

Algorytmy Geometryczne

Zadanie 2, laboratorium 2 - sprawozdanie

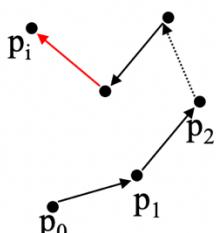
Radosław Kawa

1. Opis ćwiczenia

Zadaniem na laboratorium 2 było wyznaczenie otoczki wypukłej dla zadanego zbioru punktów. Do wyznaczenia otoczki wykorzystane były 2 algorytmy Grahama oraz Jarvisa:

Algorytm Grahama

1. W zbiorze S wybieramy punkt p_0 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 (p_0, p) z dodatnim kierunkiem osi x . Jeśli kilka punktów tworzy ten sam kąt, usuwamy wszystkie z wyjątkiem najbardziej oddalonego od p_0 . Niech uzyskanym ciągiem będzie p_1, p_2, \dots, p_m .
3. Do początkowo pustego stosu s wkładamy punkty p_0, p_1, p_2 .
 $t = \text{indeks stosu}; i \leftarrow 3$
4. **while** $i < m$ **do**
 if p_i leży na lewo od prostej (p_{t-1}, p_t)
 then push(p_i), $i \leftarrow i+1$
 else pop(s)



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 krok4

Algorytm Jarvisa

znajdź punkt i_0 z S o najmniejszej współrzędnej y ; $i \leftarrow i_0$

repeat

for $j \neq i$ **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

$i \leftarrow k$

until $i = i_0$

Koszt algorytmu:

Złożoność rzędu $O(n^2)$. Gdy liczba wierzchołków otoczki jest ograniczona przez stałą k , jego złożoność jest rzędu $O(kn)$.

2. Biblioteki oraz specyfikacja

Zadanie było wykonane w jupyter notebook, w języku Python. Wersja 3.10.0. Do obliczeń wykorzystałem bibliotekę numpy, aby uzyskać dokładniejsze wartości liczb i szybciej je generować. Do rysowania wykresów

seaborna oraz bibliotekę matplotlib tam gdzie to konieczne. Dane gromadze w dataframie, biblioteki pandas, aby było to bardziej przejrzyste przy wyświetlaniu

Specyfikacja:

System operacyjny: macOS Monterey

Procesor: Apple M1

Pamięć: 8GB

3. Algorytmy

Nazewnictwo:

generate_random_points() – algorytm generowania zbioru a)

generate_random_circle() – algorytm generowania zbioru b)

generate_random_rectangle() – algorytm generowania zbioru c)

generate_random_square() – algorytm generowania zbioru d)

Jarvis – algorytm Jarvisa (owijanie prezentu)

Graham – algorytm grahama

4. Wygenerowane zbiory

a) 100 losowo wygenerowanych punktów o współrzędnych z przedziału [-100, 100],

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

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

d) 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.

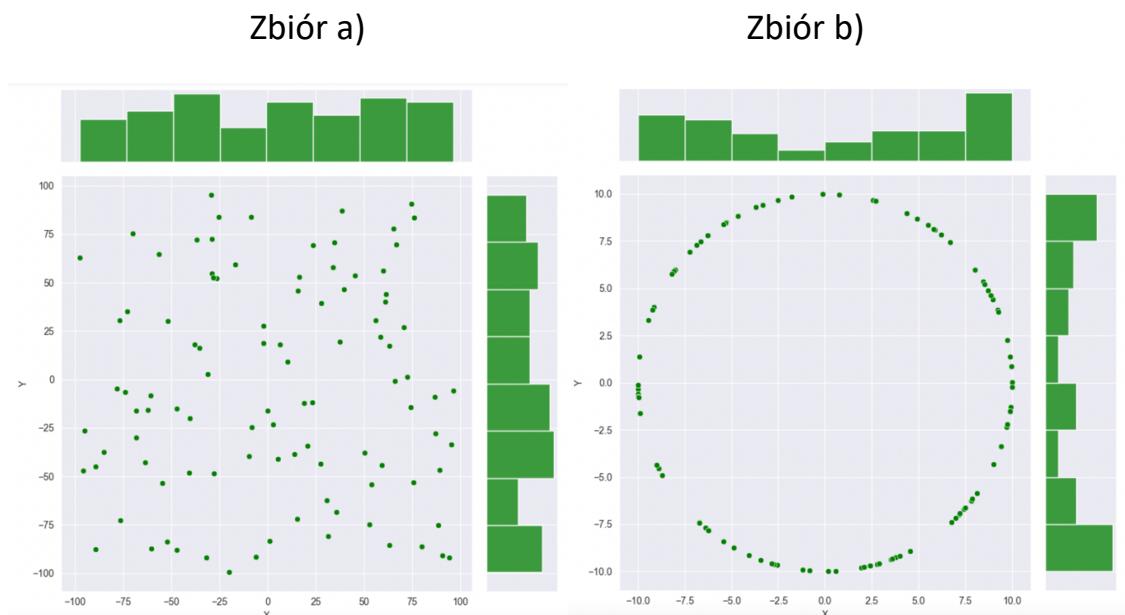
Do wygenerowania punktów wykorzystałem z biblioteki numpy numpy.random.uniform aby mieć bardziej dokładniejsze te punkty oraz było to szybsze, dodatkowo przechowuje je w dataframie, dla lepszej wizualizacji oraz wygody.

Przykładowo dla punktu a) oraz b) pierwsze parę wyników:

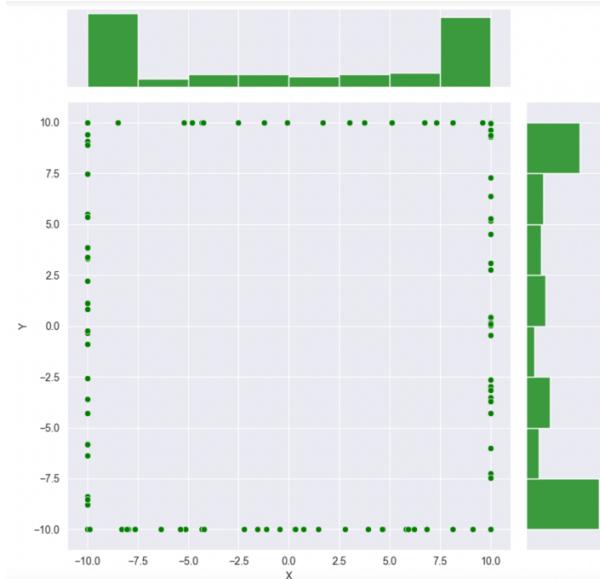
	X	Y		X	Y
0	70.592423	26.838248	0	-8.901340	-4.556989
1	23.279086	-11.763070	1	-9.994568	-0.329565
2	-56.410022	64.553248	2	-5.416031	-8.406343
3	94.072933	-91.869119	3	-6.390499	-7.691653
4	-94.902405	-26.333176	4	0.575527	-9.983425

Wykresy zbiorów:

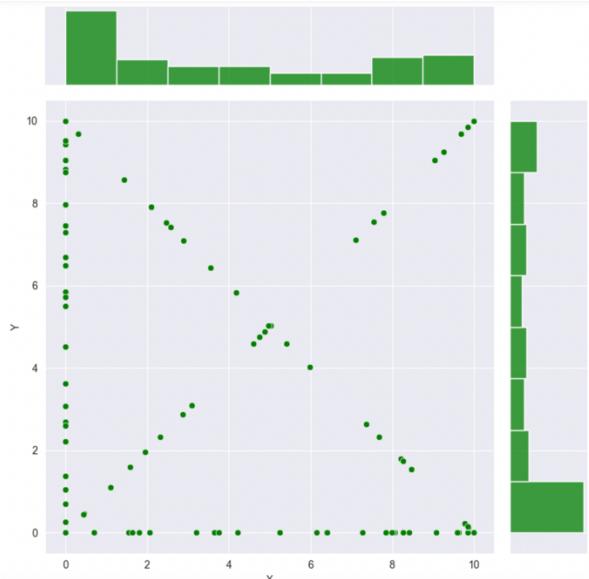
Wykresy (osie) zostały dodatkowo podpisane, zbiory punktów zostały pokazane za pomocą biblioteki seaborn z histogramem



Zbiór c)



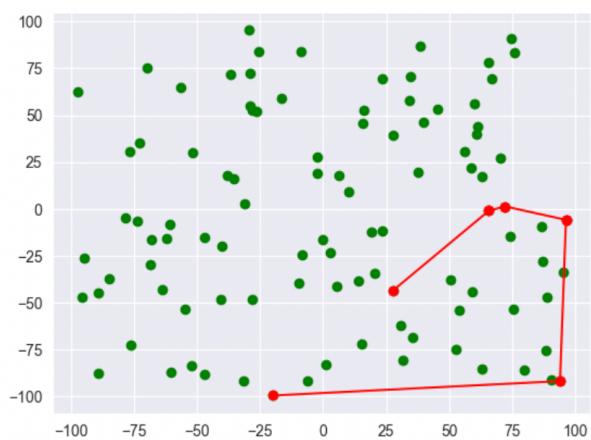
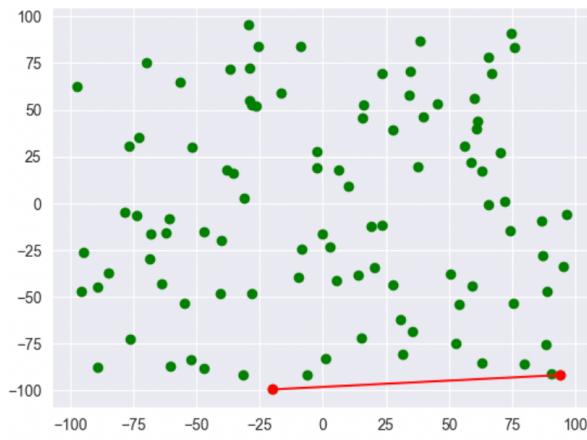
Zbiór d)

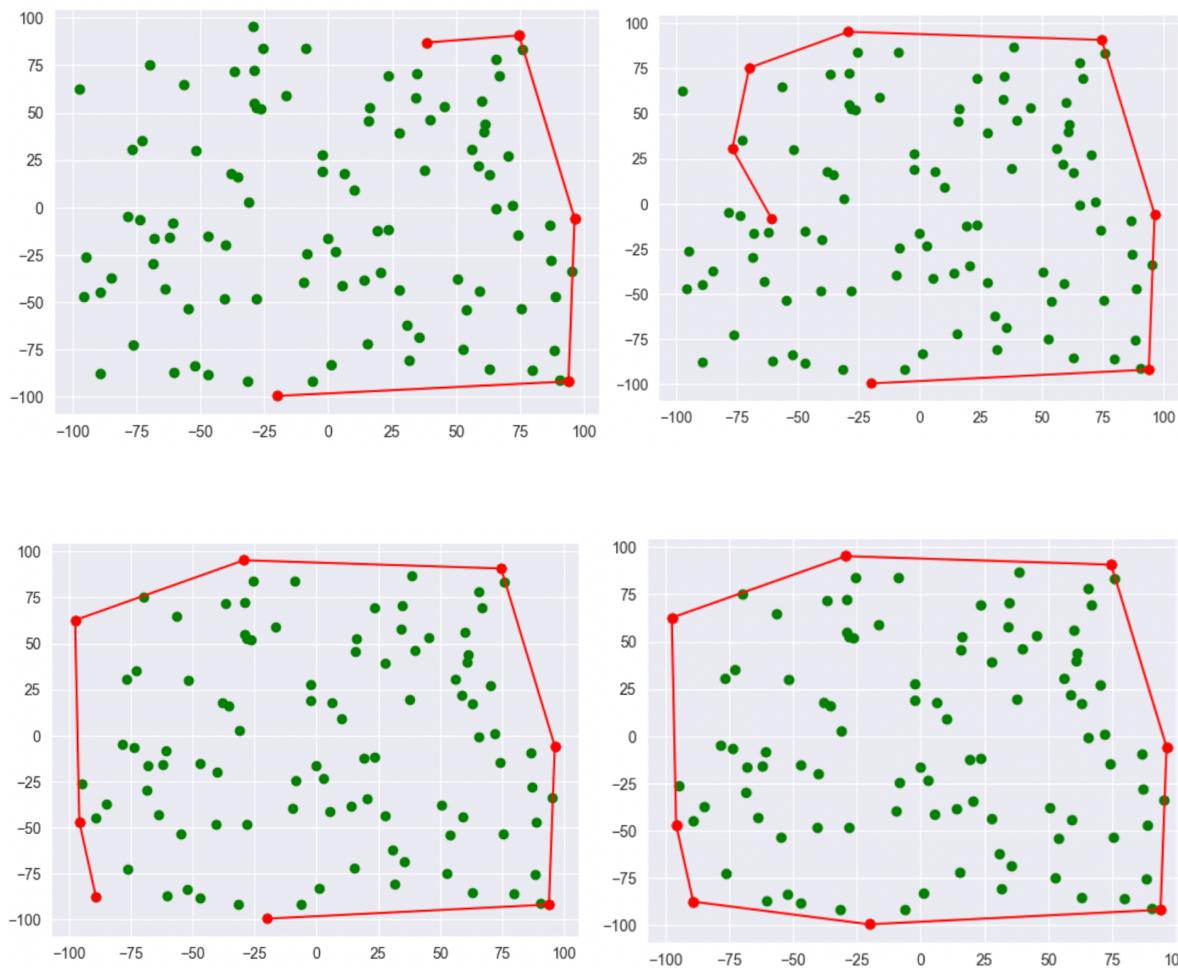


5. Wizualizacja algorytmu Grahama

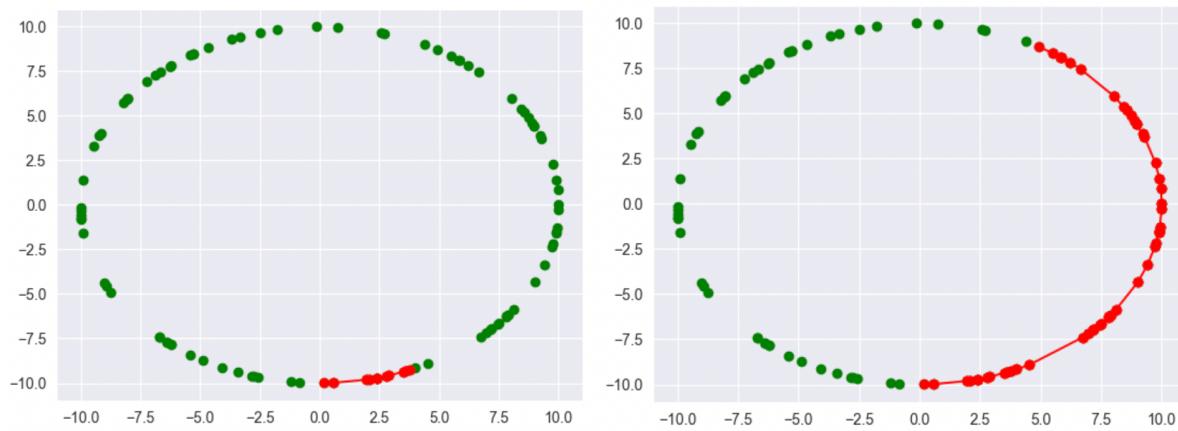
Do punktów został zastosowany epsilon – 10^{-12}

