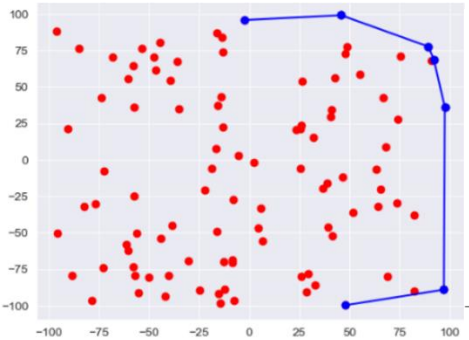


(rys.7)

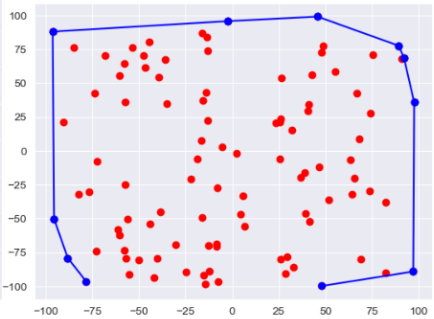


(rys.8)

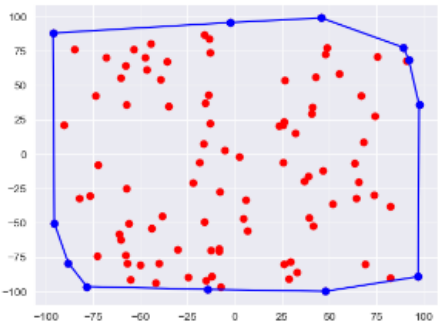
2.Algorytm Jarvisa



(rys.9)



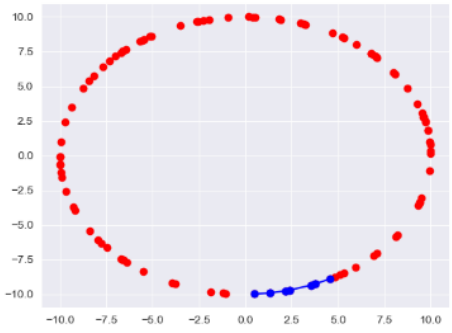
(rys.10)



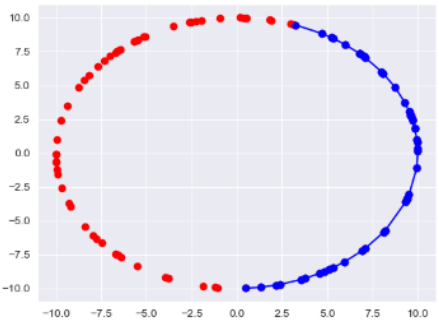
(rys.11)

Zbiór B:

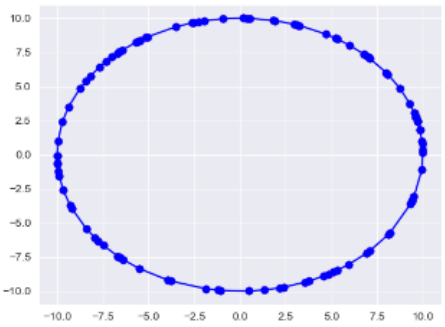
1.Algorytm Grahama:



(rys.12)

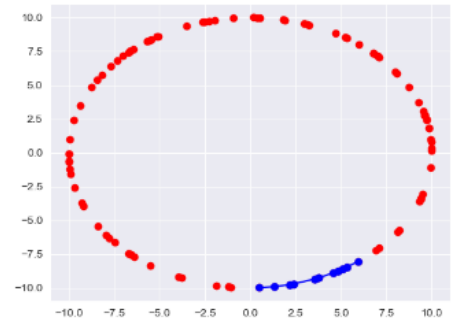


(rys.13)

