Szymon Budziak
Informatyka II rok, grupa poniedziałek 14:40 - 16:10

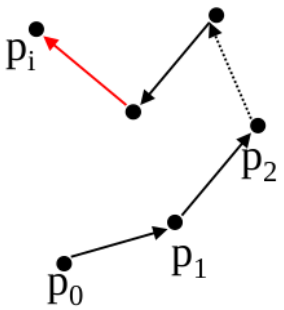
Algorytmy geometryczne, laboratorium 2 - sprawozdanie

1.Opis ćwiczenia

Zadaniem które należy wykonać na laboratorium 2 jest wyznaczenie otoczki wypukłej dla danego zbioru punktów, czyli najmniejszy zbiór wypukły zawierający podzbiór wszystkich punktów na płaszczyźnie. Do wyznaczenia otoczki zostały użyte dwa algorytmy: algorytm Grahama oraz algorytm Jarvisa.

Algorytm Grahama działa w następujący sposób:

- 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,
- 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, …, p_m,
- Do początkowo pustego stosu s wkładamy punkty p_0, p_1, p_2. t – indeks stosu; i←3
- **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)



Każdy punkt, który został usunięty ze stosu, nie należy do otoczki wypukłej. Zbiór punktów na stosie tworzy wielokąt wypukły.

Koszt algorytmu:

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

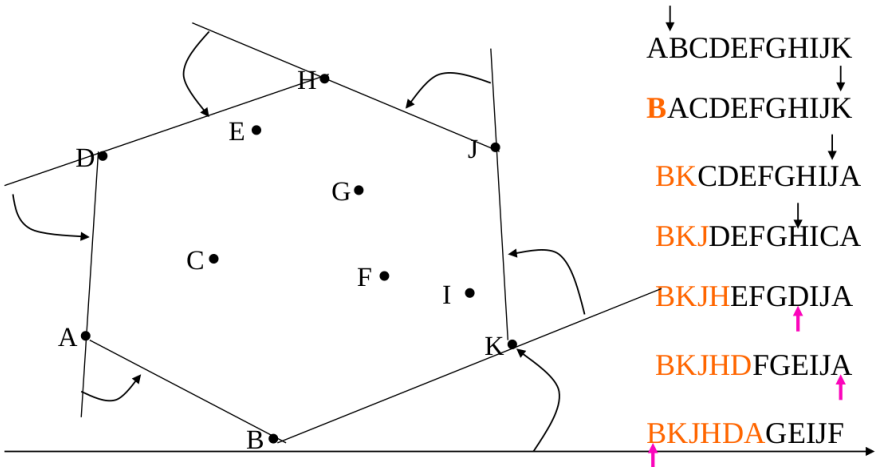
O(n) + O(nlogn) + O(1) + O(n-3) = O(nlogn)

szukanie minimum | sortowanie | inicjalizacja stosu

Algorytm Jarvis (owijanie prezentu) działa w następujący sposób:

- znajdź punkt i_0 z S o najmniejszej współrzędnej y; i←i_0,
- **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 = 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. Środowisko, biblioteki oraz użyte narzędzia

Ćwiczenie zostało wykonane w Jupyter Notebook i napisane w języku Python. Dodatkowo zostały użyte biblioteki takie jak pandas oraz numpy aby w przejrzysty sposób w dataframe przedstawiać wyniki z poszczególnych obliczeń. Do rysowania wykresów zostały użyte biblioteki matplotlib oraz seaborn, które w świetny sposób obrazują wykresy z danych które są w dataframe. Do porównania czasów działania algorytmów została użyta funkcja z biblioteki time - perf_counter. Wszystko było wykonywane na systemie operacyjnym Linux Ubuntu 20.04 oraz na procesorze Intel Core i5-7300HQ 2.50GHz.

3. Plan i sposób wykonania ćwiczenia

Na początku należało wygenerować zbiory punktów o współrzędnych rzeczywistych typu double:

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.

5 pierwszych wierszy z dataframe kolejno z podpunktu a), b), c), d):

