Algorytmy geometryczne

Sprawozdanie z laboratorium 2.

Dawid Zawiślak gr. Czw. 13:00 A

Dane techniczne urządzenia oraz narzędzia za pomocą których wykonano ćwiczenia:

- Laptop z systemem operacyjnym Windows 11
- Procesor Intel Pentium Gold 8505
- RAM 8 GB

Do realizacji ćwiczenia użyto środowiska Jupyter Notebook z Pythonem w wersji 3.9, wykorzystując bibliotekę numpy (do obliczeń numerycznych), random (do generowania liczb pseudolosowych), pandas (do wyświetlania podsumowań w tabelach), time (do mierzenia czasu wykonywania algorytmów) oraz visualizer przygotowany przez KN BIT(do tworzenia wykresów).

Opis ćwiczenia:

Naszym celem na drugich zajęciach laboratoryjnych było wyznaczanie otoczki wypukłej dla danych zbiorów punktów, czyli najmniejszego zbioru punktów tworzącego wielokąt wypukły i zawierającego wewnątrz wszystkie pozostałe punkty.
Do wyznaczania otoczki użyte zostały dwa algorytmy: algorytm Grahama oraz algorytm Jarvisa.

Algorytm Grahama:

1) W zbiorze punktów Q wybieramy punkt p_0 o najmniejszej współrzędnej y, oraz najmniejszą współrzędną x w przypadku, gdy wiele punktów ma tą samą współrzędną y.
2) Niech (p_1, p_2, .., p_n) będzie pozostałym zbiorem punktów w Q posortowanym zgodnie z przeciwnym ruchem wskazówek zegara wokół punktu p_0, licząc kąt odchylenia od dodatniego kierunku osi OX (jeżeli więcej niż jeden punkt ma ten sam kąt to usuwamy wszystkie punkty z wyjątkiem tego najbardziej oddalonego od p_0).
3) Tworzymy pusty stos S i umieszczamy w nim kolejno punkty p_0, p_1, p_2.
   t – indeks ostatniego element stosu.
4) **For** i = 3 .. n

   While kąt utworzony przez S[t-1], S[t] oraz p_i tworzy lewostronny skręt

   S.pop()

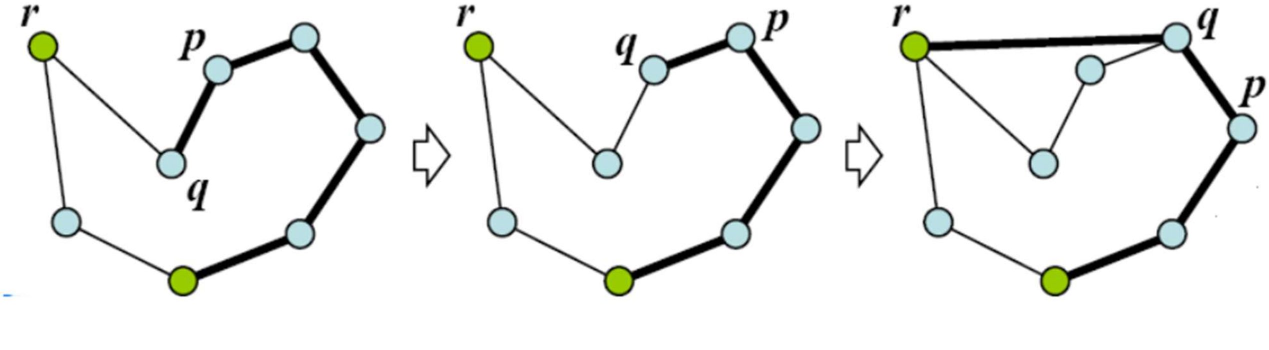
   S.append(p_i)

5) Po wykonaniu tej pętli na stosie znajdują się tylko wierzchołki należące do otoczki.

Złożoność obliczeniowa:

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

szukanie p_0 sortowanie krok 4)



Algorytm Jarvisa (owijanie prezentu):

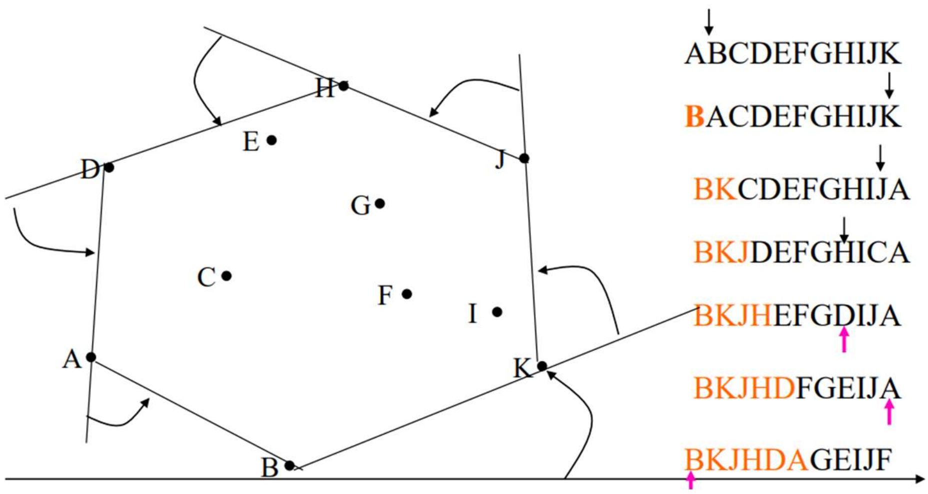
1) W zbiorze punktów Q wybieramy punkt p_0 o najmniejszej współrzędnej y, oraz najmniejszą współrzędną x w przypadku, gdy wiele punktów ma tą samą współrzędną y.
2) **For** i = 0 .. n

   For j = 0 .. i -1, i+1 .. n:

   Znajdź punkt, dla którego kąt liczny przeciwnie do wskazówek zegara odniesieniu do ostatniej krawędzi otoczki jest najmniejszy i dodaj go do zbioru punktów otoczki.
   Jeśli ostatnio dodany wierzchołek to p_0 przerwij algorytm.