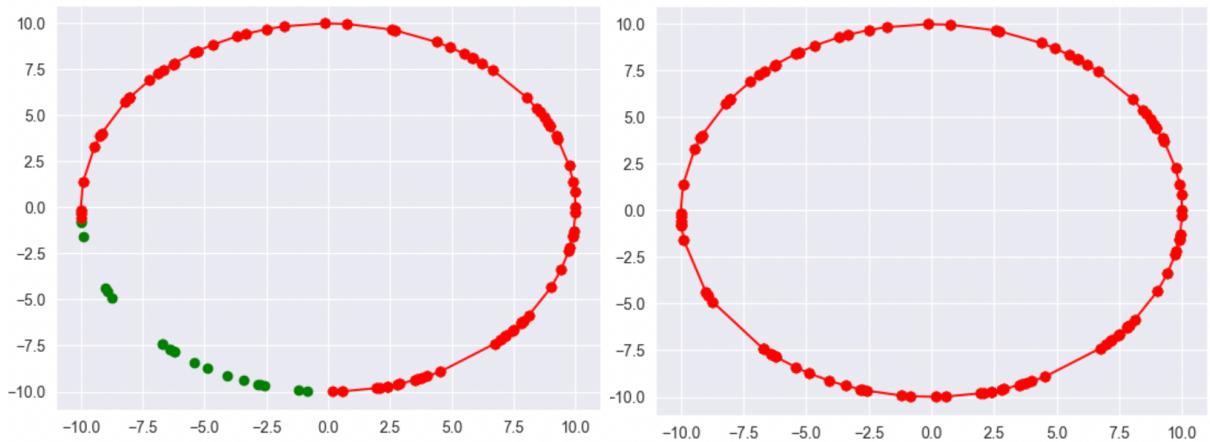
Zbiór a): ilość punktów = 100, zakres = [-100,100]



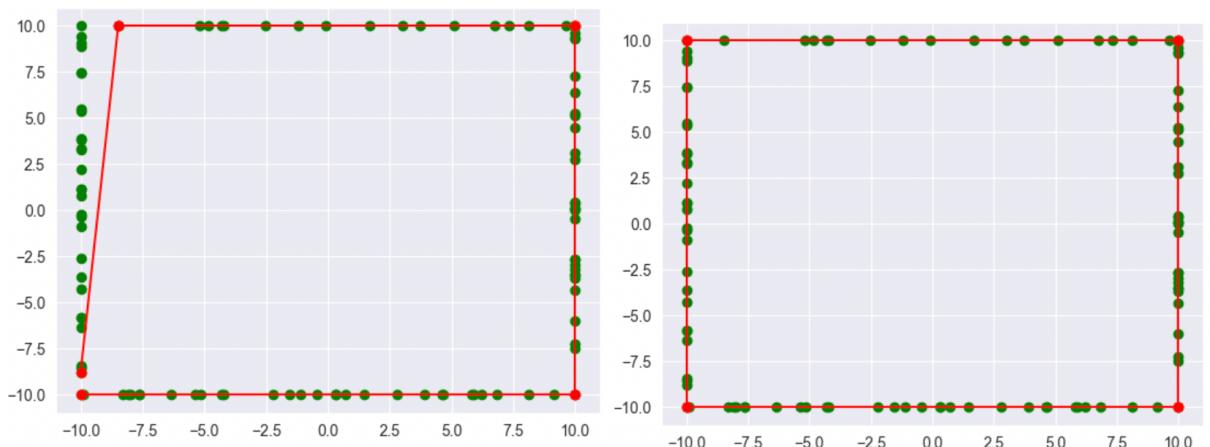
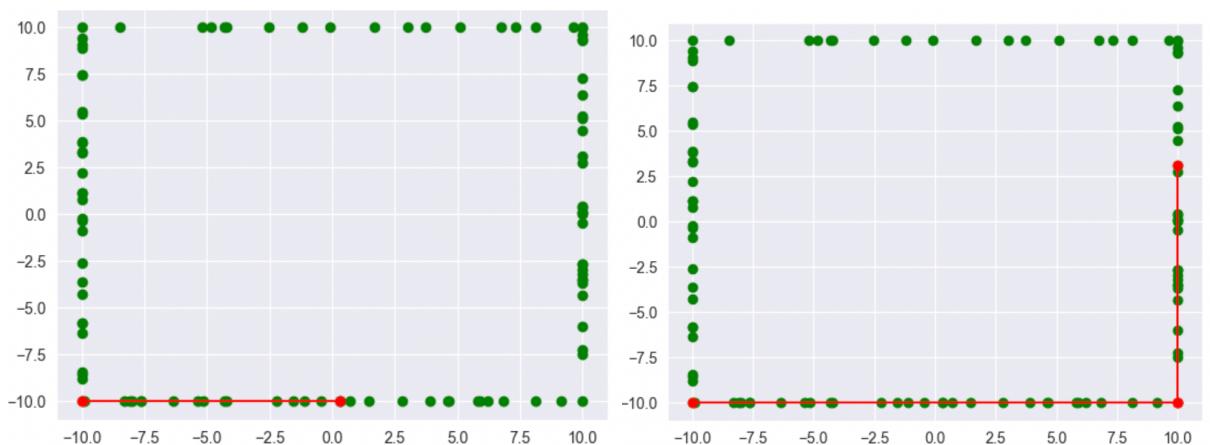


