Akademia Górniczo Hutnicza
im. Stanisława Staszica
w Krakowie

Algorytmy Geometryczne

Otoczka wypukła dla zbioru punktów w przestrzeni dwuwymiarowej

Smyda Tomasz

Wiśniewski Jakub

2 stycznia 2024

Spis treści

1 Dokumentacja

1.1 Część techniczna

1.1.1 Użyte biblioteki oraz narzędzia

Implementacje algorytmów zostały napisane w języku `Python`. Główna część projektu znajduje się w dwóch plikach Jupyter Notebook. W pliku o nazwie `main` znajdują się implementacje algorytmów bez wizualizacji oraz przeprowadzone testy wydajnościowe algorytmów na wygenerowanych zbiorach testowych. W pliku o nazwie `visualizations` znajdują się zaimplementowane algorytmy wraz z wizualizacją krokową. W projekcie używaliśmy takich bibliotek jak: `random` - do generowania losowych punktów na płaszczyźnie, `pandas` oraz `numpy` - do przejrzystej prezentacji wyników pomiarów czasowych dla algorytmów, `Seaborn` - do wygenerowania wykresów dotyczących czasów działania, `time` - użyliśmy funkcji `perf_counter` do porównania czasów działania algorytmów. Do zaprezentowania wizualizacji krokowej algorytmów użyliśmy narzędzia `pyplot` z biblioteki `matplotlib`.

1.1.2 Funkcje pomocnicze

Funkcje pomocnicze użyte w programie:

- `det(a, b, c)` - zwraca wartość wyznacznika dla podanych na wejściu punktów `a`, `b`, `c`
- `points_orientation(a, b, c, eps = 0)` - zwraca położenie punktu `c` względem prostej przechodzącej przez punkty `a` i `b`; 1 jeżeli `c` leży po lewej stronie, -1 jeżeli po prawej, 0 jeżeli na prostej
- `points_distance_square(a, b)` - zwraca odległość (metryka euklidesowa) pomiędzy punktem `a` i `b`
- `generate_uniform_points(left, right, n)` - funkcja generuje `n` losowych punktów, których wartości współrzędnych są z przedziału `(left, right)`
- `generate_circle_points(O, R, n)` - funkcja generuje `n` jednostajnie położonych punktów na okręgu o środku w punkcie `O` i promieniu `R`
- `generate_rectangle_points(a, b, c, d, n)` - funkcja generuje `n` punktów rozłożonych losowo na bokach prostokąta o wierzchołkach w punktach `a`, `b`, `c`, `d`. Ważne jest że wielokąt zadajemy zgodnie z kierunkiem przeciwnym do ruchu wskazówek zegara, wierzchołek `a` znajduje się w lewym dolnym rogu.
- `generate_square_points(a, b, c, d, axis_n, diag_n)` - funkcja generuje losowo `axis_n` punktów na dwóch bokach kwadratu oraz `diag_n` punktów na dwóch przekątnych kwadratu zadanego przez punkty `a`, `b`, `c`, `d`. Ważne jest że kwadrat zadajemy zgodnie z kierunkiem przeciwnym do ruchu wskazówek zegara, wierzchołek a znajduje się w lewym dolnym rogu.

1.1.3 Specyfikacja

Porównania czasowe algorytmów zostały wykonane na systemie operacyjnym Windows 10 Pro oraz na procesorze Intel Core I5-7500 3.40 GHz.

1.2 Część użytkowa

Aby skorzystać z programu należy uruchomić plik `main` lub plik `visualizations` (jeżeli chcemy zobaczyć wizualizację działania algorytmów) przy pomocy narzędzia Jupyter Notebook oraz po kolei uruchamiać komórki, które zawierają kod. Wszystkie algorytmy przyjmują na wejściu listę n punktów w postaci $[(x_1, y_1), (x_2, y_3), ..., (x_n, y_n)]$, dla których wyznaczona ma być otoczka. Algorytmy zwracają listę punktów, które należą do otoczki w postaci $[(x_{h_1}, y_{h_1}), (x_{h_2}, y_{h_3}), ..., (x_{h_m}, y_{h_m})]$. Algorytmy z wizualizacją dodatkowo wyświetlają animację GIF, która pokazuje w jaki sposób dany algorytm realizuje znajdywanie otoczki wypukłej.