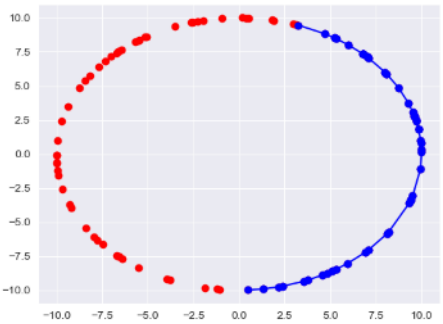


(rys.14)

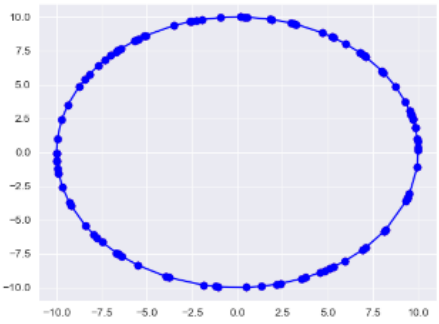
2.Algorytm Jarvisa:



(rys.15)



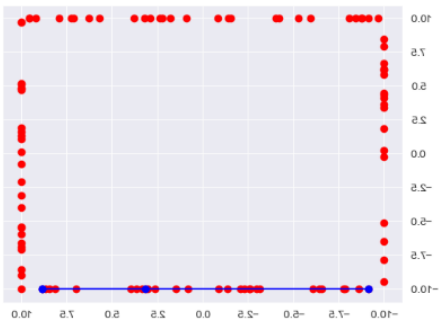
(rys.16)



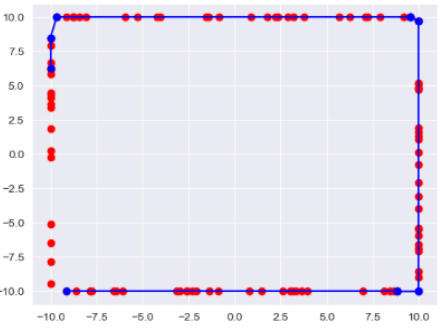
(rys.17)

Zbiór C:

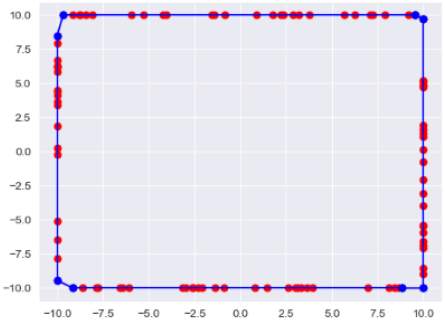
1.Algorytm Grahama:



(rys.18)



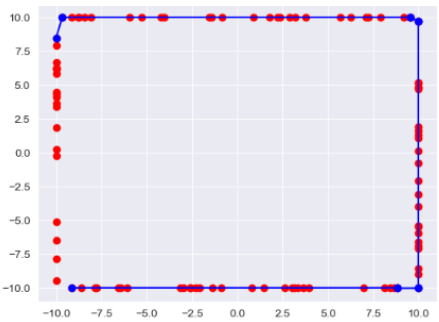
(rys.19)



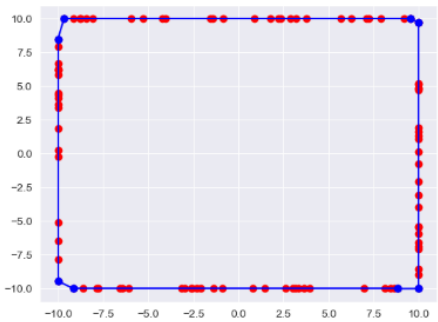
(rys.20)

2.Algorytm Jarvisa:

(rys.21)



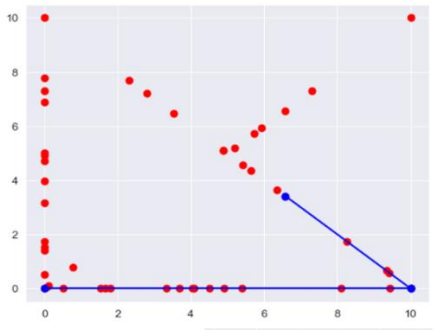
(rys.22)