Zbiór b): ilość punktów = 100, środek = (0,0), R = 10

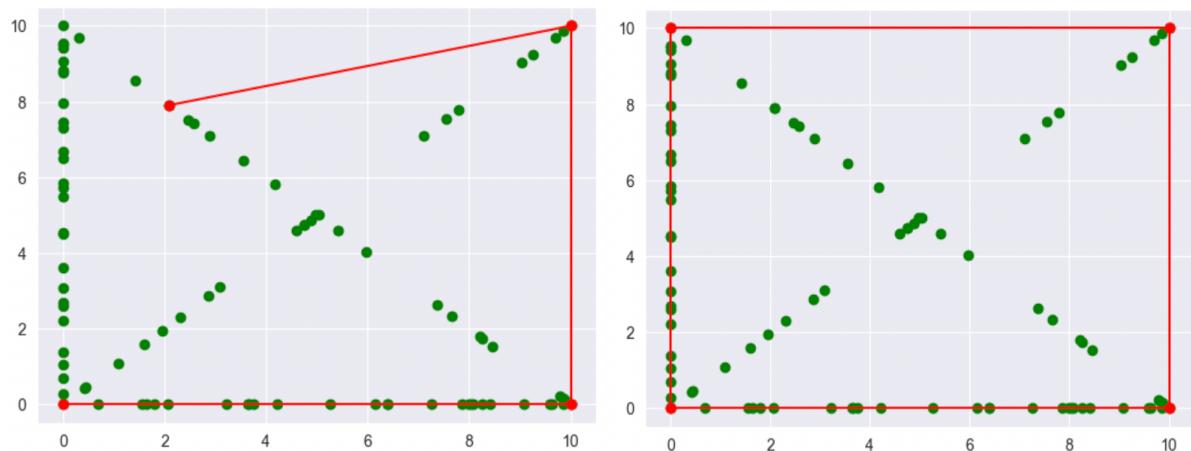
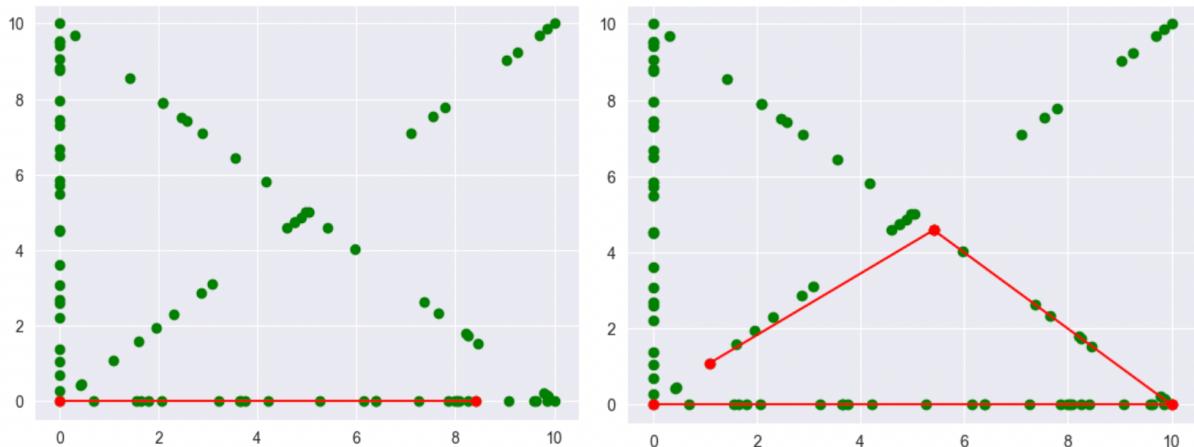




Zbiór c): ilość punktów = 100, wierzchołki = (-10, 10), (-10,-10), (10,-10), (10,10)

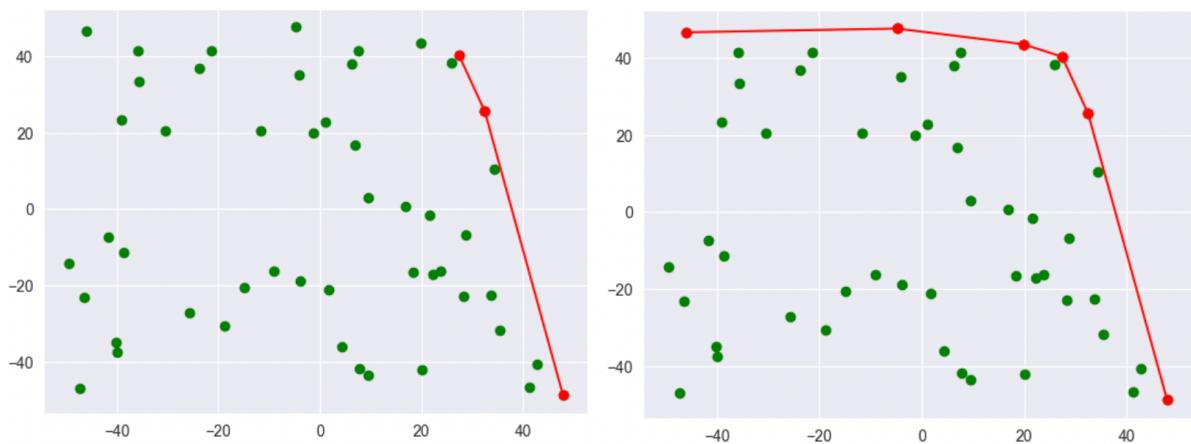


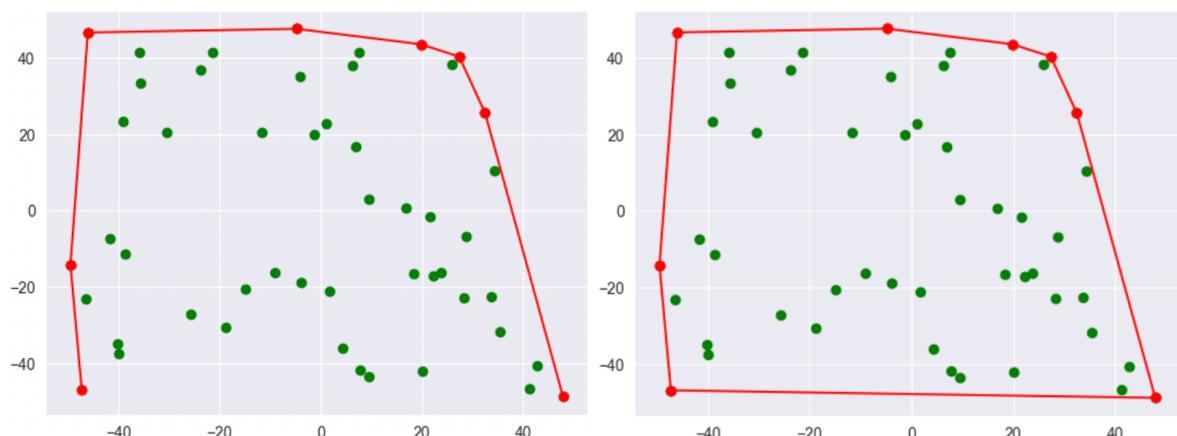
Zbiór d): boki = 25, przekątne = 20, wierzchołki = (0, 0), (10, 0), (10, 10), (0, 10)