2 Sprawozdanie

2.1 Opis projektu

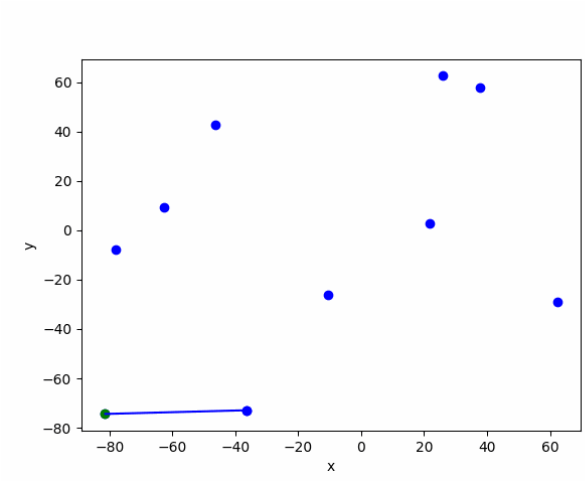
Celem projektu było zaimplementowanie oraz porównanie czasu działania algorytmów wyznaczania otoczki wypukłej dla zbioru punktów na płaszczyźnie. Algorytmy, które zaimplementowaliśmy to:

- Algorytm Grahama
- Algorytm Jarvisa
- Algorytm Dziel i rządź
- Algorytm Chana
- Algorytm Przyrostowy
- Algorytm Quickhull
- Algorytm Górnej i dolnej otoczki

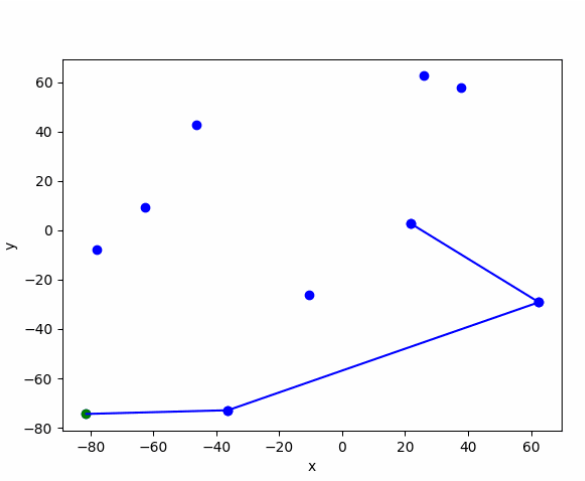
2.2 Przebiegi algorytmów

Poniżej przedstawiono dla każdego algorytmu po 4 klatki z wizualizacji krokowej.