Złożoność obliczeniowa:

O(n²), lub gdy liczba wierzchołków otoczki jest ograniczona przez stałą k, to **O(nk)**



Wykonanie ćwiczenia:

1. Generowanie zbiorów punktów do testów

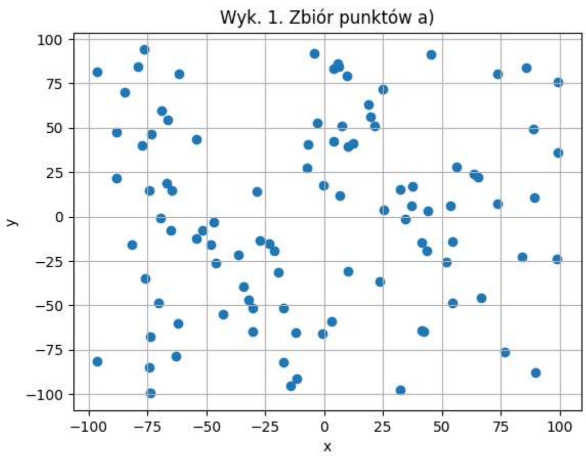
W celu realizacji ćwiczenia konieczne było przygotowanie odpowiednich danych testowych. Do pierwszej części wykorzystałem następujące zbiory współrzędnych typu float wygenerowane przy pomocy funkcji **uniform** z biblioteki **random**:

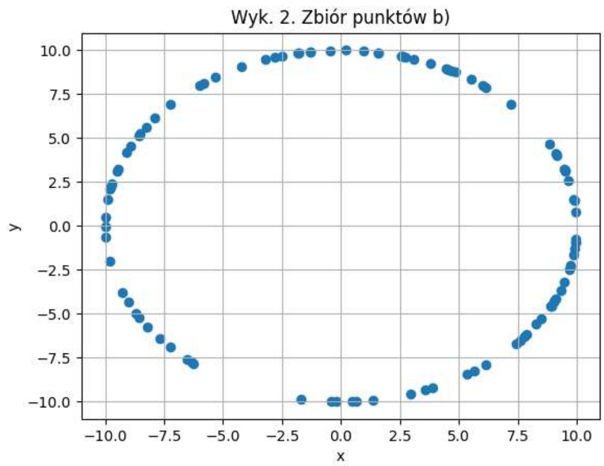
a) zawierający 100 losowo wygenerowanych punktów o współrzędnych z przedziału [-100, 100] (Wyk 1.),

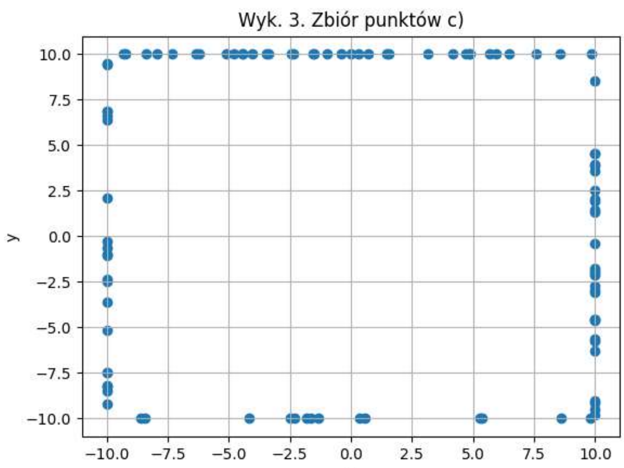
b) zawierający 100 losowo wygenerowanych punktów leżących na okręgu o środku (0,0) i promieniu R=10 (Wyk 2.),

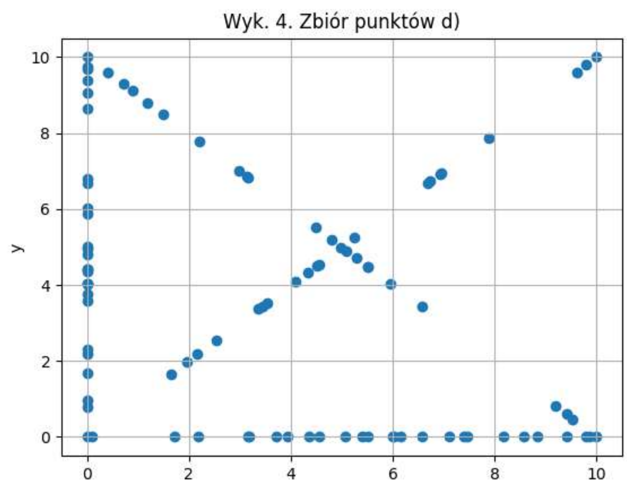
c) zawierający 100 losowo wygenerowanych punktów leżących na bokach prostokąta o wierzchołkach (-10, 10), (-10,-10), (10,-10), (10,10) (Wyk 3.),

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 (Wyk 4.),









Do wykonania kolejnej części zadania przygotowałem funkcje generujące sparametryzowane zbiory odpowiadające typom z 4.1. ze względu na ilość i położenie punktów:

a) generate_uniform_points(left, right, n) – generująca zbiór A przy podanych granicach osi x i y oraz zadanej liczności
b) generate_circle_points(O, R, n) – generująca zbiór B o zadanym środku i promieniu oraz liczności
c) generate_rectangle_points(a, b, c, d, n) – generująca zbiór C na określonym prostokącie i zadanej liczności
d) generate_square_points(a, b, c, d, axis_n, diag_n) – generująca zbiór D na określonym kwadracie oraz z określoną licznością punktów na przekątnych i bokach

2. Wyznaczanie otoczek dla pierwszych zbiorów testowych

Algorytmy są realizowane odpowiednio przez funkcje: **graham_algorithm, jarvis_algorithm**, a ich wizualizacje przez **graham_algorithm_draw, jarvis_algorithm_draw.**

Do wyznaczania odchylenia punktów użyłem funkcji **orient**, która korzysta z wyznacznika 3x3 z tolerancją dla wartości zera wykorzystywaną podczas porównań równą 0, ponieważ dla takiej wartości algorytmy przechodziły wszystkie testy poprawnie.