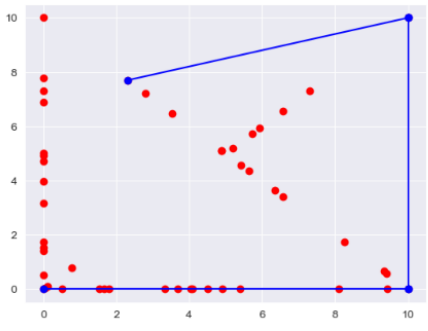
(rys.23)

Zbiór D:

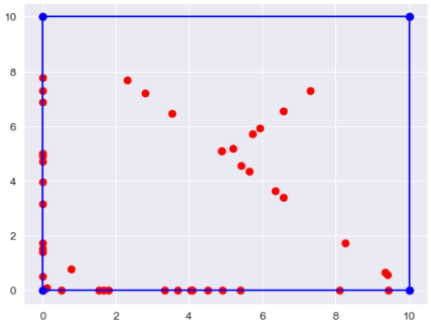
1.Algorytm Grahama:



(rys.24)

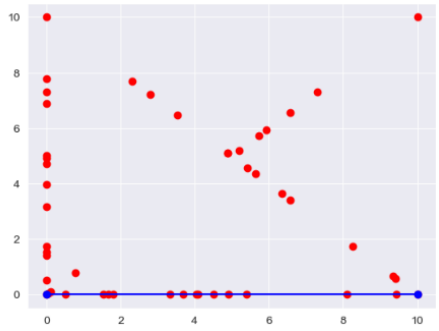


(rys.25)

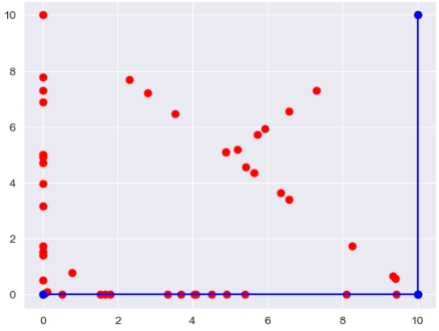


(rys.26)

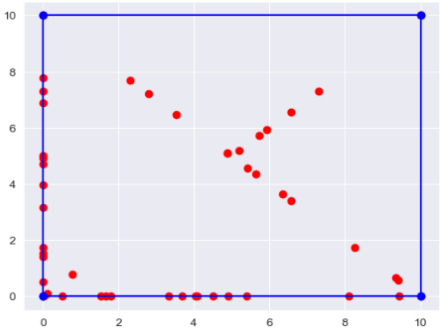
2. Algorytm Jarvisa:



(rys.27)



(rys.28)



(rys.29)