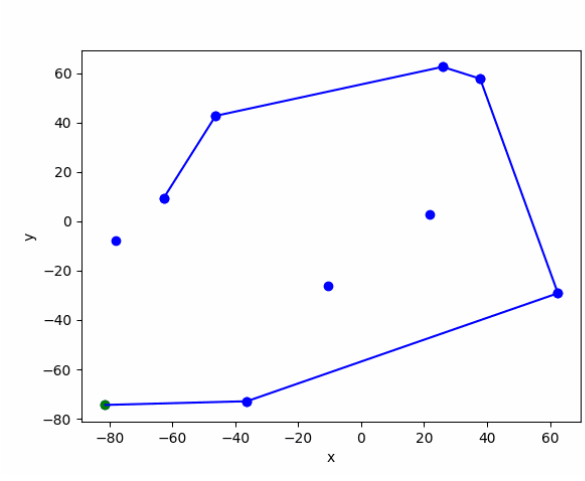
2.2.1 Algorytm Grahama



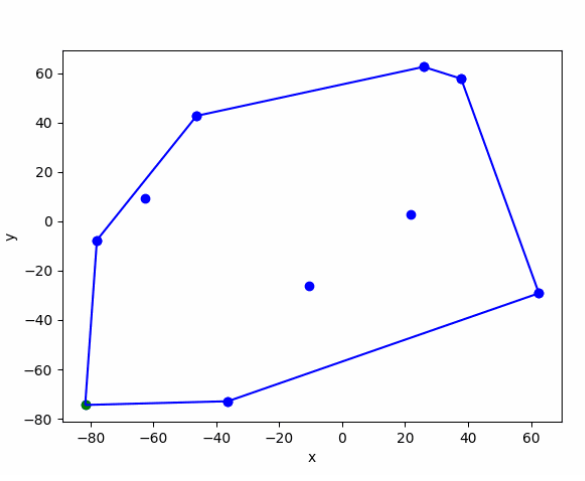
Rysunek 1



Rysunek 2

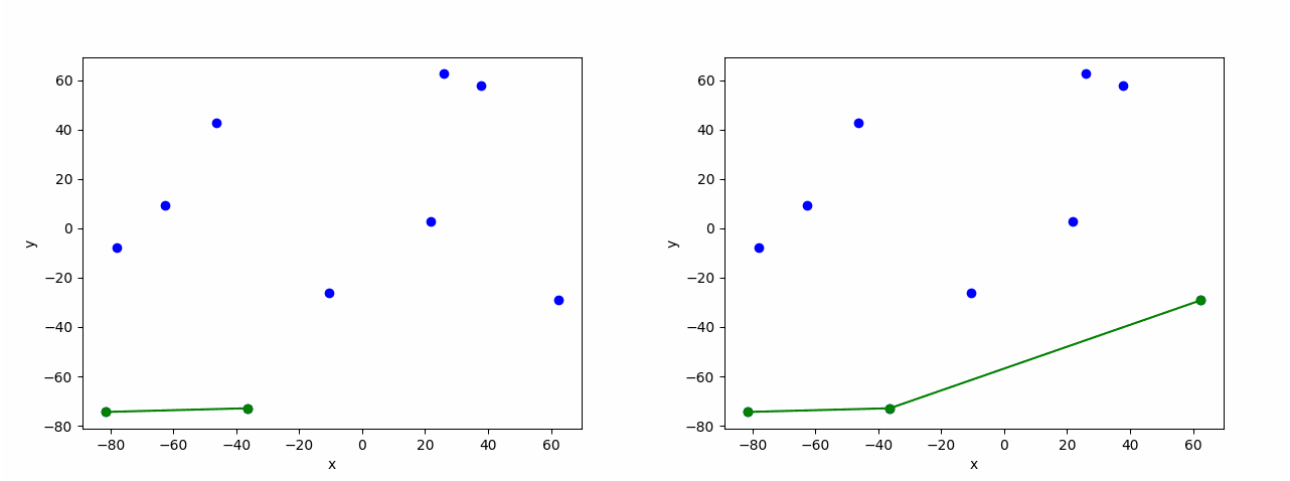


Rysunek 3



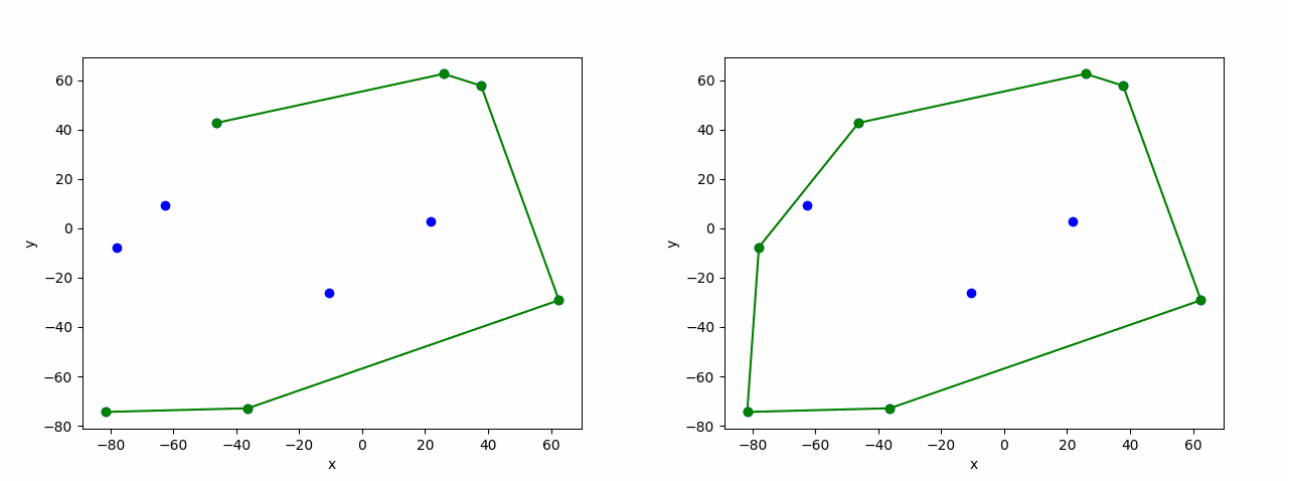
Rysunek 4