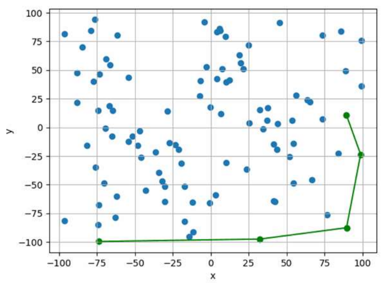
W algorytmie Grahama sortowanie realizuje za pomocą funkcji sort dostępnej w Pythonie oraz zmodyfikowanemu komparatorowi, których sortuje punkty po kącie odchylenia i odległości od punktów p_o.

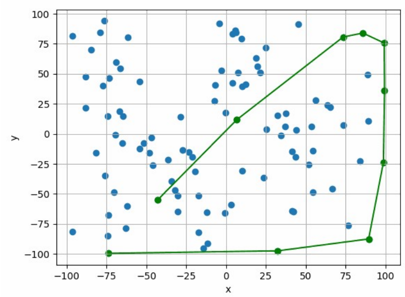
Algorytm Jarvisa jeśli funkcja orient zwraca 0, to sprawdzam który punkt jest oddalony bardziej od punktu p_o, aby uniknąć dodawanie niepotrzebnych współliniowych punktów do otoczki zwiększając tym stałą k.

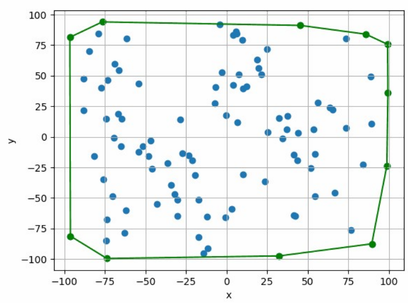
Wizualizacja wszystkich kroków obu algorytmów zawarta jest w jupyter notebooku w postaci gifów, a poniżej umieszczam przykładowe kroki działania tych algorytmów:

Zbiór A

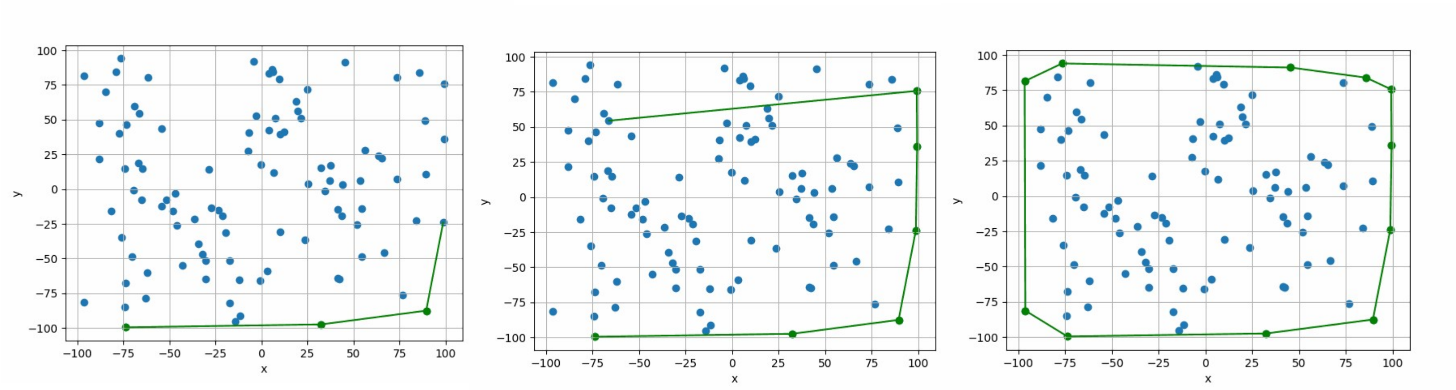
Algorytm Grahama:





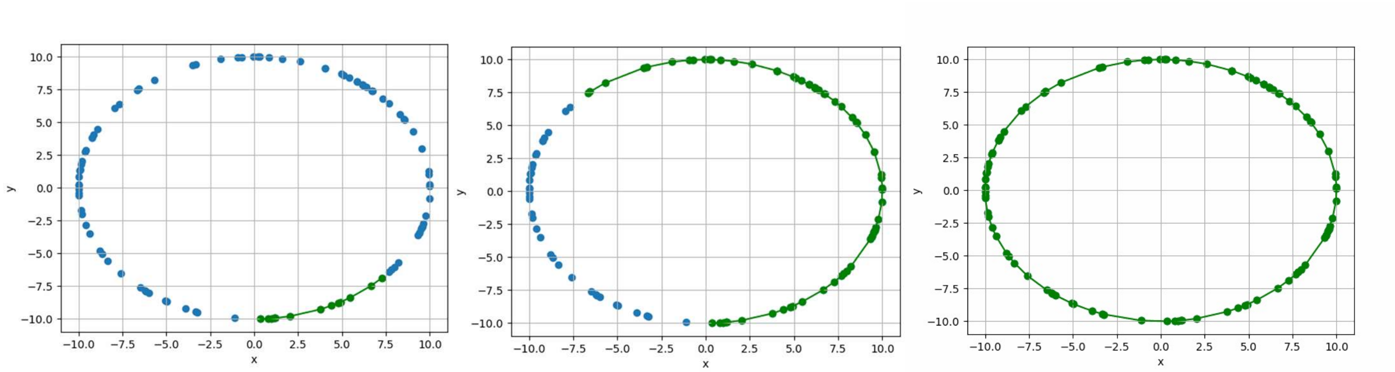


Algorytm Jarvis:

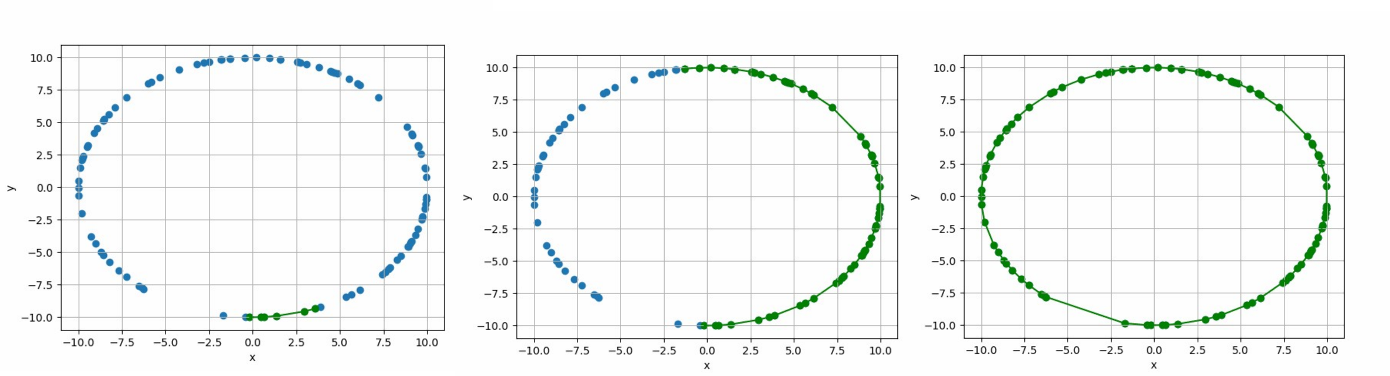


Zbiór B

Algorytm Grahama:

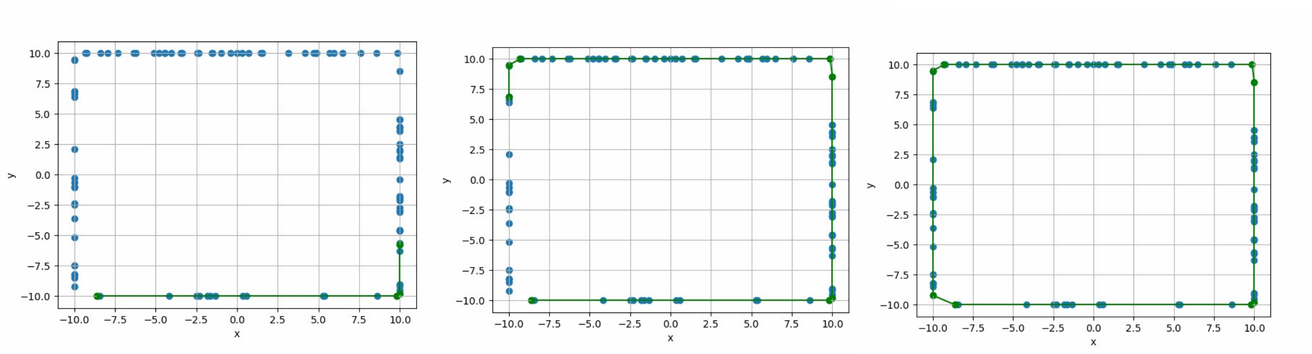


Algorytm Jarvisa:

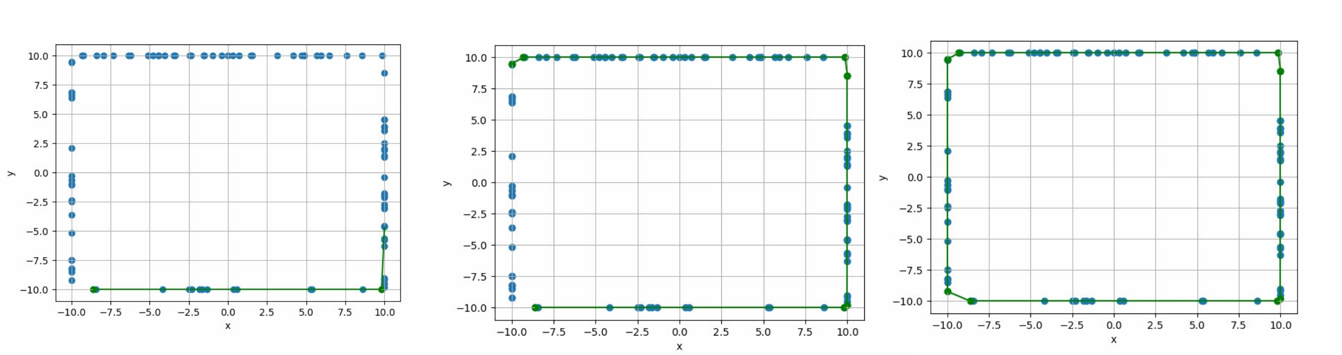


Zbiór C

Algorytm Grahama:

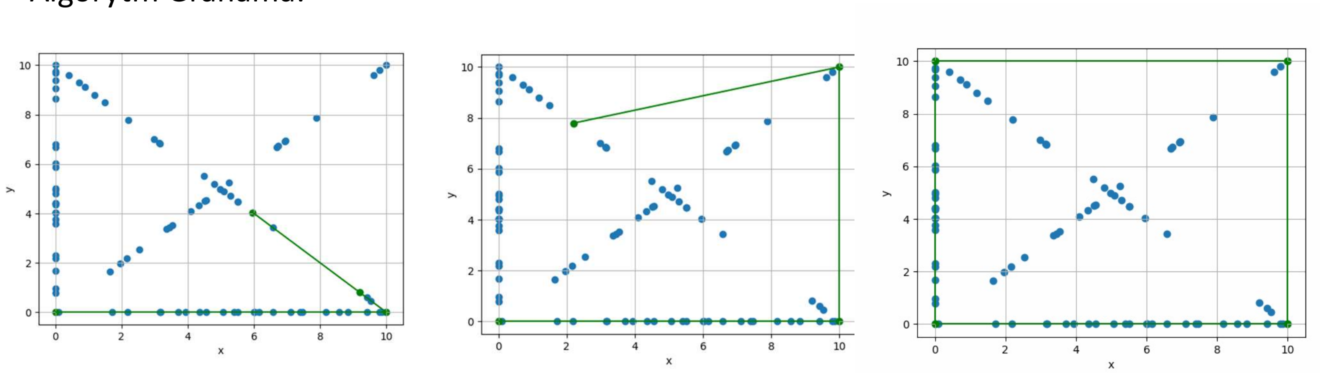


Algorytm Jarvisa:

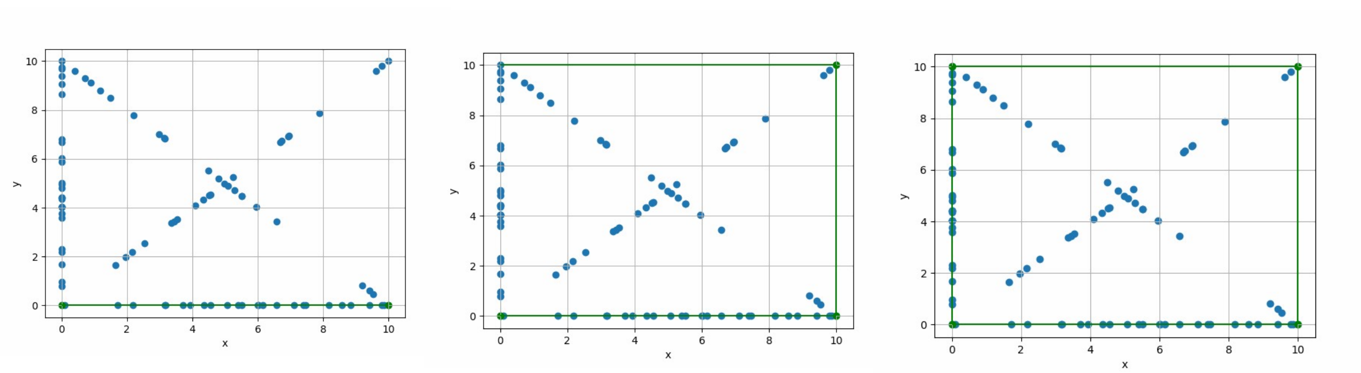


Zbiór D

Algorytm Grahama:



Algorytm Jarvisa:



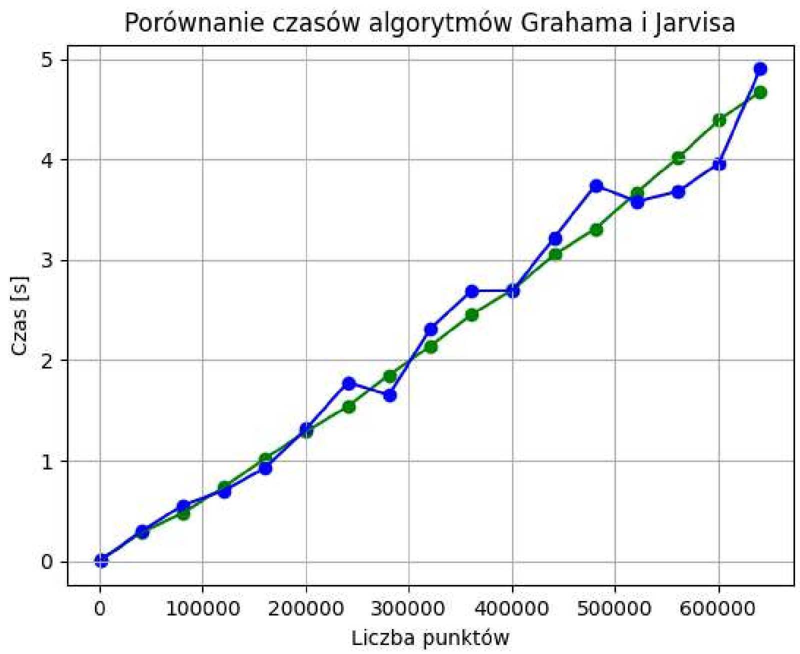
3. Porównanie czasów algorytmów

Do porównania czasów działania algorytmów użyłem pomocniczej funkcji **test_set** oraz funkcji **count_time** korzystającej z funkcji bibliotecznej **perf_counter** z modułu **time.** Porówanie czasów działania obu algorytmów dla określonych danych testowych przedstawiłem w tabelach oraz na wykresach, gdzie na zielono pokazany jest algorytm Grahama, a na niebiesko algorytm Jarvisa.