a) Tabela_1

	X	Y
0	15.387291	5.425735
1	-71.297262	-70.517711
2	95.367173	21.539581
3	-17.685090	92.529971
4	-7.649027	-52.283744

b) Tabela_2

	X	Y
0	9.802967	1.975305
1	-1.687577	-9.856576
2	4.240441	9.056415
3	-7.411732	6.713138
4	-5.131219	8.583158

c) Tabela_3

	X	Y
0	-10.000000	7.890974
1	-10.000000	-6.383396
2	-10.000000	7.899138
3	-4.228799	-10.000000
4	-10.000000	-5.090268

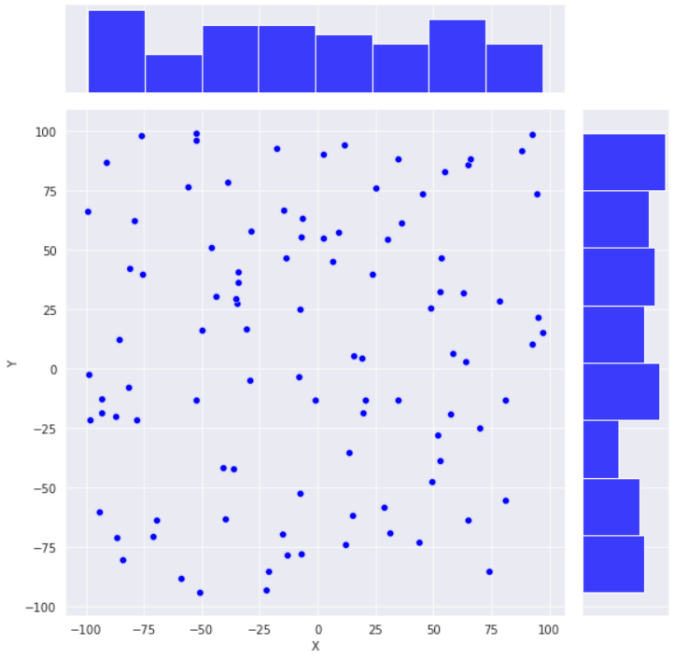
d) Tabela_4

	X	Y
0	4.613974	0.000000
1	0.562618	0.000000
2	1.757702	0.000000
3	1.737318	0.000000
4	0.000000	3.983858

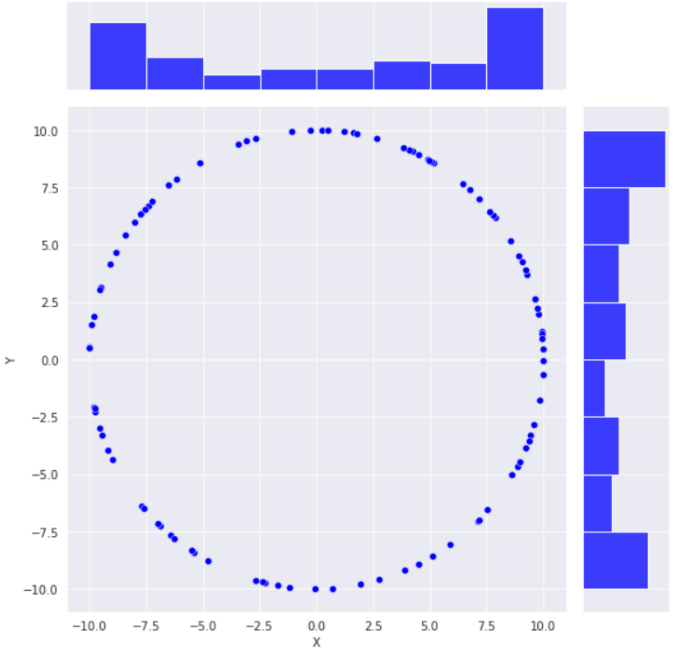
Do wygenerowania punktów z powyższych podpunktów użyto funkcji uniform z biblioteki numpy która znajduje się w module random (np.random.uniform). Użycie funkcji random.uniform z biblioteki numpy niż uniform ze standardowej biblioteki Python jest efektywniejsze, ponieważ działa szybciej oraz zapewnia znacznie większą liczbę rozkładów prawdopodobieństwa do wyboru. Wygenerowanie punktów dla podpunktów c) oraz d) zostało wygenerowane w najbardziej losowy możliwy sposób. Każdy bok ma swój przedział, jeżeli bok jest dłuższy to jego przedział jest większy, jeżeli bok jest krótszy to jego przedział jest też krótszy. Dzięki temu otrzymujemy jeden duży przedział w którym losowany jest 1 punkt i jest przydzielany do odpowiedniego boku. Taki podział generowania punktów na bokach prostokąta zapewnia najlepszy rozkład prawdopodobieństwa dla różnych długości boków.

Następnym krokiem była wizualizacja utworzonych zbiorów punktów:

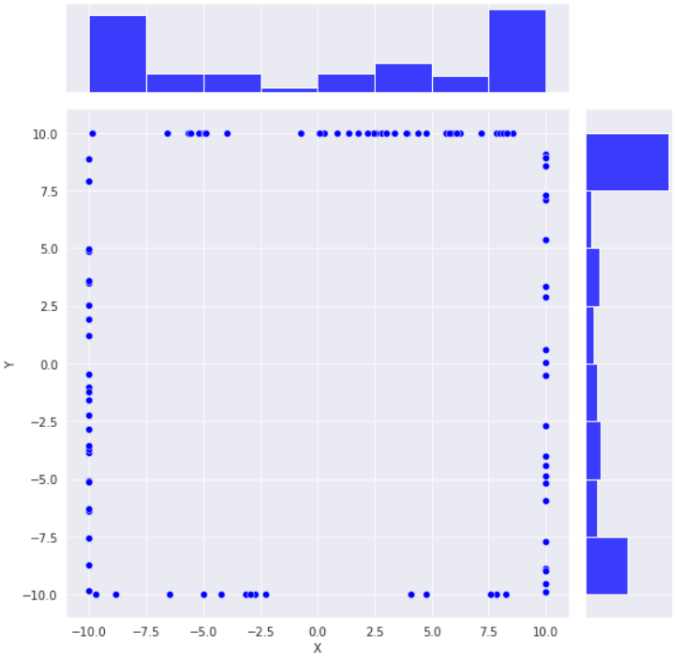
a) Wykres_1



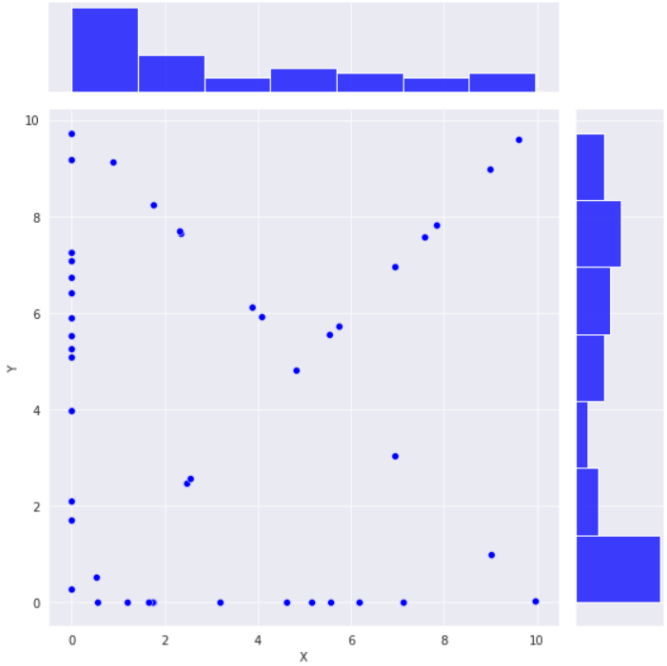
b) Wykres_2



c) Wykres_3



d) Wykres_4



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’.

Następnym krokiem była implementacja programów generujących losowe punkty na płaszczyźnie ale z możliwością dodania określonych parametrów dla poszczególnych schematów losowania punktów:

a) **random_points_on_the_range(num_of_points, ranges)** - generowanie zbioru punktów przedziału z parametrami liczba punktów i przedziały dla współrzędnych,
b) **random_points_on_the_circle(num_of_points, center, R)** - generowanie zbioru punktów leżących na okręgu z parametrami liczba punktów, środek i promień okręgu,
c) **random_points_on_the_rectangle(num_of_points, vertices)** - generowanie zbioru punktów leżących na prostokącie z parametrami liczba punktów i wierzchołki prostokąta,
d) **random_points_on_square(vertices, side_num_of_points, diag_num_of_points)** - generowanie zbioru punktów które leżą na bokach kwadratu leżących na osiach oraz na jego przekątnych z parametrami wierzchołki kwadratu, liczba punktów na osiach oraz liczba punktów na przekątnych.