5. Porównanie czasu działania algorytmów

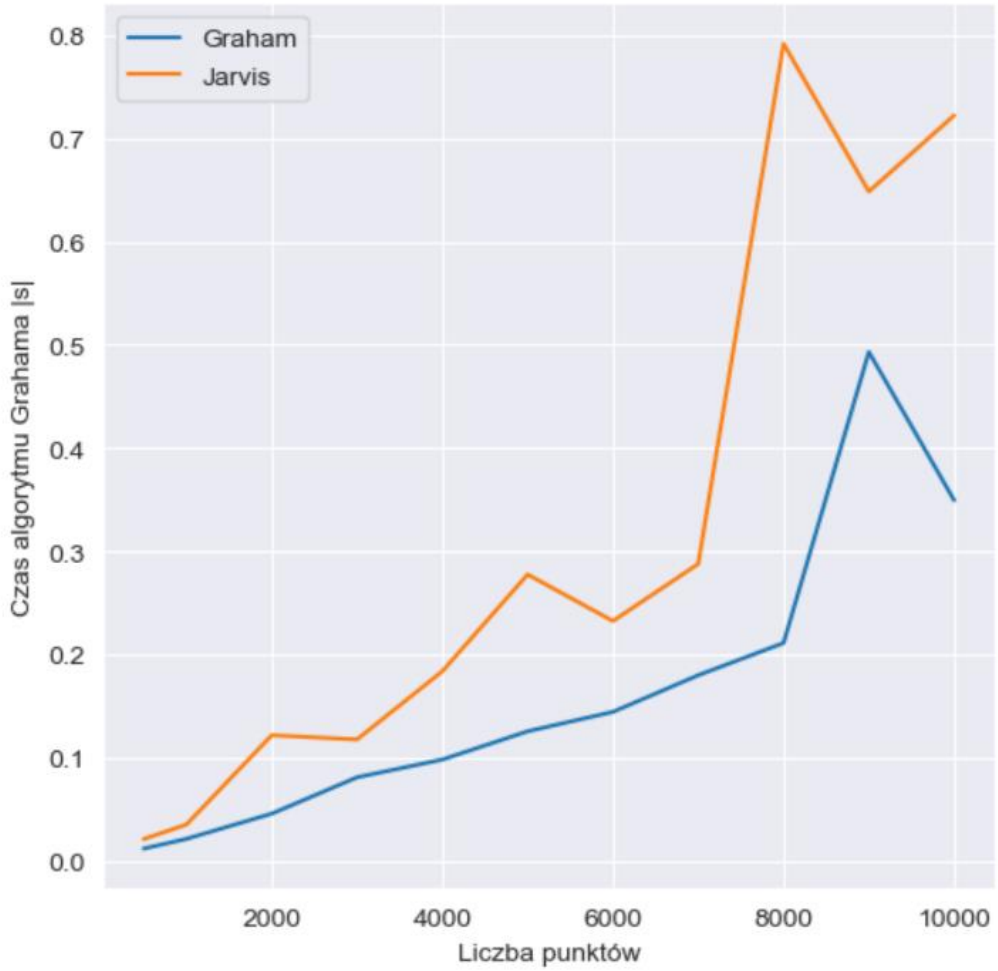
Do porównania czasu działania algorytmów użyłem funkcji count_time, w której za samo porównywanie czasów odpowiadała funkcja perf_counter z biblioteki time. Następnie z pobranych danych stworzyłem tabelkę która porównuje czasy algorytmów. W tych tabelach wyświetlam ilość punktów oraz ich zakres, następnie 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

	Ilość punktów	Początek zakresu	Koniec zakresu	Czas algorytmu Grahama |s|	Czas algorytmu Jarvisa |s|	Szybszy algorytm	Róznica czasu |s|
0	500	-200	200	0.020795	0.020625	Algorytm Jarvisa	0.000169
1	1000	-200	200	0.020382	0.032571	Algorytm Grahama	0.012189
2	2000	-200	200	0.048382	0.103802	Algorytm Grahama	0.055420
3	3000	-200	200	0.073228	0.119372	Algorytm Grahama	0.046145
4	4000	-200	200	0.092850	0.172533	Algorytm Grahama	0.079683
5	5000	-200	200	0.116078	0.200673	Algorytm Grahama	0.084595
6	6000	-200	200	0.171828	0.207462	Algorytm Grahama	0.035634
7	7000	-200	200	0.184937	0.480434	Algorytm Grahama	0.295497
8	8000	-200	200	0.290904	0.562574	Algorytm Grahama	0.271670
9	9000	-200	200	0.339625	0.617003	Algorytm Grahama	0.277377
10	10000	-200	200	0.402498	0.691845	Algorytm Grahama	0.289347

Tab 1.