a) Zbiory typu A, losowe punkty, gdzie współrzędne x i y są z zadanego zakresu

Liczba punktów	Początek zakresu	Koniec zakresu	Czas wykonania Graham [s]	Czas wykonania Jarvis [s]	Szybszy algorytm	Różnica czasów [s]
1000	-200	200	0.003193	0.004318	Graham	0.001125
41000	-200	200	0.284798	0.296657	Graham	0.011859
81000	-200	200	0.476044	0.554110	Graham	0.078066
121000	-200	200	0.738307	0.701088	Jarvis	0.037219
161000	-200	200	1.020744	0.927454	Jarvis	0.093289
201000	-200	200	1.291939	1.317741	Graham	0.025802
241000	-200	200	1.539199	1.780863	Graham	0.241664
281000	-200	200	1.853821	1.652859	Jarvis	0.200962
321000	-200	200	2.140468	2.313581	Graham	0.173113
361000	-200	200	2.455875	2.691559	Graham	0.235684
401000	-200	200	2.703166	2.694144	Jarvis	0.009022
441000	-200	200	3.051110	3.218369	Graham	0.167259
481000	-200	200	3.309553	3.743119	Graham	0.433566
521000	-200	200	3.672541	3.581228	Jarvis	0.091313
561000	-200	200	4.016505	3.680975	Jarvis	0.335530
601000	-200	200	4.396061	3.963846	Jarvis	0.432215
641000	-200	200	4.673584	4.903598	Graham	0.230014

Tabela 1 Czasy działania dla zbiorów typu A

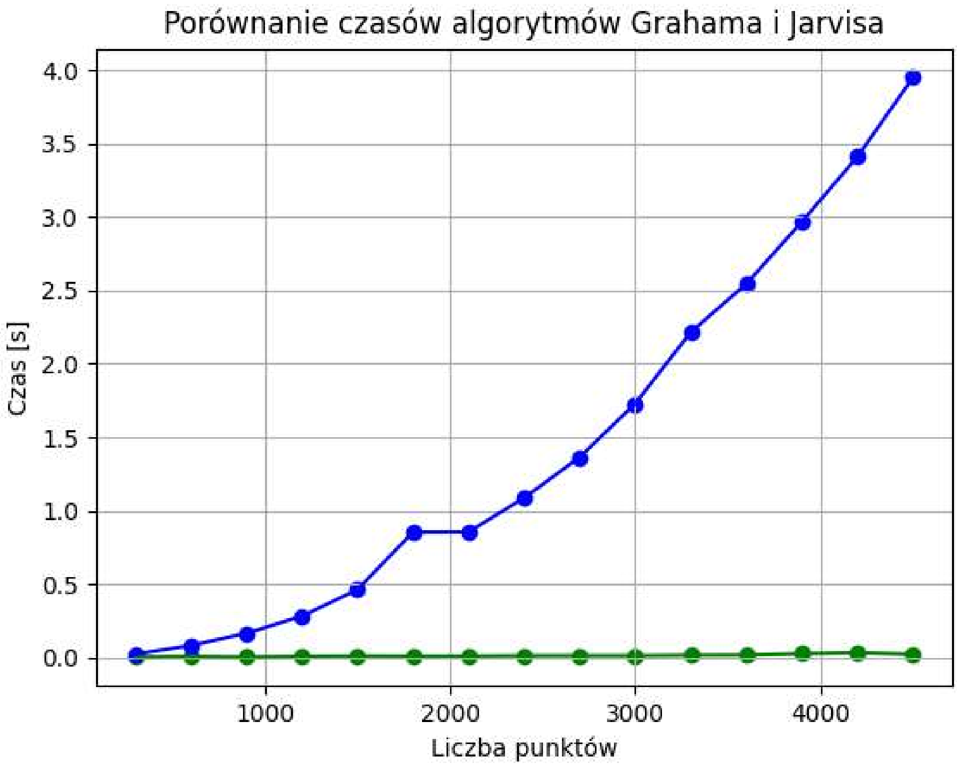


Wyk. 5 Porównanie czasów algorytmów Grahama i Jarvisa dla zbiorów typu A. Zielony - Graham, niebieski - Jarvis

b) Zbiory typu B, punkty leżące na okręgu o określonym promieniu i współrzędnych środka

Liczba punktów	Środek okręgu	Promień	Czas wykonania Graham [s]	Czas wykonania Jarvis [s]	Szybszy algorytm	Różnica czasów [s]
300	(-0.11055207335448713, 7.419062625818988)	0.201019	0.005294	0.023310	Graham	0.018016
600	(-13.84818445942873, 7.299878049784773)	13.881287	0.008281	0.082019	Graham	0.073739
900	(12.3100858566944, -16.3289992796517)	3.510356	0.004423	0.164374	Graham	0.159951
1200	(18.92771523480465, -8.582221517120825)	4.239726	0.009161	0.281906	Graham	0.272745
1500	(15.535745196033375, -18.65253120581162)	10.791620	0.011031	0.461919	Graham	0.450888
1800	(1.0314657998888848, 12.301092703729509)	11.808267	0.010304	0.853397	Graham	0.843093
2100	(-6.885298665495728, 11.69638668958942)	3.313690	0.010248	0.855146	Graham	0.844898
2400	(15.360391458752865, -10.650218976815577)	9.214257	0.012832	1.086719	Graham	1.073887
2700	(15.964175075818375, -18.98161682080596)	0.576863	0.013340	1.364113	Graham	1.350773
3000	(17.02776098805576, 16.066715262349305)	12.557783	0.013103	1.728766	Graham	1.715663
3300	(-0.6440550043916673, -7.148960867922893)	10.578697	0.016178	2.215686	Graham	2.199508
3600	(-6.873279251657994, -2.190039493173643)	2.122663	0.017358	2.544464	Graham	2.527107
3900	(-4.617739278077902, 7.552694559770515)	10.002858	0.026890	2.967845	Graham	2.940955
4200	(10.846741216067418, -12.409721480070623)	13.597140	0.033349	3.410613	Graham	3.377265
4500	(-16.943244533672388, -2.5773992846916904)	19.463036	0.022895	3.949900	Graham	3.927005