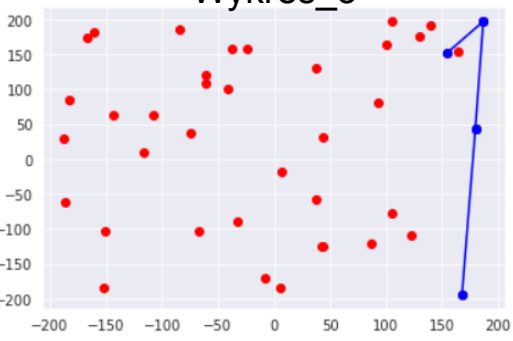
4. Wizualizacja graficzna działania algorytmów Grahama oraz Jarvis dla różnych podpunktów (epsilon = 1e-12), pełna wizualizacja znajduje się w Jupyter Notebooku, tutaj będą zawarte tylko niektóre wykresy

1) Algorytm Grahama

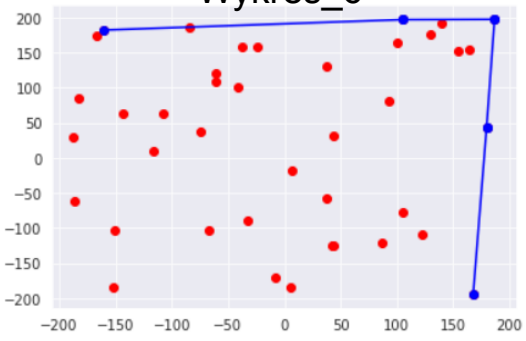
- dla podpunktu a)

z parametrami num_of_points = 40 i ranges = [-200, 200]

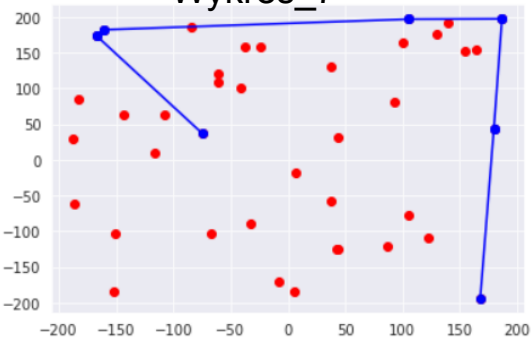
Wykres_5



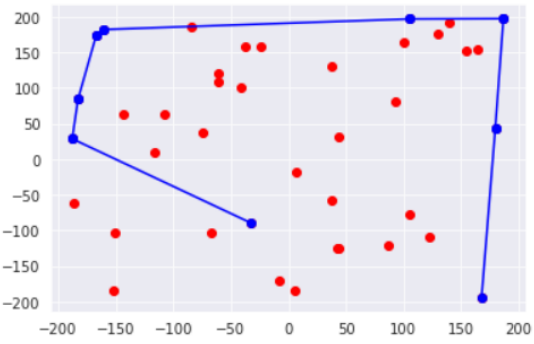
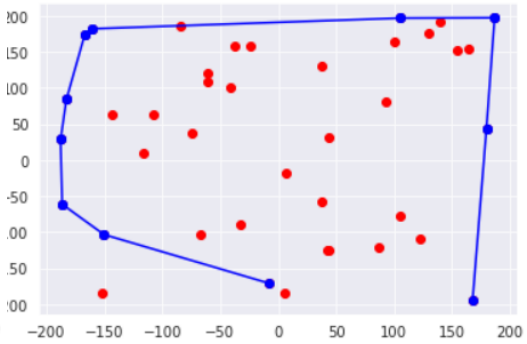
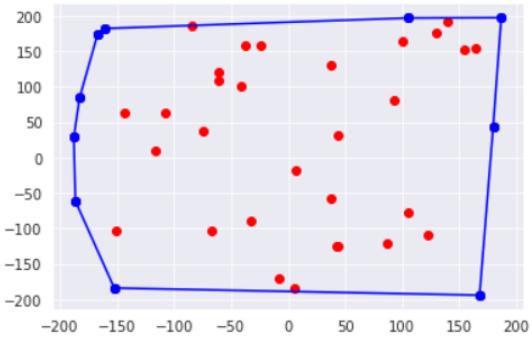
Wykres_6



Wykres_7



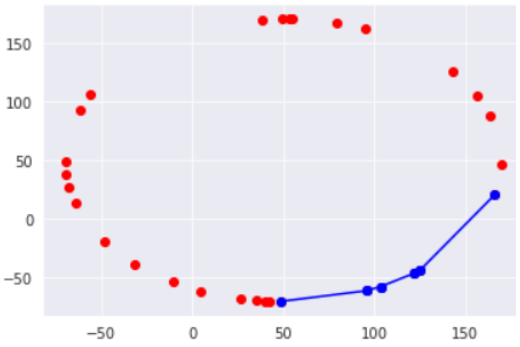
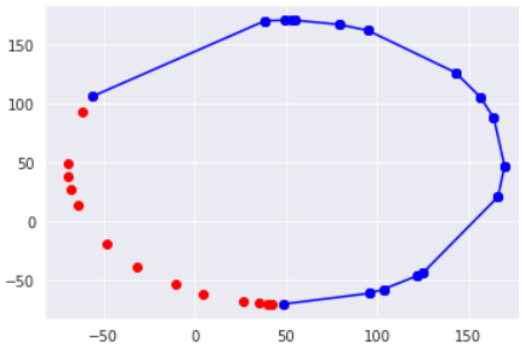
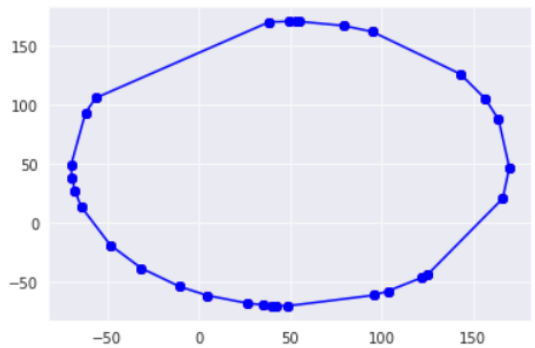
Wykres_8 Wykres_9 Wykres_10