2.2.2 Algorytm Jarvisa



Rysunek 5

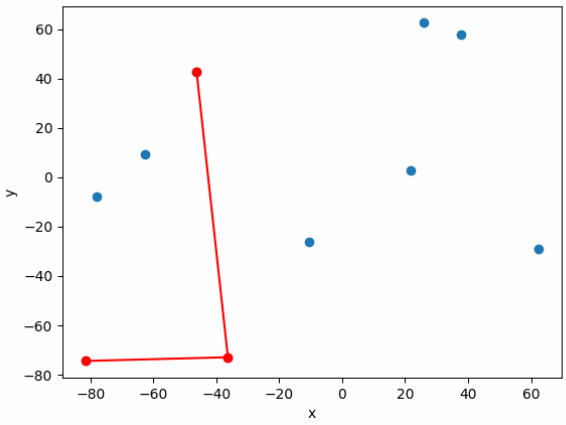
Rysunek 6



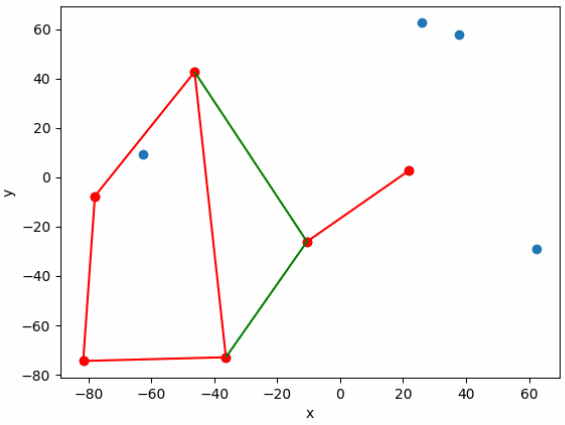
Rysunek 7

Rysunek 8

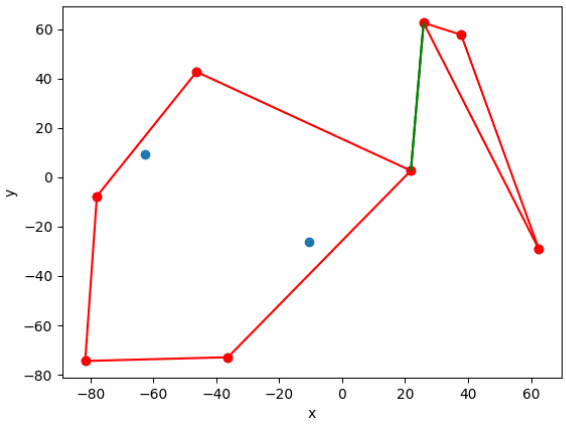
2.2.3 Algorytm dziel i rządź



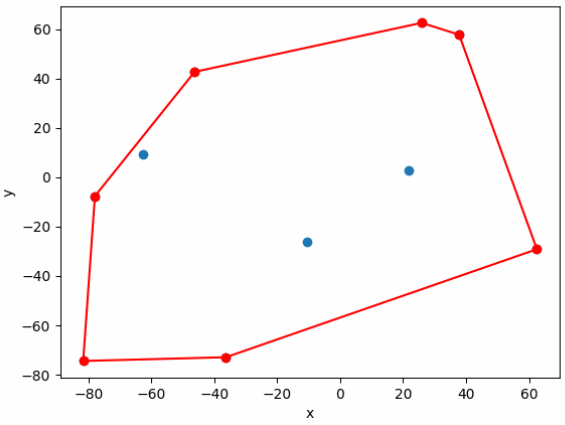