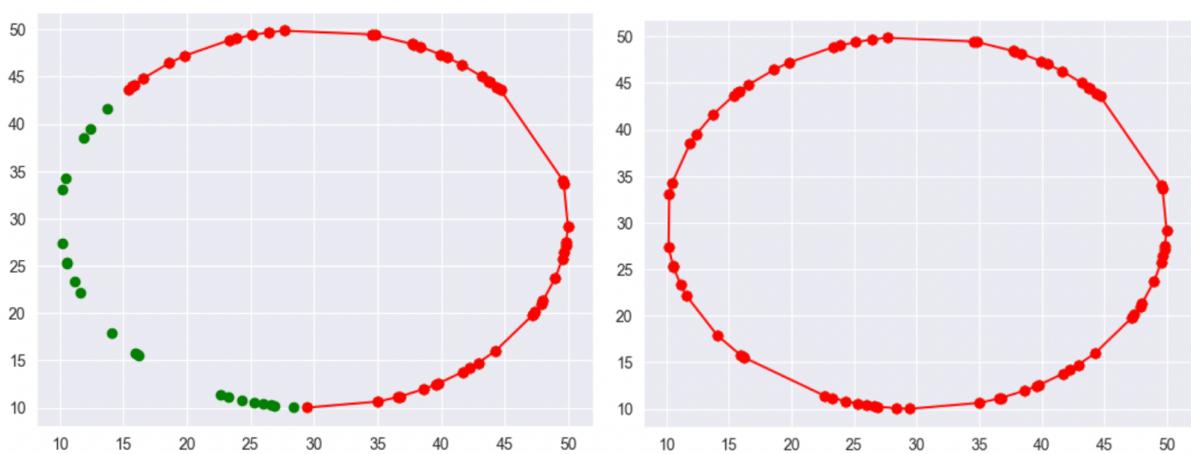
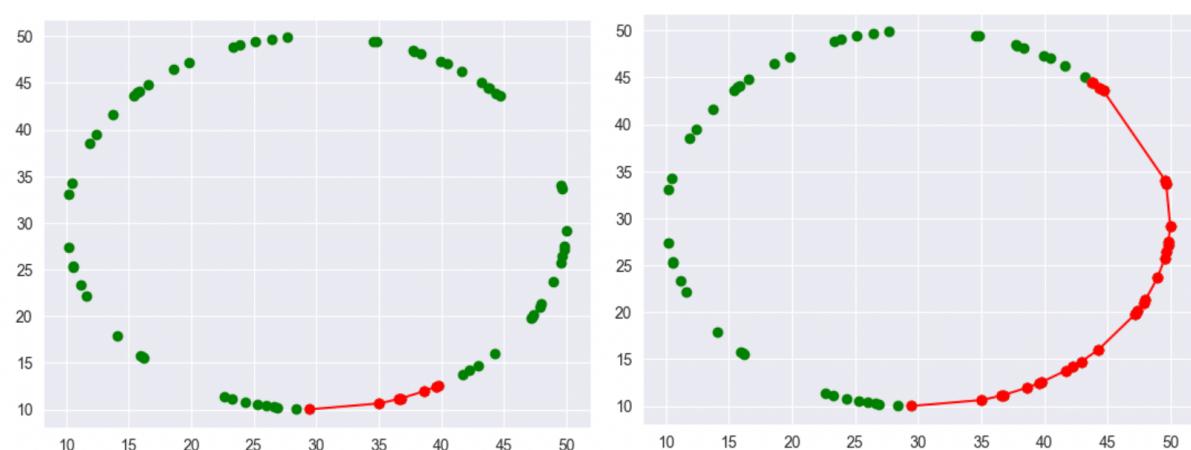
6. Wizualizacja algorytmu Jarvisa

Zbiór a): ilość punktów = 50, zakres = [-50,50]

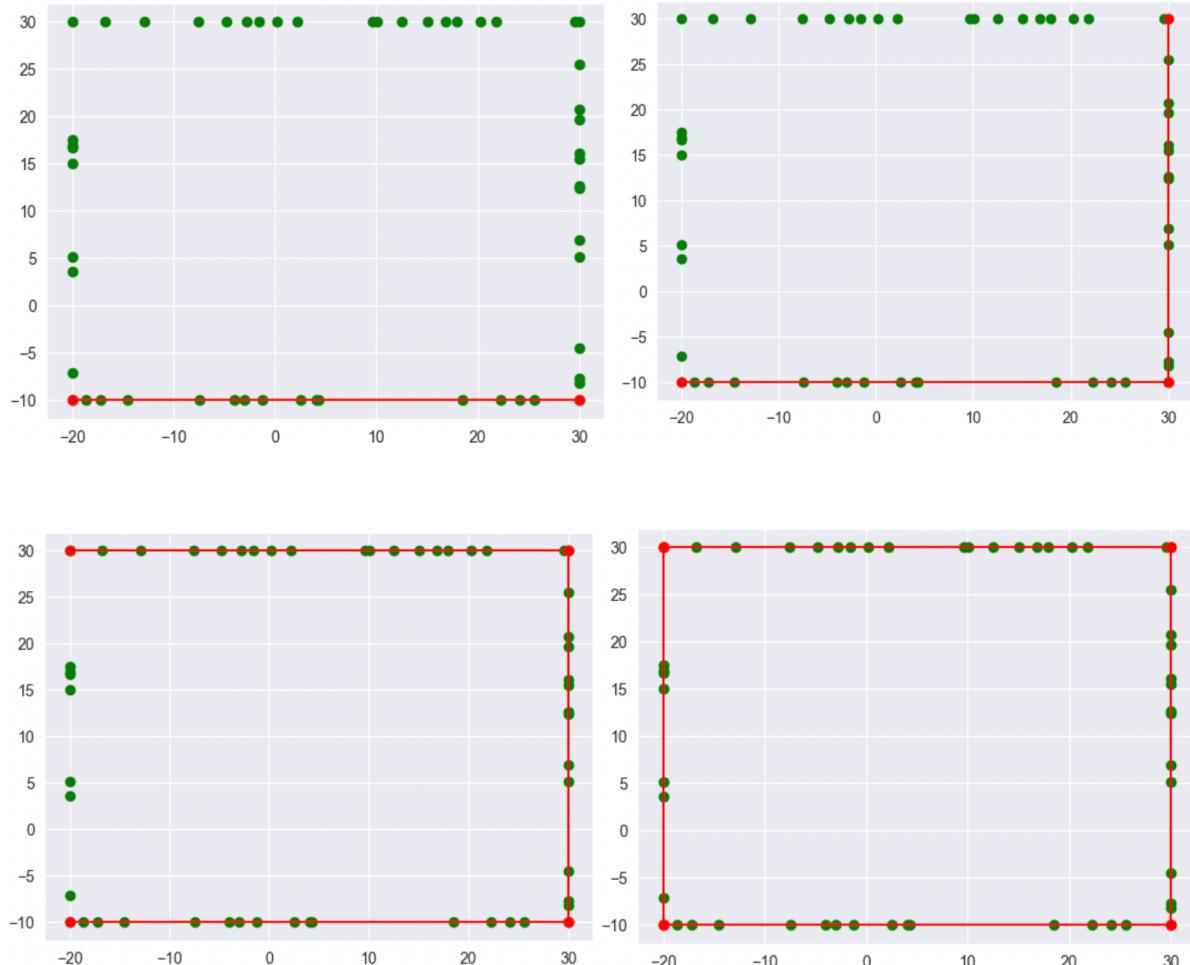




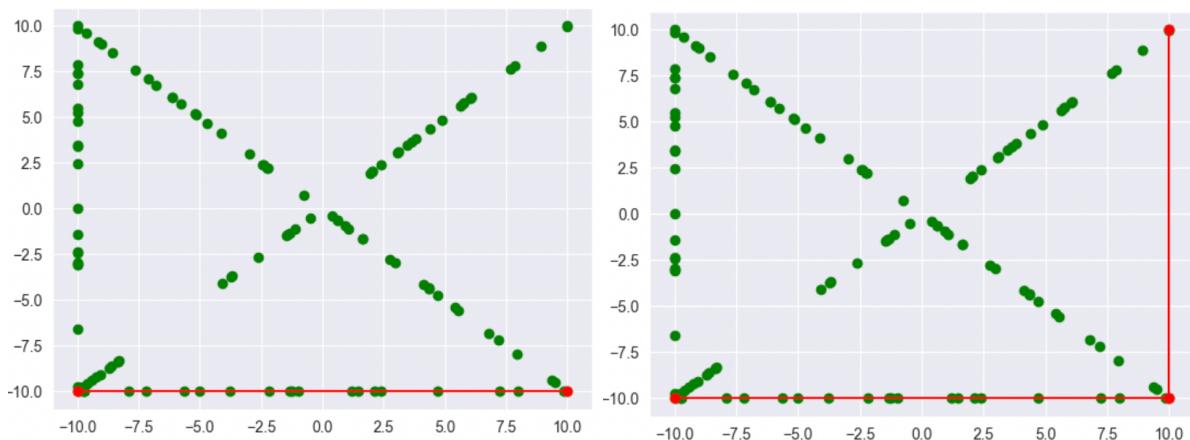
Zbiór b): ilość punktów = 70, środek = (30,30), R = 20