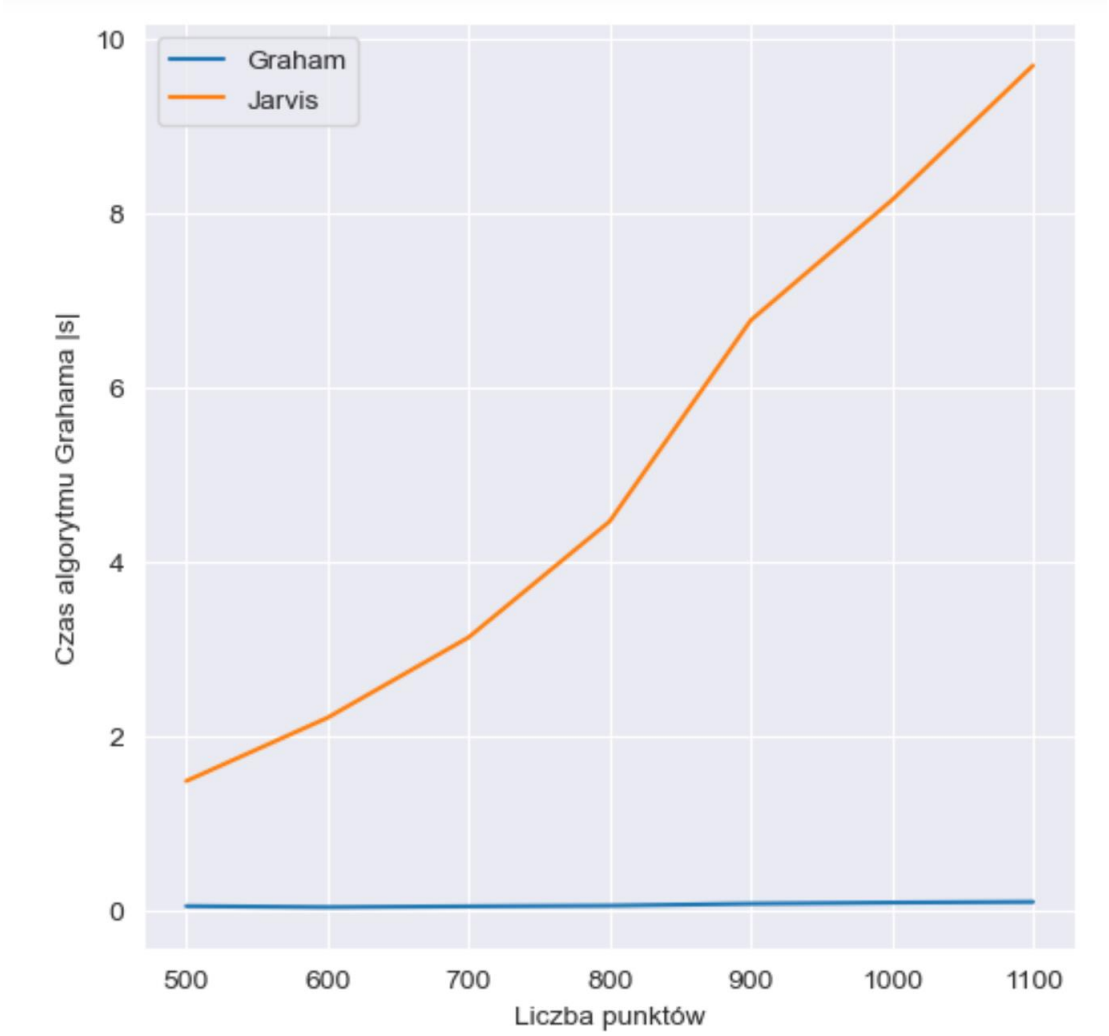


Rys. 30

b) Zbiór B, losowe punkty leżące na okręgu

	Ilość punktów	Środek okręgu	Promień	Czas algorytmu Grahama |s|	Czas algorytmu Jarvisa |s|	Szybszy algorytm	Róznica czasu |s|
0	500	[10.887596870875814, -14.715751640204346]	1.643253	0.039475	1.456693	Algorytm Grahama	1.417218
1	600	[-7.723467169390116, 8.252201188015881]	12.061577	0.028751	2.107848	Algorytm Grahama	2.079096
2	700	[-11.538171996770359, -11.650285731758018]	5.644652	0.041276	3.920774	Algorytm Grahama	3.879498
3	800	[-9.053120635699754, 18.702731373304992]	15.091450	0.069029	5.081643	Algorytm Grahama	5.012615
4	900	[15.813980993129029, 11.514504744698634]	2.194118	0.083614	6.546754	Algorytm Grahama	6.463140
5	1000	[-0.7652100008734948, 17.379462281240535]	6.379929	0.083755	8.101776	Algorytm Grahama	8.018021
6	1100	[8.854421732997832, -18.023827631184602]	13.910215	0.091313	9.773577	Algorytm Grahama	9.682264

Tab 2.