- dla podpunktu b)

z parametrami num_of_points = 30, center=[50,50], R = 120

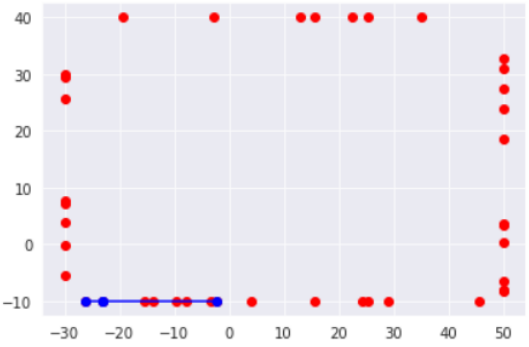
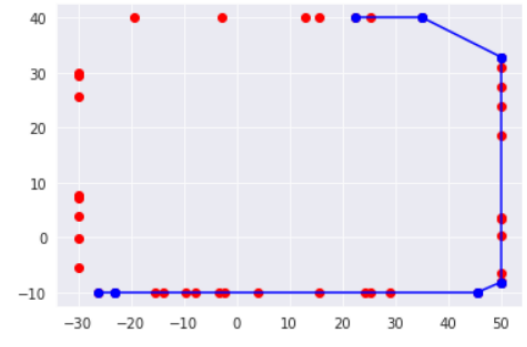
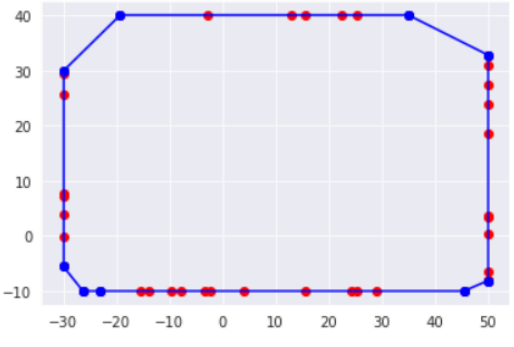
Wykres_11 Wykres_12 Wykres_13

- dla podpunktu c)

z parametrami num_of_points = 40, vertices = [(-30, -10), (-30, 40), (50, 40), (50, -10)]

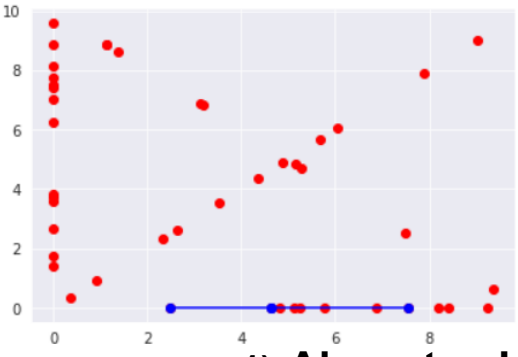
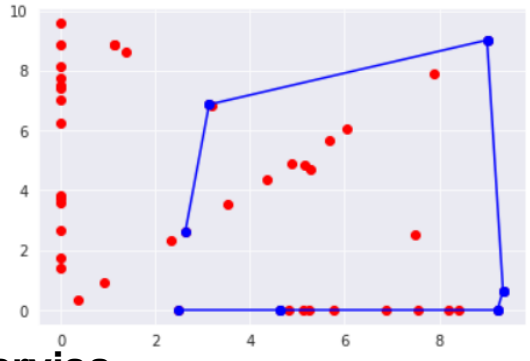
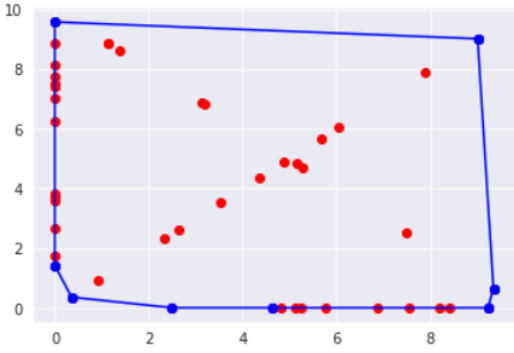
Wykres_14 Wykres_15 Wykres_16

- dla podpunktu d)

z parametrami vertices = [(-30, -10), (-30, 30), (10, 30), (10, -10)], side_num_of_points = 30, diag_num_of_points = 20

Wykres_17 Wykres_18 Wykres_19

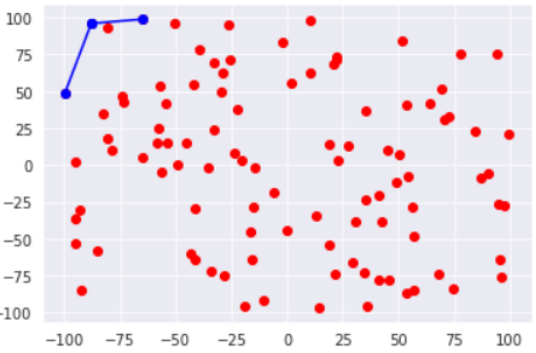
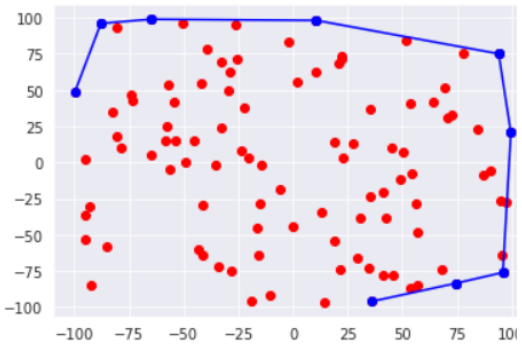
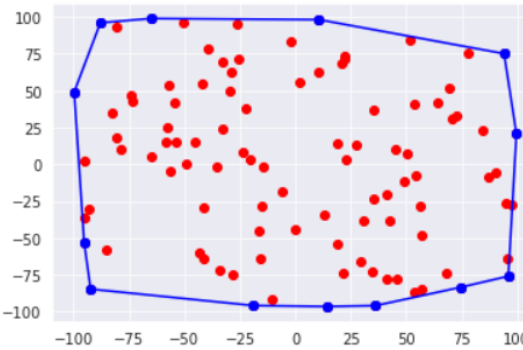