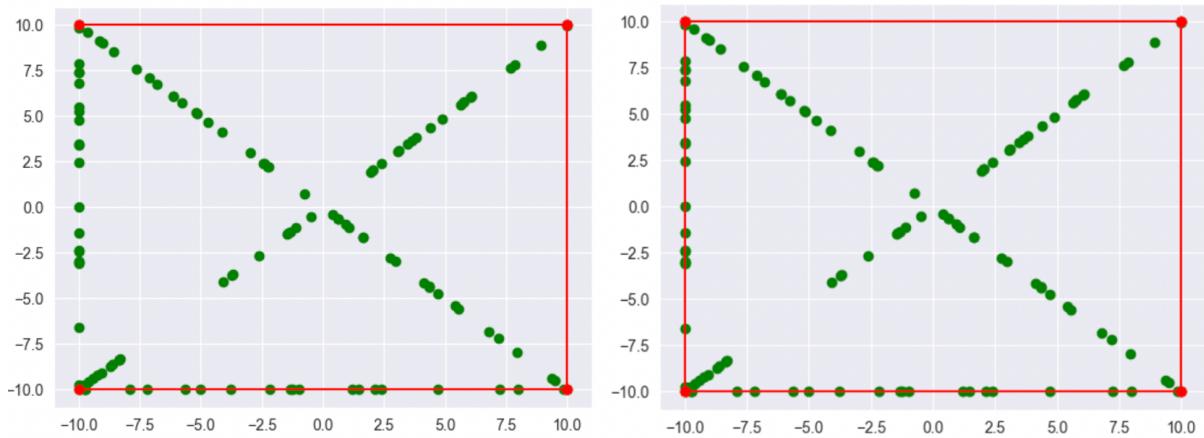


Zbiór c): ilość punktów = 50, wierzchołki = (-20, -10), (-20,30), (30,30), (30,-10)



Zbiór d): boki = 50, przekątne = 40, wierzchołki = (-10, -10), (10, -10), (10, 10), (-10, 10)





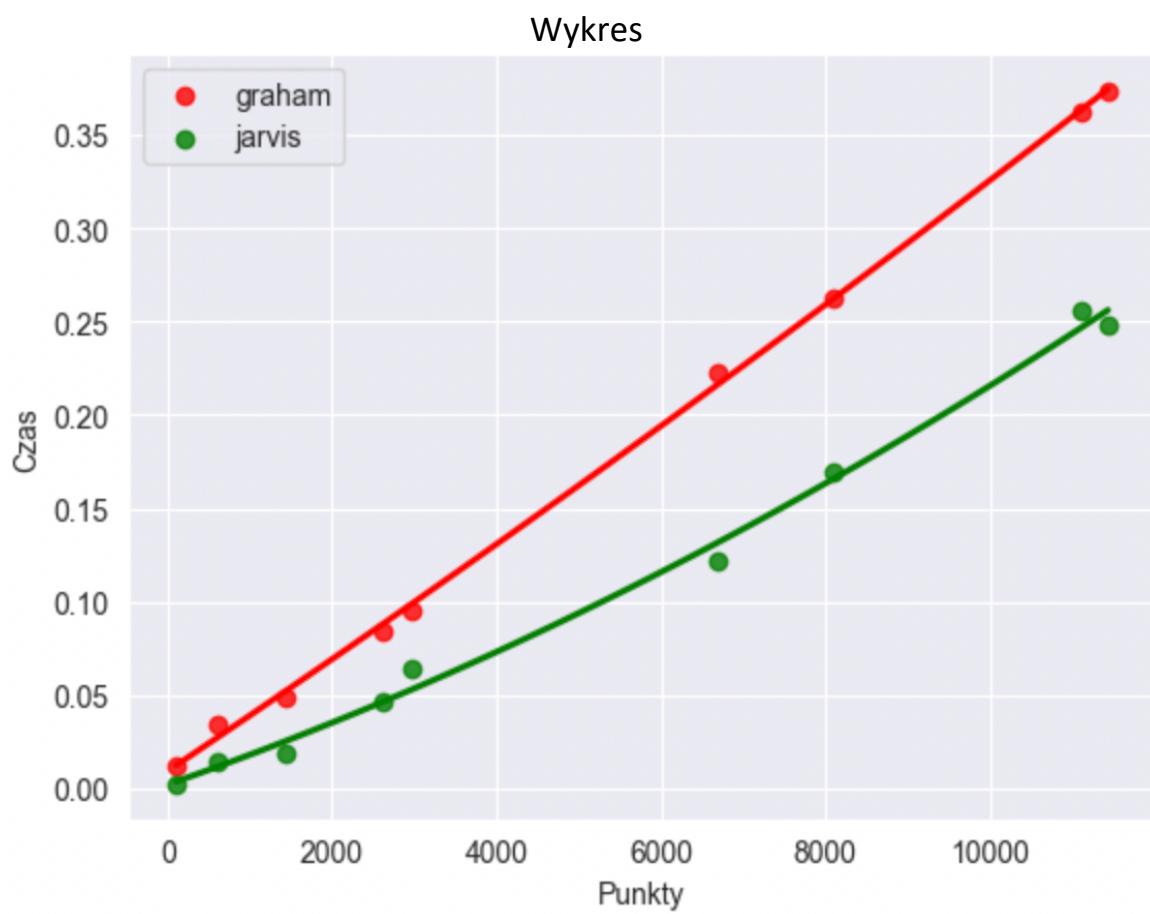
7. Porównanie czasów

Poniżej zamieszczone są wykresy, z czasami działania algorytmów Jarvisa oraz Grahama w zależności od ilości punktów, oraz czasu. Czas jest podany w [s]
Do mierzenia czasu użyto funkcji timeit.timeit

Losowe punkty - Zbiór a):

Tabela

Punkty	Zakres a	Zakres b	Czas grahama	Czas jarvisa	Szybszy	Porównanie czasów	
0	104	-200	200	0.011974	0.002269	jarvis	0.009705
1	610	-200	200	0.034674	0.014170	jarvis	0.020504
2	1431	-200	200	0.048551	0.018539	jarvis	0.030012
3	2612	-200	200	0.083732	0.046291	jarvis	0.037441
4	2975	-200	200	0.095439	0.063684	jarvis	0.031755
5	6690	-200	200	0.222559	0.121889	jarvis	0.100669
6	8106	-200	200	0.262622	0.169580	jarvis	0.093042
7	11432	-200	200	0.373896	0.247948	jarvis	0.125948
8	11097	-200	200	0.362113	0.256284	jarvis	0.105829