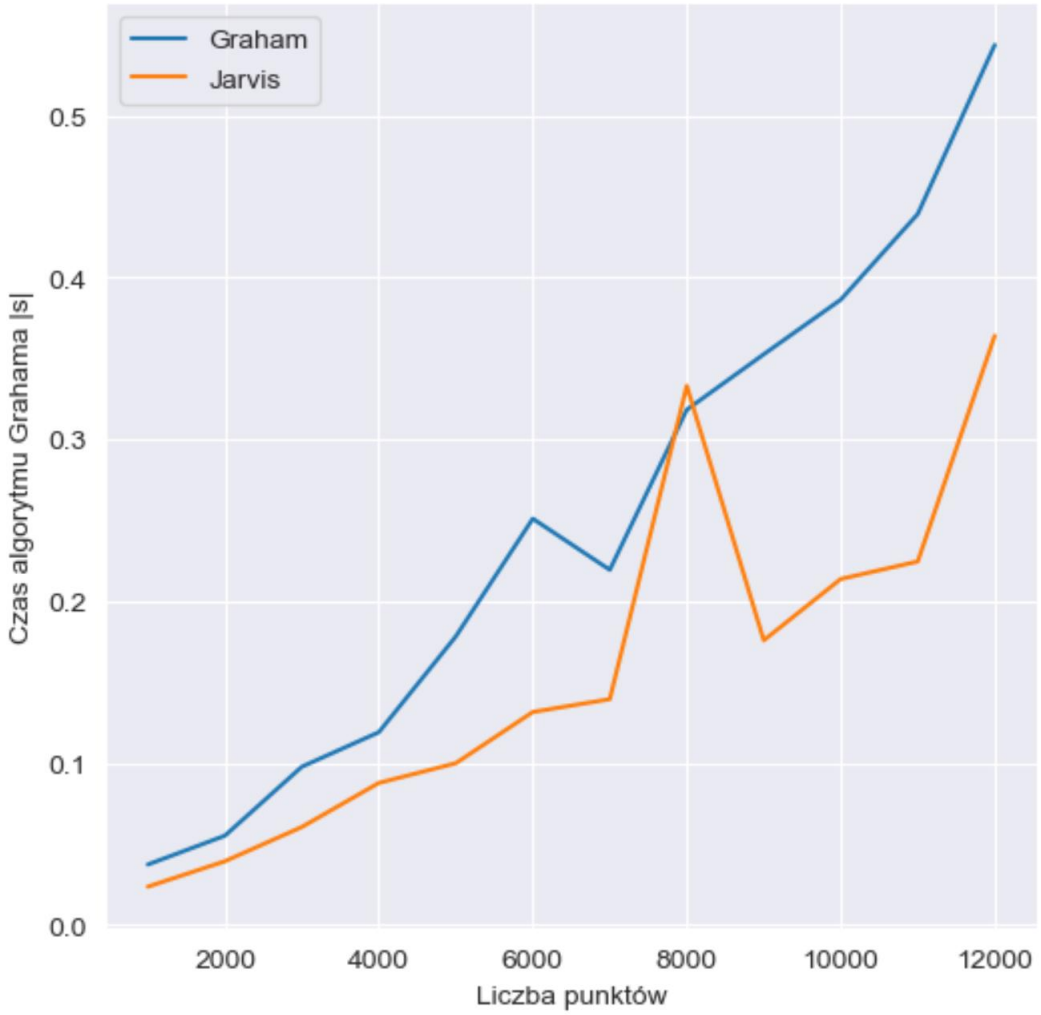


Rys. 31

c) Zbiór C, losowe punkty leżące na prostokącie

	Liczba punktów	Wierzchołki prostokątu	Czas algorytmu Grahama |s|	Czas algorytmu Jarvisa |s|	Szybszy algorytm	Róznica czasu |s|
0	1000	[(34, -36), (34, -33), (57, -36), (57, -33)]	0.037301	0.023602	Algorytm Jarvisa	0.013699
1	2000	[(48, 56), (48, -8), (0, 56), (0, -8)]	0.055114	0.039319	Algorytm Jarvisa	0.015795
2	3000	[(31, -34), (31, -37), (53, -34), (53, -37)]	0.097740	0.060512	Algorytm Jarvisa	0.037228
3	4000	[(-16, -6), (-16, 28), (-32, -6), (-32, 28)]	0.119075	0.087724	Algorytm Jarvisa	0.031350
4	5000	[(39, -42), (39, -49), (-20, -42), (-20, -49)]	0.178310	0.099919	Algorytm Jarvisa	0.078391
5	6000	[(-21, 0), (-21, -31), (66, 0), (66, -31)]	0.251059	0.131560	Algorytm Jarvisa	0.119499
6	7000	[(-18, -8), (-18, -11), (47, -8), (47, -11)]	0.219446	0.139514	Algorytm Jarvisa	0.079933
7	8000	[(-40, -24), (-40, -51), (-33, -24), (-33, -51)]	0.318416	0.333197	Algorytm Grahama	0.014782
8	9000	[(0, -27), (0, 27), (68, -27), (68, 27)]	0.352681	0.175876	Algorytm Jarvisa	0.176805
9	10000	[(-33, -45), (-33, 11), (43, -45), (43, 11)]	0.386432	0.213790	Algorytm Jarvisa	0.172642
10	11000	[(-37, -16), (-37, 62), (49, -16), (49, 62)]	0.439370	0.224608	Algorytm Jarvisa	0.214762
11	12000	[(-57, -49), (-57, -16), (13, -49), (13, -16)]	0.543880	0.364135	Algorytm Jarvisa	0.179745