Rysunek 9



Rysunek 10

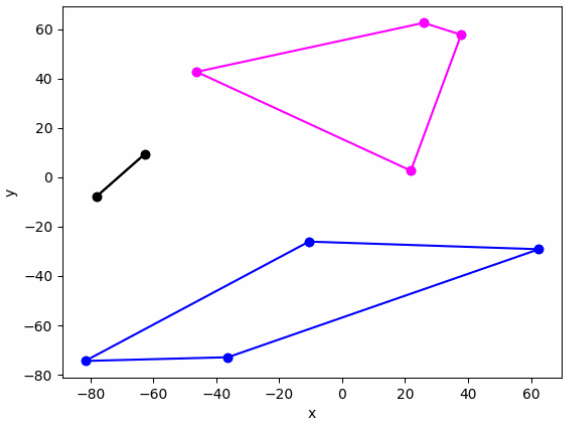


Rysunek 11

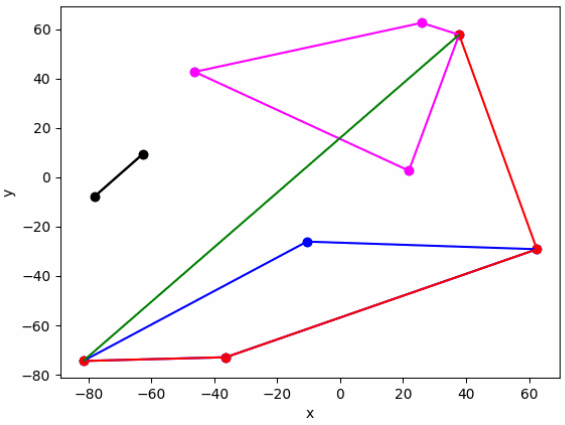


Rysunek 12

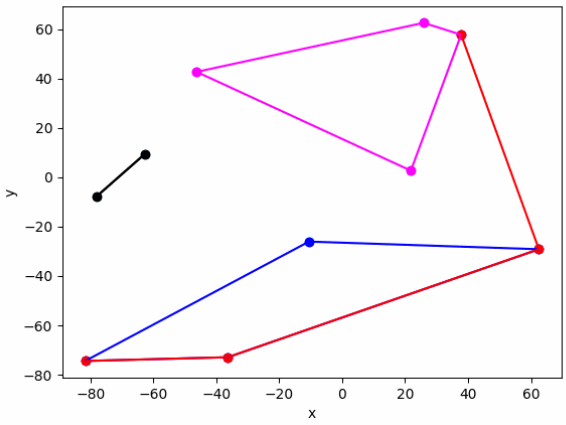
2.2.4 Algorytm Chana



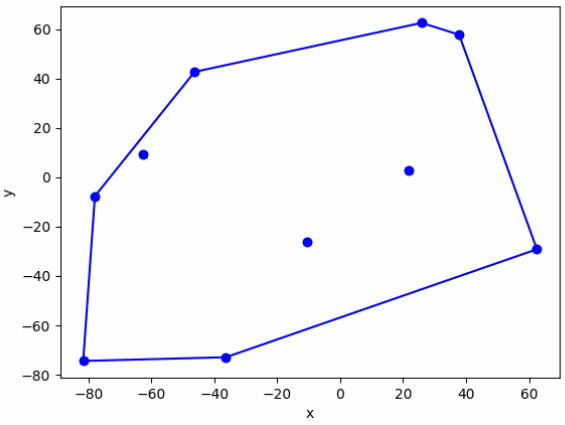