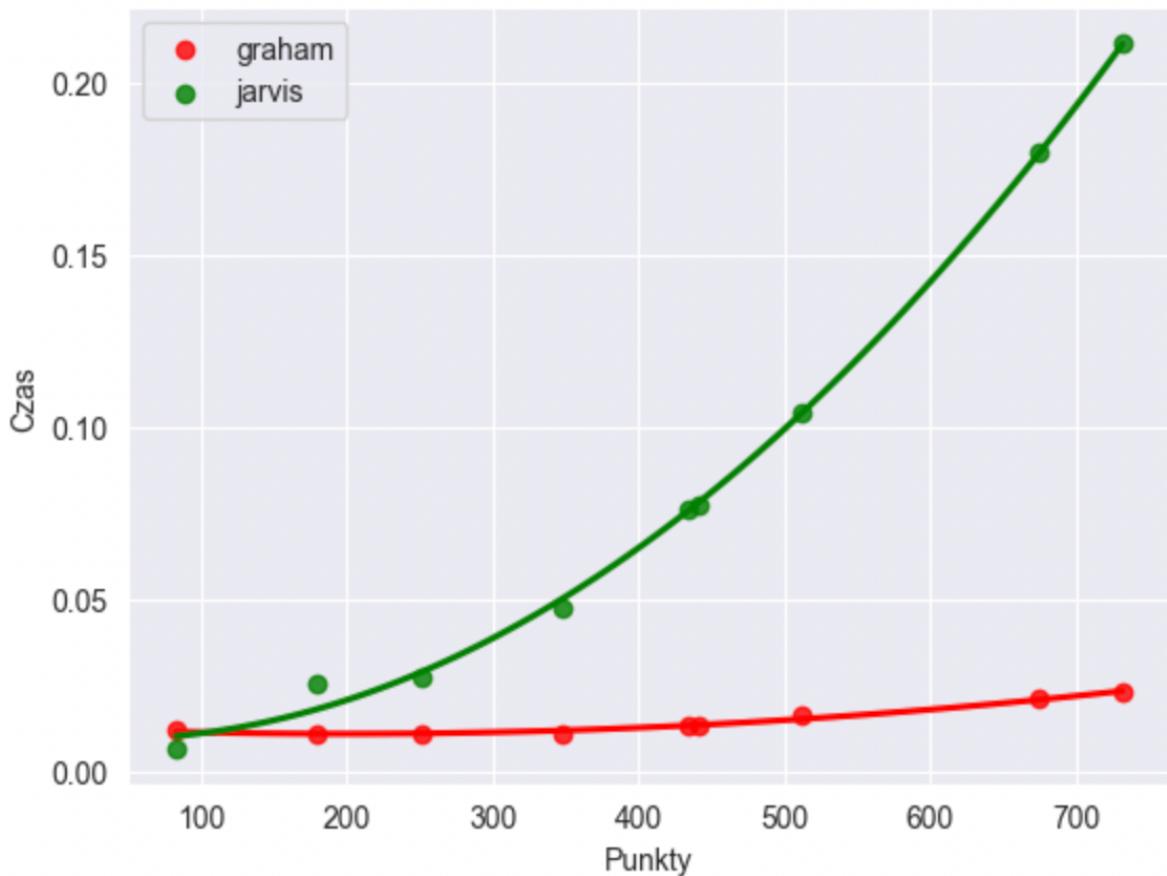


Okrąg - Zbiór b):

Tabela

Punkty	Środek	R	Czas grahama	Czas jarvisa	Szybszy	Porównanie czasów	
0	83	[-3, -7]	2	0.011840	0.006505	jarvis	0.005336
1	179	[3, -5]	7	0.010527	0.025232	graham	0.014706
2	252	[-4, -6]	7	0.010822	0.027441	graham	0.016618
3	348	[-10, 0]	3	0.010713	0.047208	graham	0.036495
4	434	[4, 2]	8	0.013230	0.076075	graham	0.062845
5	441	[-2, -4]	4	0.013373	0.077479	graham	0.064105
6	512	[-6, -8]	8	0.016318	0.103751	graham	0.087433
7	675	[9, 4]	3	0.020974	0.179466	graham	0.158492
8	732	[3, 0]	2	0.022626	0.211364	graham	0.188738

Wykres

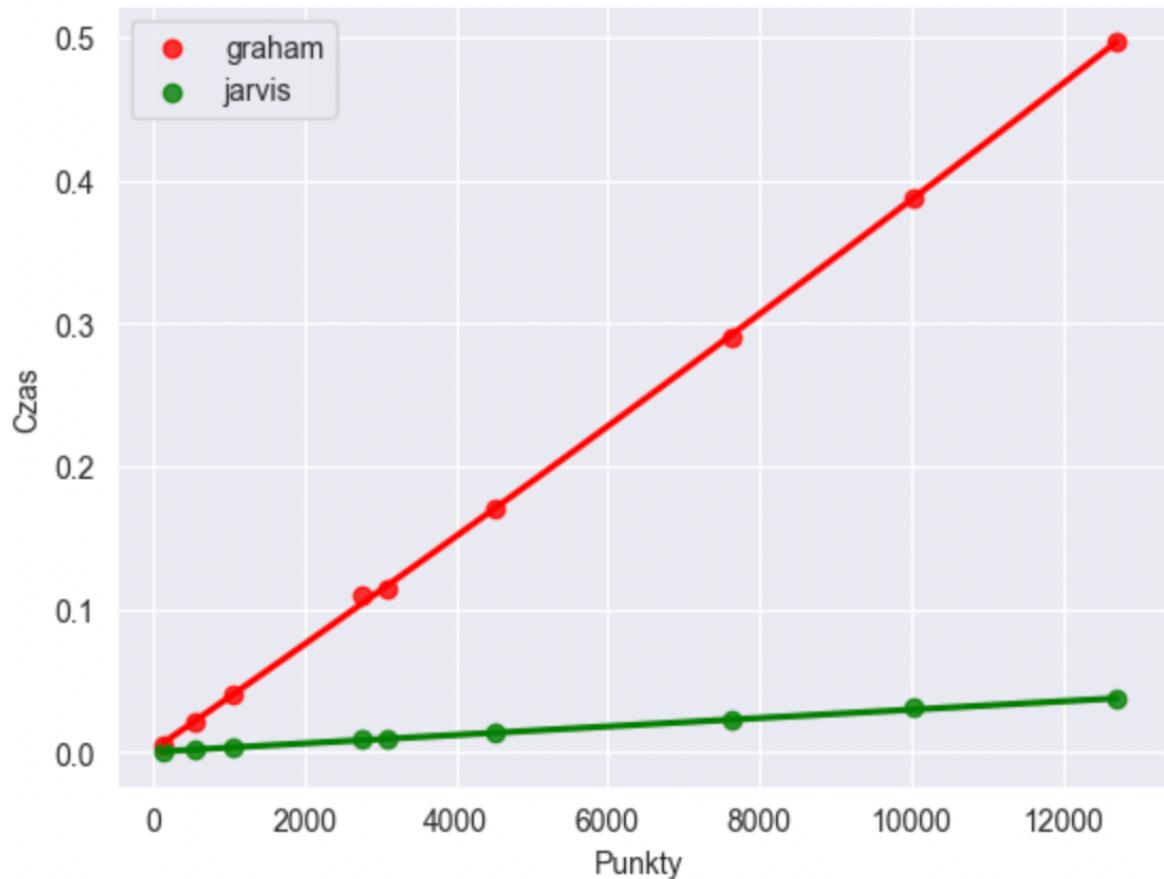


Prostokąt - Zbiór c)