Tab. 3

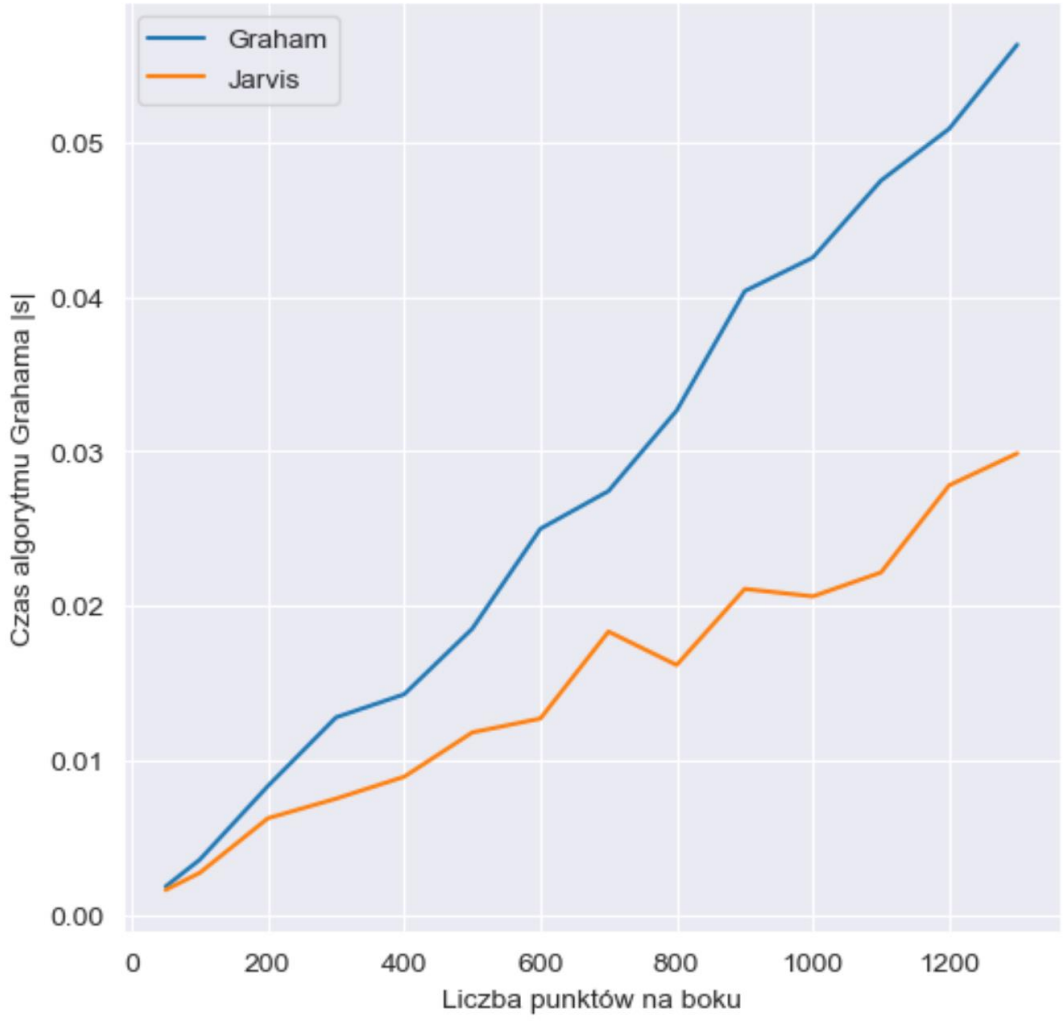


Rys. 32

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

	Ilość punktów na boku	Liczba punktów na przekątnej	Wierzchołki	Czas algorytmu Grahama |s|	Czas algorytmu Jarvisa |s|	Szybszy algorytm	Róznica czasu |s|
0	50	25	[(-31, 21), (-31, 75), (23, 75), (23, 21)]	0.001541	0.001346	Algorytm Jarvisa	0.000195
1	100	50	[(-17, -9), (-17, 4), (-4, 4), (-4, -9)]	0.003777	0.003729	Algorytm Jarvisa	0.000049
2	200	100	[(-7, -30), (-7, 11), (34, 11), (34, -30)]	0.009355	0.005415	Algorytm Jarvisa	0.003940
3	300	150	[(1, 34), (1, 102), (69, 102), (69, 34)]	0.013558	0.013874	Algorytm Grahama	0.000316
4	400	200	[(-3, -47), (-3, -6), (38, -6), (38, -47)]	0.017576	0.012023	Algorytm Jarvisa	0.005554
5	500	250	[(31, 17), (31, 56), (70, 56), (70, 17)]	0.024190	0.016742	Algorytm Jarvisa	0.007448
6	600	300	[(0, 2), (0, 87), (85, 87), (85, 2)]	0.030938	0.015627	Algorytm Jarvisa	0.015311
7	700	350	[(34, -36), (34, -6), (64, -6), (64, -36)]	0.040092	0.022559	Algorytm Jarvisa	0.017534
8	800	400	[(-5, -12), (-5, 30), (37, 30), (37, -12)]	0.046373	0.024331	Algorytm Jarvisa	0.022042
9	900	450	[(-5, -29), (-5, 17), (41, 17), (41, -29)]	0.050309	0.023707	Algorytm Jarvisa	0.026602
10	1000	500	[(-16, -13), (-16, 28), (25, 28), (25, -13)]	0.069850	0.028427	Algorytm Jarvisa	0.041423
11	1100	550	[(0, 12), (0, 74), (62, 74), (62, 12)]	0.063724	0.040420	Algorytm Jarvisa	0.023304
12	1200	600	[(-33, 32), (-33, 79), (14, 79), (14, 32)]	0.078960	0.037934	Algorytm Jarvisa	0.041027
13	1300	650	[(-12, -5), (-12, 61), (54, 61), (54, -5)]	0.086891	0.050540	Algorytm Jarvisa	0.036351

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.