1) Algorytm Jarvisa

- dla podpunktu a)

z parametrami num_of_points = 100 i ranges = [-100, 100]

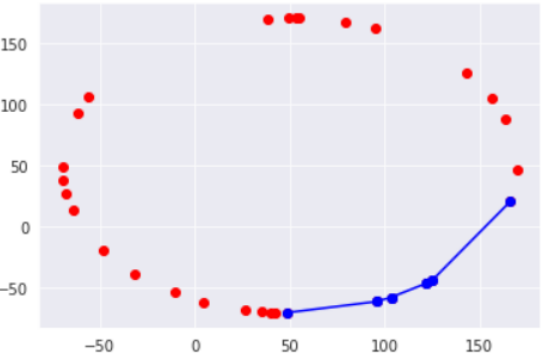
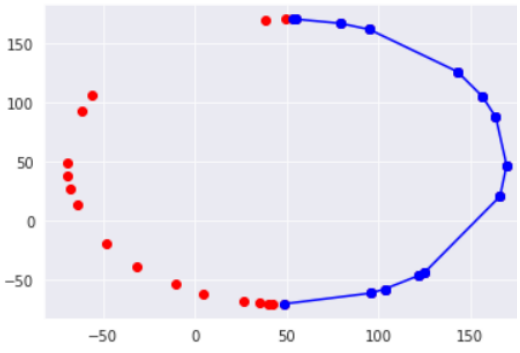
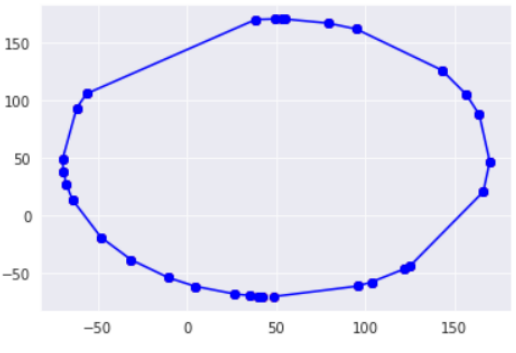
Wykres_20 Wykres_21 Wykres_22

- dla podpunktu b)

z parametrami num_of_points = 30, center=[50,50], R = 120

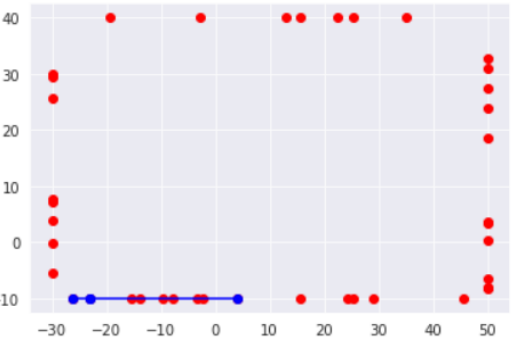
Wykres_23 Wykres_24 Wykres_25

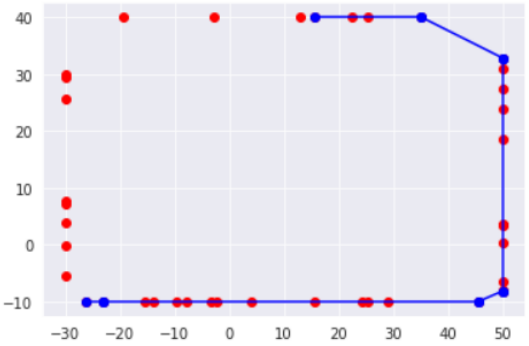
- dla podpunktu c)

z parametrami num_of_points = 100, vertices = [(-10, 10), (-10,-10), (10,-10), (10,10)]

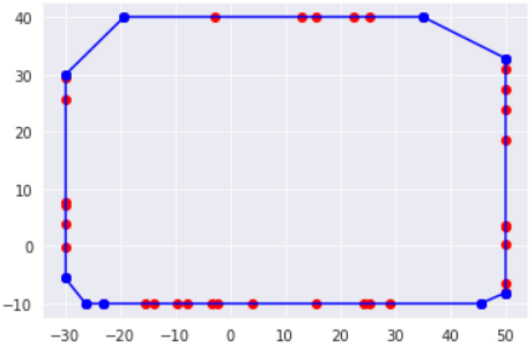
Wykres_26



Wykres_27



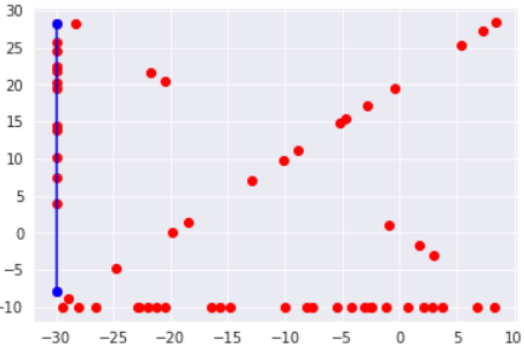
Wykres_28



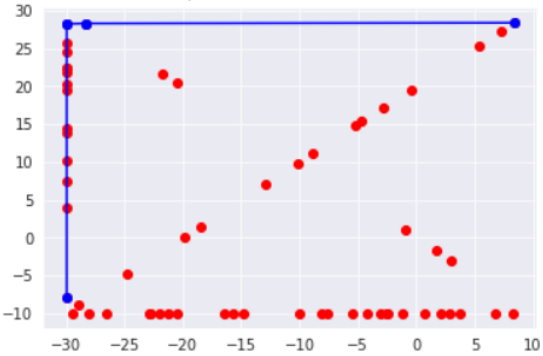
- dla podpunktu d)

z parametrami vertices = [(0, 0), (10, 0), (10, 10), (0, 10)], side_num_of_points = 25, diag_num_of_points = 20