Rysunek 13



Rysunek 14

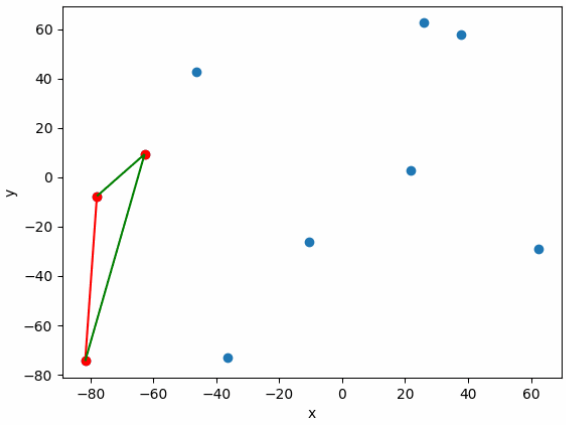


Rysunek 15

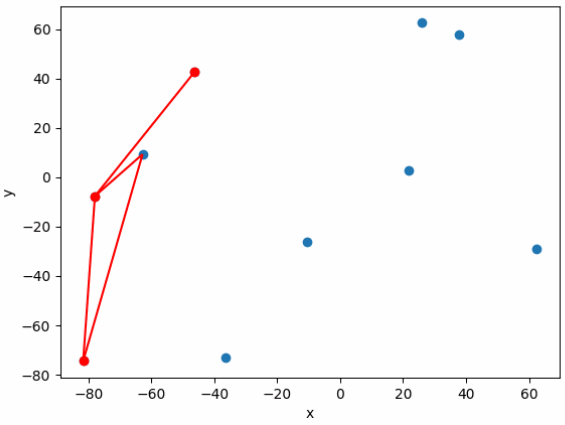


Rysunek 16

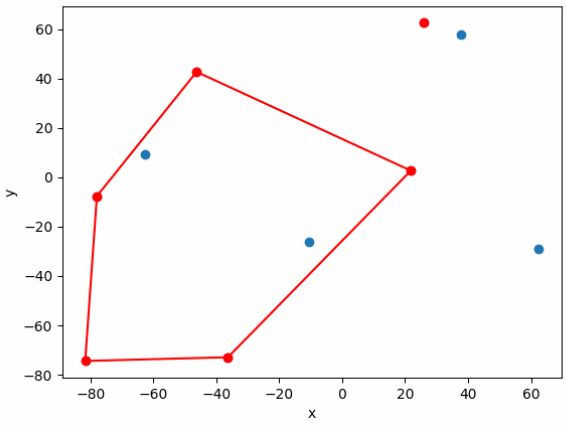
2.2.5 Algorytm przyrostowy



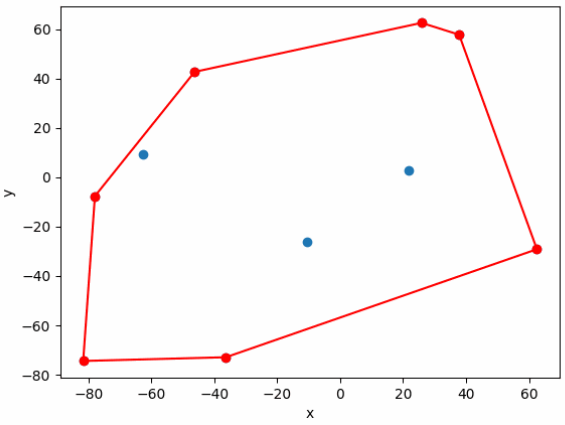