Tabela

	Punkty	Rogi	Czas grahama	Czas jarvisa	Szybszy	Porównanie czasów
0	142	[(-10, 23), (-10, 17), (-4, 23), (-4, 17)]	0.005110	0.000728	jarvis	0.004382
1	556	[(10, 29), (10, 17), (-20, 29), (-20, 17)]	0.021401	0.001770	jarvis	0.019631
2	1048	[(-17, -19), (-17, 7), (-6, -19), (-6, 7)]	0.040446	0.003296	jarvis	0.037151
3	2756	[(-1, 5), (-1, -1), (-11, 5), (-11, -1)]	0.109240	0.008852	jarvis	0.100389
4	3094	[(-16, -14), (-16, 3), (-13, -14), (-13, 3)]	0.114177	0.008984	jarvis	0.105193
5	4498	[(16, 17), (16, 25), (26, 17), (26, 25)]	0.169731	0.013204	jarvis	0.156527
6	7641	[(-8, 0), (-8, 12), (-5, 0), (-5, 12)]	0.290467	0.022123	jarvis	0.268344
7	12708	[(21, 29), (21, 18), (-8, 29), (-8, 18)]	0.497220	0.037111	jarvis	0.460109
8	10039	[(28, -26), (28, -7), (8, -26), (8, -7)]	0.388465	0.031133	jarvis	0.357332

Wykres

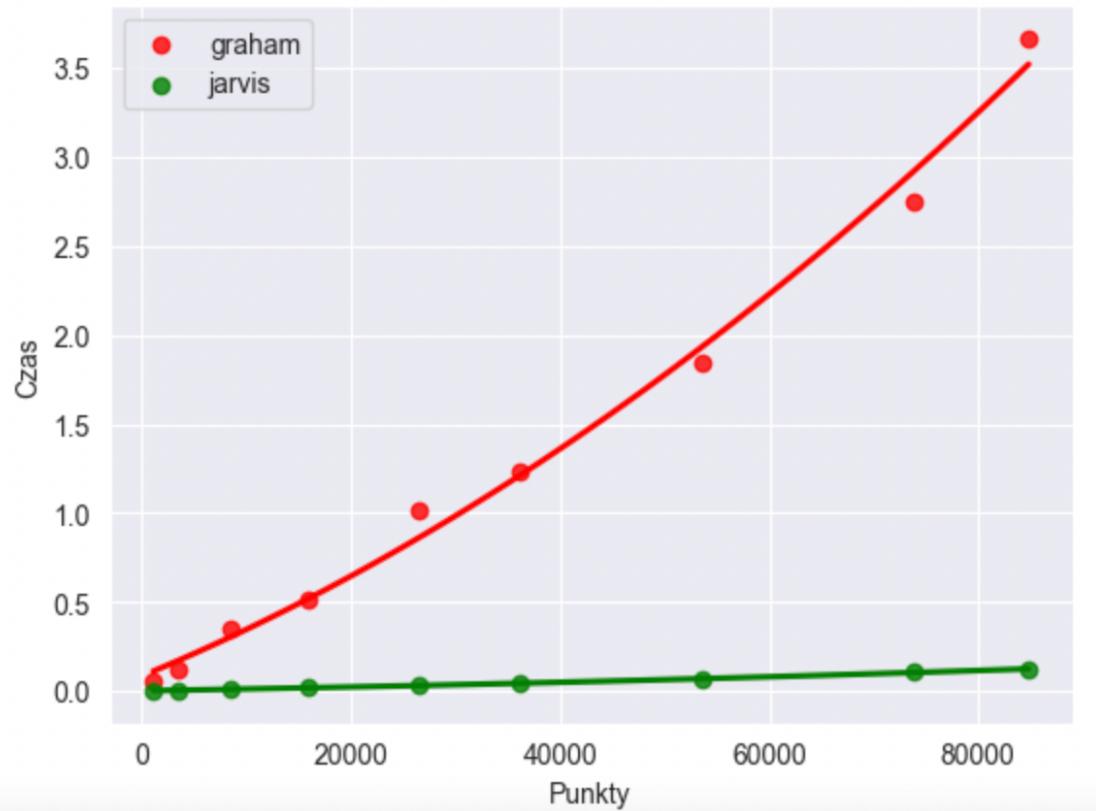


Kwadrat – Zbiór d)

Tabela

	Razem	Boki	Przekątne	Rogi	Czas grahama	Czas jarvisa	Szybszy	Porównanie czasów
0	1073	537	532	[(−2, 11), (16, 11), (−2, 29), (16, 29)]	0.058847	0.001668	jarvis	0.057180
1	3500	1738	1758	[(-11, 28), (-26, 28), (-11, 43), (-26, 43)]	0.122993	0.004421	jarvis	0.118572
2	8533	3828	4701	[(-10, 22), (-5, 22), (-10, 27), (-5, 27)]	0.347784	0.011609	jarvis	0.336175
3	15904	8704	7196	[(22, -2), (-28, -2), (22, 48), (-28, 48)]	0.513934	0.017862	jarvis	0.496072
4	26464	12935	13525	[(28, -15), (4, -15), (28, 9), (4, 9)]	1.017350	0.033829	jarvis	0.983521
5	36064	20616	15444	[(2, -15), (25, -15), (2, 8), (25, 8)]	1.229639	0.042451	jarvis	1.187189
6	53498	28287	25207	[(4, 26), (19, 26), (4, 41), (19, 41)]	1.845279	0.063304	jarvis	1.781975
7	73892	37088	36800	[(14, -6), (-22, -6), (14, 30), (-22, 30)]	2.753435	0.113354	jarvis	2.640081
8	84802	36774	48024	[(−3, −27), (3, −27), (−3, −21), (3, −21)]	3.665733	0.119825	jarvis	3.545908

Wykres



7. Wnioski

Nie w każdym przypadku algorytm grahama okazał się szybszy, co jest związane ze złożonością tych algorytmów. Można zauważyć, że algorytm Grahama dużo lepiej działa na punktach wygenerowanych w taki sposób, że większość należy do otoczki, tak jak to jest przy okręgu (w porównaniu do Jarvisa). Możemy zaobserwować złożoności zgodne z oczekiwanymi Graham – $O(n\log n)$, Jarvis $O(n^*k)$. Dla pozostałych wykresów ze zbiorami c) i d) Jarvis powinien radzić sobie znacznie lepiej co widać w porównaniu z Grahamem, gdyż złożoność wynosi $O(4^*n)$ dla tych dwóch zbiorów (4 punkty w otoczce). Czasowo bardzo podobnie wypadają oba algorytmy w zbiorach c) i d). Po wygenerowaniu regresji wielomianowej dla tych czasów, możemy stwierdzić, że Jarvis dla podpunktu b) jest $O(n^2)$ dla reszty blisko liniowej. Zaś Graham stale ma złożoność $O(n\log n)$. Dodatkowo dla podpunktu a) jarvis również okazał się szybszy, prawdopodobnie bardzo mało punktów zalicza do otoczki, przez co Graham jest trochę gorszy czasowo, ale niewiele. Pojawił się pojedynczy przypadek w zbiorze b) gdzie Jarvis jest szybszy. Możemy stwierdzić, że algorytm Jarvisa nadaje się najbardziej do punktów wygenerowanych w taki sposób, że jest mało ich w otoczce, zaś Grahama do ilości punktów w otoczce bliżej wygenerowanych punktów.