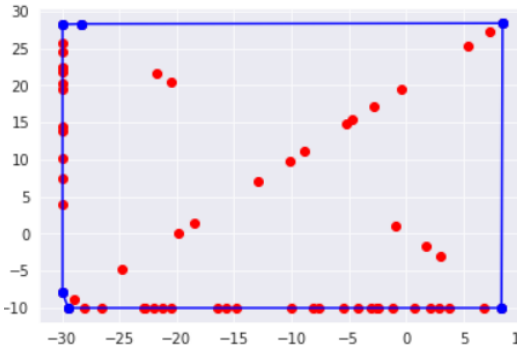
Wykres_29



Wykres_30



Wykres_31



5. Porównanie czasu działania algorytmów

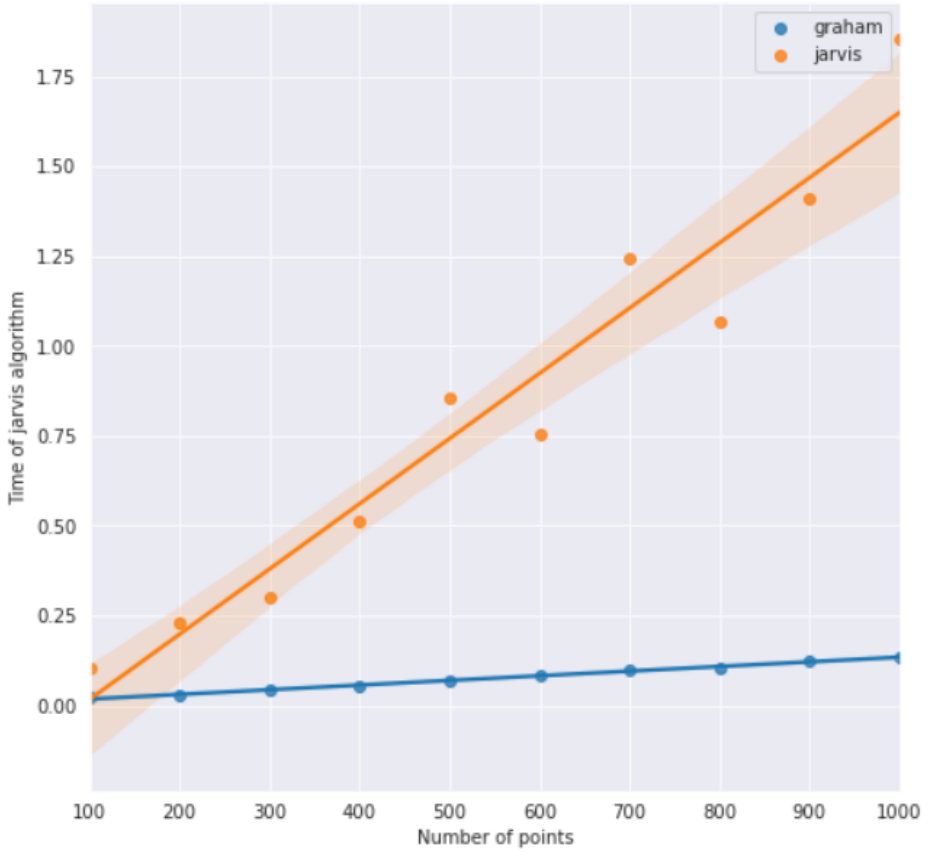
W poniższych tabelach przedstawione zostały czasy działania algorytmów Jarvisa i Grahama dla wszystkich podpunktów oraz dla różnych wartości parametrów. Każda tabela zawiera numerację przypadków, liczbę wygenerowanych punktów, czas działania algorytmu Graham oraz Jarvisa, wyznaczanie który algorytm w danym przypadku był szybszy oraz o ile. Na każdym wykresie jest zamieszczona legenda.

a) losowe punkty z przedziału

Tabela_5

	Number of points	a	b	Time of graham algorithm	Time of jarvis algorithm	Faster algorithm	Time comparison
0	100	-100	100	0.028619	0.137884	graham_algorithm	0.109264
1	200	-100	100	0.028077	0.269341	graham_algorithm	0.241264
2	300	-100	100	0.041202	0.428449	graham_algorithm	0.387248
3	400	-100	100	0.056122	0.541761	graham_algorithm	0.485639
4	500	-100	100	0.071184	0.656996	graham_algorithm	0.585812
5	600	-100	100	0.078102	0.906651	graham_algorithm	0.828549
6	700	-100	100	0.089049	1.095611	graham_algorithm	1.006562
7	800	-100	100	0.100741	1.291618	graham_algorithm	1.190877
8	900	-100	100	0.117528	1.420521	graham_algorithm	1.302993
9	1000	-100	100	0.129131	1.331670	graham_algorithm	1.202539

Wykres_32



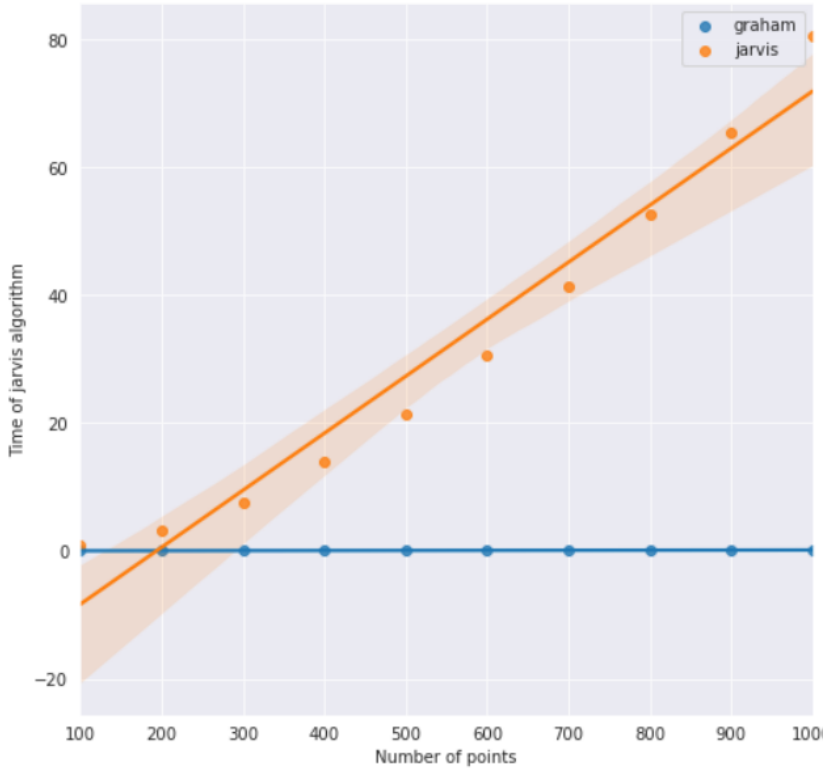
Na tym wykresie z porównania czasów działania obu algorytmów widzimy, że algorytm Grahama (niebieski) jest szybszy od algorytmu Jarvis (czerwony). Oznacza to, że liczba punktów na otoczce rośnie szybciej niż logn.