Rysunek 17



Rysunek 18

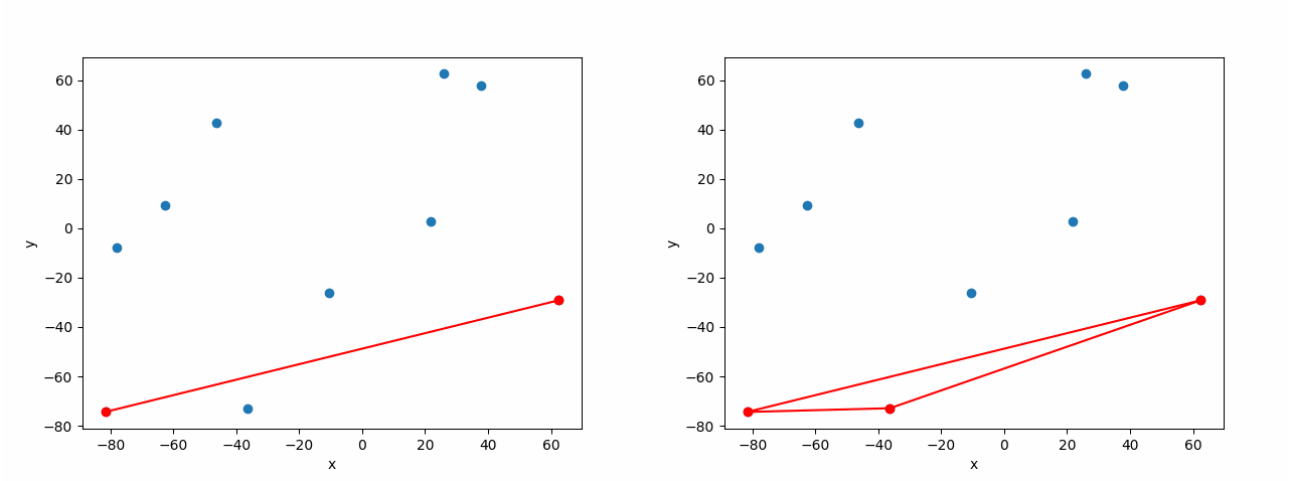


Rysunek 19



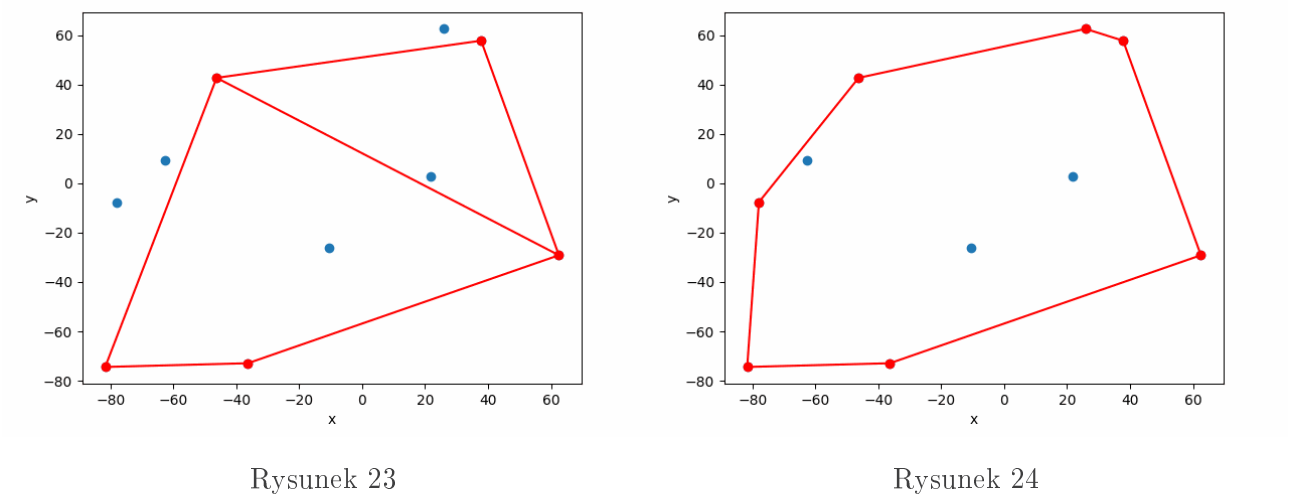
Rysunek 20

2.2.6 Algorytm quickhull



Rysunek 21

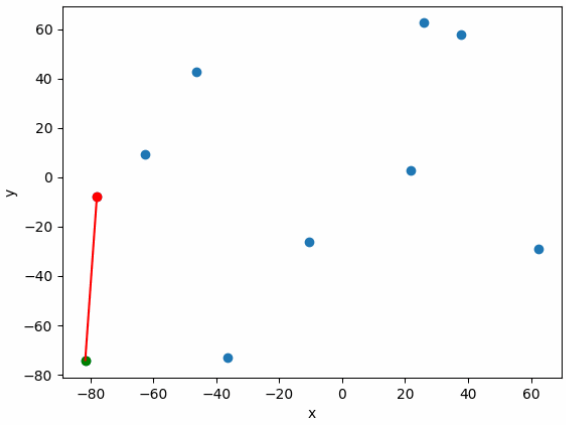
Rysunek 22