Tabela 2 Czasy działania dla zbiorów typu B

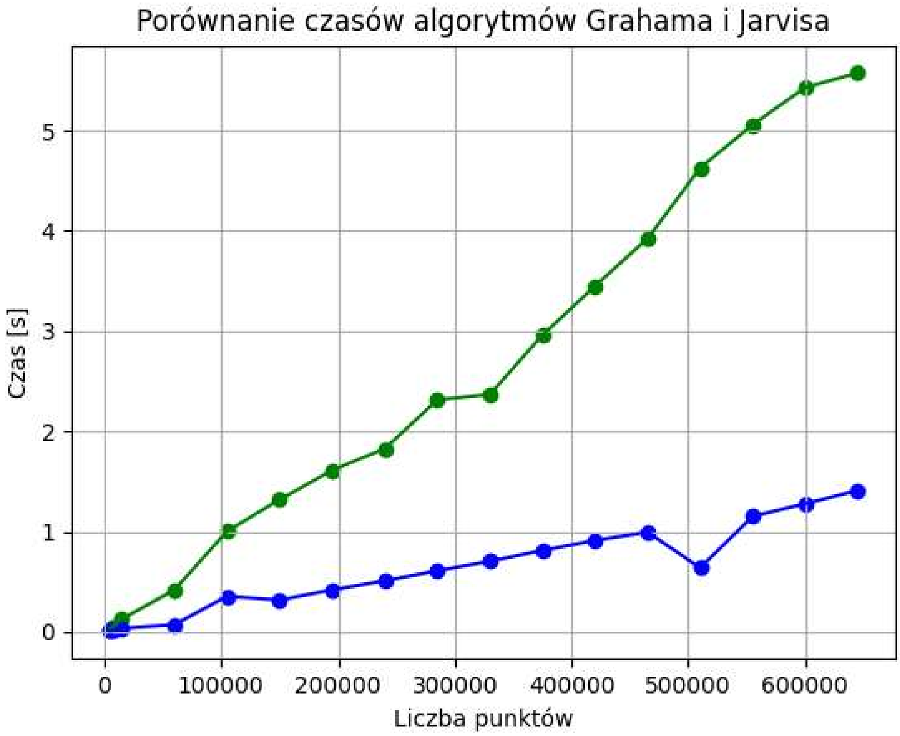


Wyk. 6 Porównanie czasów algorytmów Grahama i Jarvisa dla zbiorów typu B. Zielony - Graham, niebieski - Jarvis

c) Zbiory typu C, punkty leżące na prostokącie o ustalonych wierzchołkach

Liczba punktów	Wierzchołki prostokąta	Czas wykonania Graham [s]	Czas wykonania Jarvis [s]	Szybszy algorytm	Różnica czasów [s]
3750	((-57, -49), (13, -49), (13, -16), (-57, -16))	0.021881	0.007913	Jarvis	0.013968
7500	((48, -8), (0, -8), (0, 56), (48, 56))	0.052161	0.012149	Jarvis	0.040012
15000	((-57, -49), (13, -49), (13, -16), (-57, -16))	0.126767	0.034614	Jarvis	0.092152
60000	((48, -8), (0, -8), (0, 56), (48, 56))	0.417019	0.070768	Jarvis	0.346251
105000	((-18, -11), (47, -11), (47, -8), (-18, -8))	1.003209	0.353546	Jarvis	0.649663
150000	((-20, -49), (39, -49), (39, -42), (-20, -42))	1.317080	0.315595	Jarvis	1.001486
195000	((0, -27), (68, -27), (68, 27), (0, 27))	1.611766	0.418100	Jarvis	1.193666
240000	((-33, -45), (43, -45), (43, 11), (-33, 11))	1.825264	0.508759	Jarvis	1.316505
285000	((-21, -31), (66, -31), (66, 0), (-21, 0))	2.313407	0.608249	Jarvis	1.705158
330000	((-40, -51), (-33, -51), (-33, -24), (-40, -24))	2.367545	0.703342	Jarvis	1.664202
375000	((-36, -16), (49, -16), (49, 62), (-37, 62))	2.959018	0.810726	Jarvis	2.148292
420000	((-37, -16), (49, -16), (49, 62), (-37, 62))	3.448505	0.910864	Jarvis	2.537641
465000	((-57, -49), (13, -49), (13, -16), (-57, -16))	3.922117	0.992983	Jarvis	2.929134
510000	((48, -8), (0, -8), (0, 56), (48, 56))	4.626650	0.637666	Jarvis	3.988984
555000	((-18, -11), (47, -11), (47, -8), (-18, -8))	5.057593	1.150994	Jarvis	3.906599
600000	((-20, -49), (39, -49), (39, -42), (-20, -42))	5.429443	1.276433	Jarvis	4.153010
645000	((0, -27), (68, -27), (68, 27), (0, 27))	5.573162	1.410970	Jarvis	4.162193