b) losowe punkty leżące na okręgu

Tabela_6

	Number of points	center	R	Time of graham algorithm	Time of jarvis algorithm	Faster algorithm	Time comparison
0	100	[-0.4961628901437649, -7.770366160383171]	7.863717	0.024287	0.843464	graham_algorithm	0.819177
1	200	[-13.95057208108308, 1.0384295612795285]	19.472292	0.024257	3.401207	graham_algorithm	3.376950
2	300	[1.4755798205675354, -5.795686029884299]	19.067874	0.040040	7.388680	graham_algorithm	7.348640
3	400	[-13.245204111995852, 2.942866064743548]	2.365222	0.051102	13.091011	graham_algorithm	13.039909
4	500	[-14.366870443816499, -2.2411443303571232]	11.042436	0.064771	21.112228	graham_algorithm	21.047457
5	600	[-1.4184727113720825, -6.1203547385979284]	15.973277	0.080665	30.762681	graham_algorithm	30.682016
6	700	[18.142821338089846, 15.909362616097717]	15.069453	0.102261	42.707016	graham_algorithm	42.604756
7	800	[3.9870101296239078, 15.513619998678088]	3.363271	0.102206	51.318779	graham_algorithm	51.216573
8	900	[-7.829066416744531, 11.641067188610528]	10.219645	0.112990	65.558680	graham_algorithm	65.445690
9	1000	[6.891920030203572, -10.876525263928873]	9.168661	0.139199	83.500267	graham_algorithm	83.361069

Wykres_33

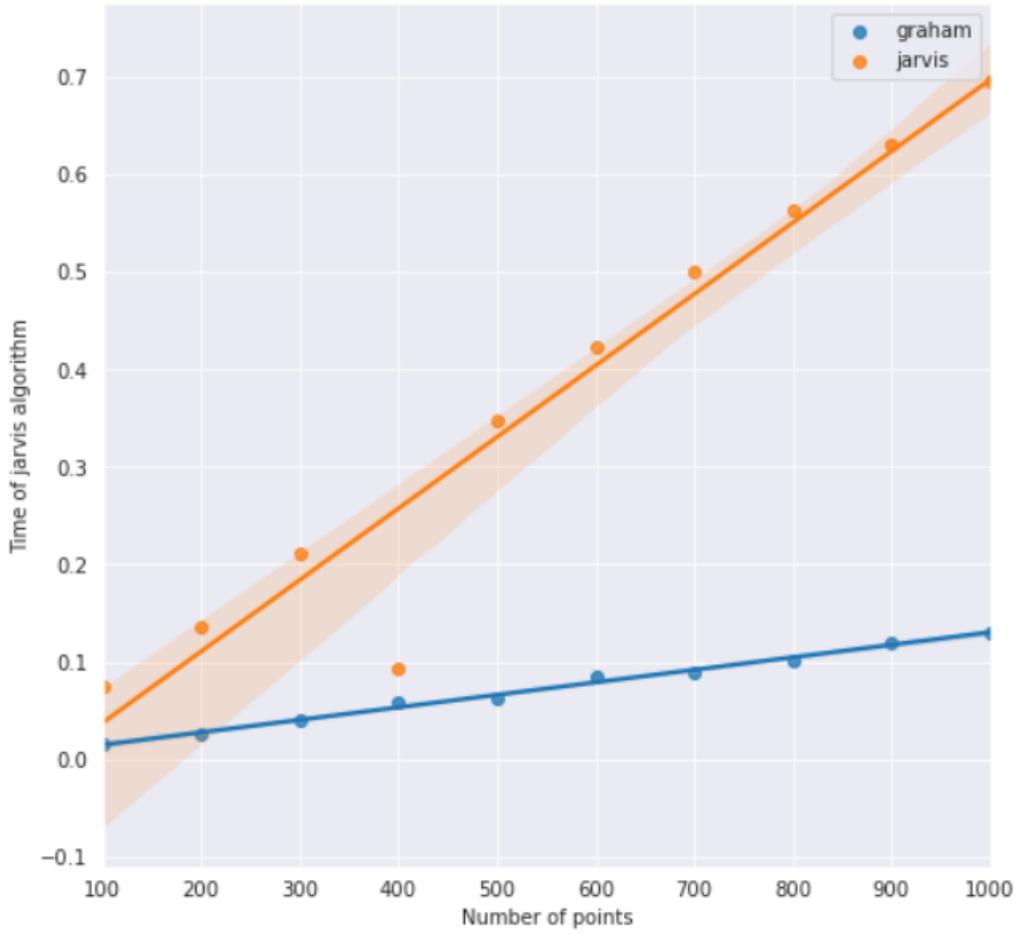


c) losowe punkty leżące na prostokącie

Tabela_7

	Number of points	vertices	Time of graham algorithm	Time of jarvis algorithm	Faster algorithm	Time comparison
0	100	[(2, -48), (2, -10), (24, -48), (24, -10)]	0.013060	0.069944	graham_algorithm	0.056884
1	200	[(39, 29), (39, -28), (0, 29), (0, -28)]	0.025212	0.140205	graham_algorithm	0.114993
2	300	[(27, -42), (27, -3), (3, -42), (3, -3)]	0.037483	0.210573	graham_algorithm	0.173090
3	400	[(-24, -48), (-24, 36), (-19, -48), (-19, 36)]	0.049001	0.277904	graham_algorithm	0.228903
4	500	[(-1, -7), (-1, 23), (35, -7), (35, 23)]	0.067435	0.350736	graham_algorithm	0.283300
5	600	[(-15, 0), (-15, -37), (-28, 0), (-28, -37)]	0.081926	0.459602	graham_algorithm	0.377676
6	700	[(-33, -5), (-33, -47), (-16, -5), (-16, -47)]	0.096910	0.524324	graham_algorithm	0.427414
7	800	[(36, -18), (36, -33), (-43, -18), (-43, -33)]	0.117984	0.602744	graham_algorithm	0.484760
8	900	[(26, -38), (26, -39), (-2, -38), (-2, -39)]	0.135696	0.684530	graham_algorithm	0.548833
9	1000	[(-26, -27), (-26, -8), (-24, -27), (-24, -8)]	0.143691	0.762155	graham_algorithm	0.618463

Wykres_34

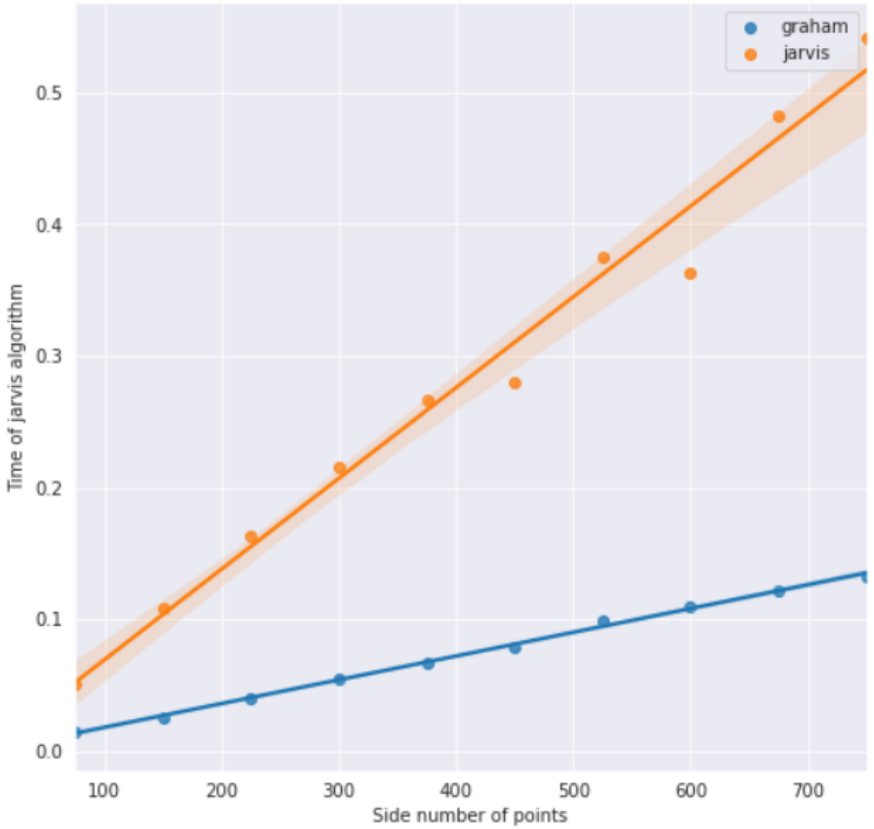


d) losowe punkty leżące na dwóch bokach kwadratu i na przekątnych

Tabela_8

	Side number of points	Diagonal number of points	vertices	Time of graham algorithm	Time of jarvis algorithm	Faster algorithm	Time comparison
0	50	50	[(-31, 23), (15, 23), (-31, 69), (15, 69)]	0.015279	0.066155	graham_algorithm	0.050877
1	100	100	[(33, 7), (33, 7), (33, 7), (33, 7)]	0.092763	3.545381	graham_algorithm	3.452618
2	150	150	[(-26, 33), (36, 33), (-26, 95), (36, 95)]	0.040962	0.195618	graham_algorithm	0.154656
3	200	200	[(10, -23), (-11, -23), (10, -2), (-11, -2)]	0.051766	0.234827	graham_algorithm	0.183061
4	250	250	[(1, 11), (-33, 11), (1, 45), (-33, 45)]	0.071525	0.332078	graham_algorithm	0.260553
5	300	300	[(37, -35), (-31, -35), (37, 33), (-31, 33)]	0.078913	0.428925	graham_algorithm	0.350013
6	350	350	[(29, 37), (-38, 37), (29, 104), (-38, 104)]	0.098258	0.326969	graham_algorithm	0.228710
7	400	400	[(-17, -35), (-41, -35), (-17, -11), (-41, -11)]	0.107694	0.510064	graham_algorithm	0.402370

Wykres_35



6. Wnioski

Algorytm Grahama dla większości zbiorów punktów jest szybszy od algorytmu Jarvisa. W szczególności algorytm Grahama lepiej sprawdza się dla zbiorów punktów, gdzie ich duża część należy do otoczki. Na podstawie otrzymanych wykresów 32-35 można stwierdzić, że czas wykonywania się algorytmów był zgodny z ich przewidywaną złożonością. Dla algorytmu Grahama O(nlogn) a dla algorytmu Jarvisa O(n*k). W poszczególnych przypadkach np. w podpunkcie a) algorytm Grahama okazał się lepszy od algorytmu Jarvisa, ponieważ funkcja log(n) od której zależna jest złożoność algorytmu Grahama rośnie wolniej niż wartość k. Jeżeli chodzi o podpunkt b) to algorytm Jarvisa dla większości przypadków miał złożoność $O(n^2)$, ponieważ wszystkie punkty wchodziły w skład otoczki wypukłej. Spowodowało to, że był on znacznie wolniejszy niż algorytm Grahama. Przechodząc do podpunktów c) i d), gdzie stosunkowo ilość punktów, które należała do otoczki była mała, algorytm Jarvisa miał złożoność bliską O(n). Podpunkt d) okazał się podpunktem dla którego oba algorytmy radziły sobie najlepiej ze znalezieniem otoczki wypukłej z pozostałych podpunktów. Najgorzej radziły sobie ze znalezieniem otoczki w podpunkcie b), czyli dla punktów należących do okręgu. Dzięki wyznaczonej regresji liniowej na każdym wykresie, możemy ocenić, że złożoność algorytmu Jarvisa dla podpunktu b) wynosi $O(n^2)$. W pozostałych przypadkach dla obydwu algorytmów obserwujemy złożoność zbliżoną do liniowej.