Tabela 3 Czasy działania dla zbiorów typu C

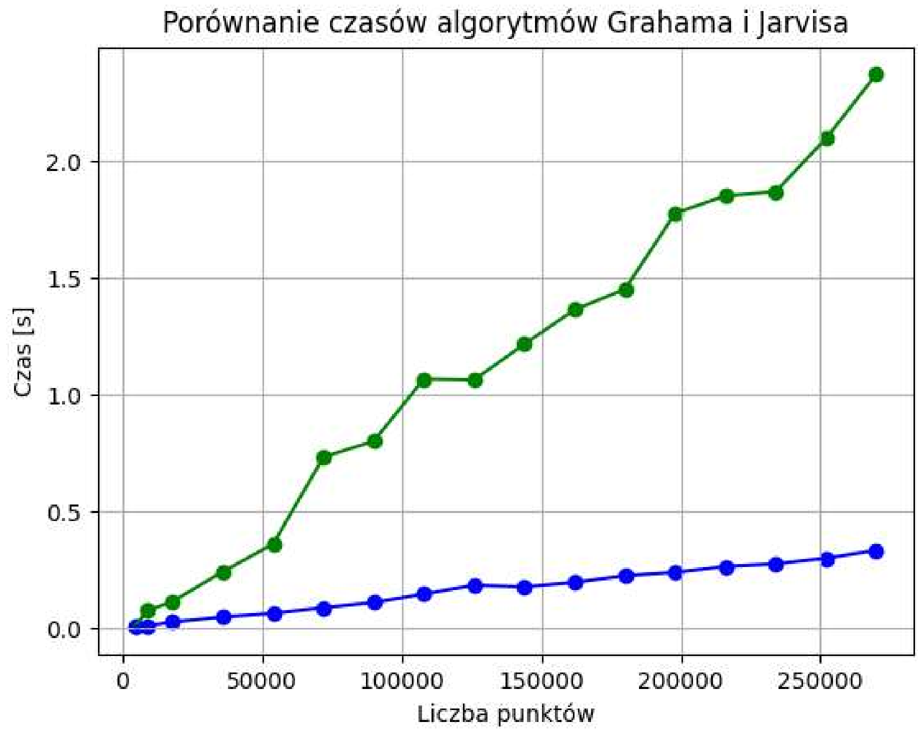


Wyk. 7 Porównanie czasów algorytmów Grahama i Jarvisa dla zbiorów typu C. Zielony - Graham, niebieski - Jarvis

d) Zbiory typu D, punkty leżące na kwadracie o ustalonych wierzchołkach i liczbie punktów na 2 bokach i 2 przekątnych

Liczba punktów na boku	Liczba punktów na przekątnej	Wierzchołki	Czas wykonania Graham [s]	Czas wykonania Jarvis [s]	Szybszy algorytm	Różnica czasów [s]
1500	750	((-31, 21), (23, 21), (23, 75), (-31, 75))	0.008265	0.004370	Jarvis	0.003895
3000	1500	((-16, -10), (4, -10), (4, 5), (-16, 5))	0.074955	0.009859	Jarvis	0.065096
6000	3000	((-31, 21), (23, 21), (23, 75), (-31, 75))	0.114244	0.026245	Jarvis	0.087999
12000	6000	((-16, -10), (4, -10), (4, 5), (-16, 5))	0.241936	0.047409	Jarvis	0.194527
18000	9000	((1, 34), (69, 34), (69, 100), (1, 100))	0.360404	0.063199	Jarvis	0.297206
24000	12000	((-5, -50), (38, -50), (38, 0), (-5, 0))	0.733465	0.087651	Jarvis	0.645814
30000	15000	((31, 17), (70, 17), (70, 56), (31, 56))	0.800618	0.109606	Jarvis	0.691012
36000	18000	((0, 2), (85, 2), (85, 87), (0, 87))	1.068053	0.145897	Jarvis	0.922156
42000	21000	((34, -36), (64, -36), (64, -6), (34, -6))	1.064480	0.183544	Jarvis	0.880936
48000	24000	((-5, -12), (37, -12), (37, 30), (-5, 30))	1.216845	0.176199	Jarvis	1.040646
54000	27000	((-5, -29), (41, -29), (41, 17), (-5, 17))	1.364952	0.196046	Jarvis	1.168906
60000	30000	((-26, -13), (25, -13), (25, 28), (-26, 28))	1.452393	0.224214	Jarvis	1.228179
66000	33000	((-31, 21), (23, 21), (23, 75), (-31, 75))	1.778072	0.238770	Jarvis	1.539302
72000	36000	((-16, -10), (4, -10), (4, 5), (-16, 5))	1.852724	0.263921	Jarvis	1.588803
78000	39000	((1, 34), (69, 34), (69, 100), (1, 100))	1.870918	0.275453	Jarvis	1.595465
84000	42000	((-5, -50), (38, -50), (38, 0), (-5, 0))	2.098240	0.299656	Jarvis	1.798585
90000	45000	((31, 17), (70, 17), (70, 56), (31, 56))	2.373265	0.332814	Jarvis	2.040451

Tabela 4 Czasy działania dla zbiorów typu D



Wyk. 8 Porównanie czasów algorytmów Grahama i Jarvisa dla zbiorów typu D. Zielony - Graham, niebieski - Jarvis

4. Wnioski

Analizowane algorytmy działały poprawnie dla wszystkich zadanych zbiorów testowych. Zbiory takie zaproponowano zapewne dlatego, aby można było oszacować złożoności algorytmów znając ich charakterystyki.

Dla zbiorów typu A algorytmy działały w podobnym czasie (Tabela 1 I Wyk. 5), punkty są losowe i stała k może być tego samego rzędu co logn.

Dla danych typu B widzimy, że złożoność algorytmu Jarvisa jest równa $O(n^2)$, ponieważ w otoczce znajdują się wszystkie punkty zbioru i dla tego przypadku algorytm Grahama jest dużo szybszy (Tabela 2 i Wyk. 6).

Dla zbiorów typu C i D za to algorytm Jarvisa wypada dużo lepiej, ponieważ zbióry typu C mogą zawierać maksymalnie 8 punktów w otoczce, a zbióry typu D maksymalnie 4 (wierzchołki), czyli złożoność jest rzędu O(n), gdzie w algorytmie Grahama to O(nlogn) (Tabele 3-4 I Wyk. 7-8).

Również widzimy, że wybór współrzędnych punktów w zbiorach nie wpływa na czasy działania algorytmów (współrzędne wierzchołków prostokątów, środek oraz promień okręgu), tylko ilość analizowanych punktów ma znaczący wpływ na czasy działania.

Na podstawie wykresów 5- 8 możemy stwierdzić, że asymptotyczne złożoności obu algorytmów zgadzają się do przewidzianych we wstępie.