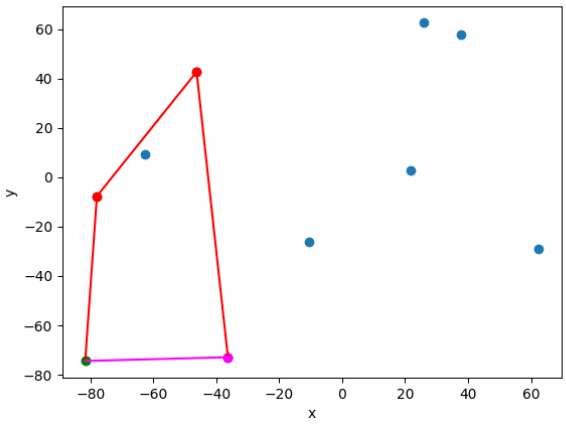
Rysunek 23

Rysunek 24

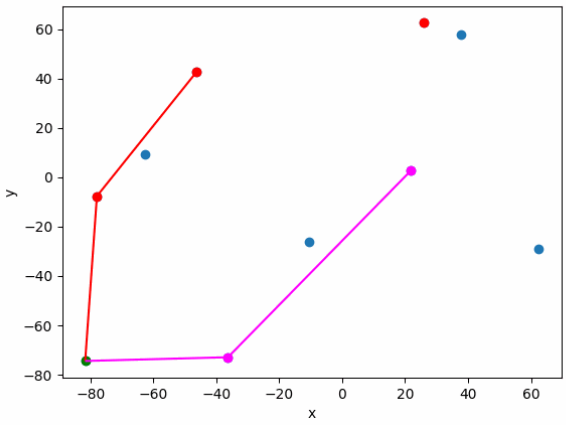
2.2.7 Algorytm górnej i dolnej otoczki



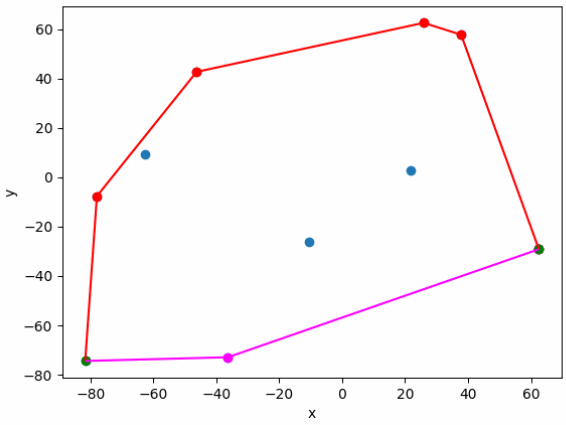
Rysunek 25



Rysunek 26



Rysunek 27



Rysunek 28

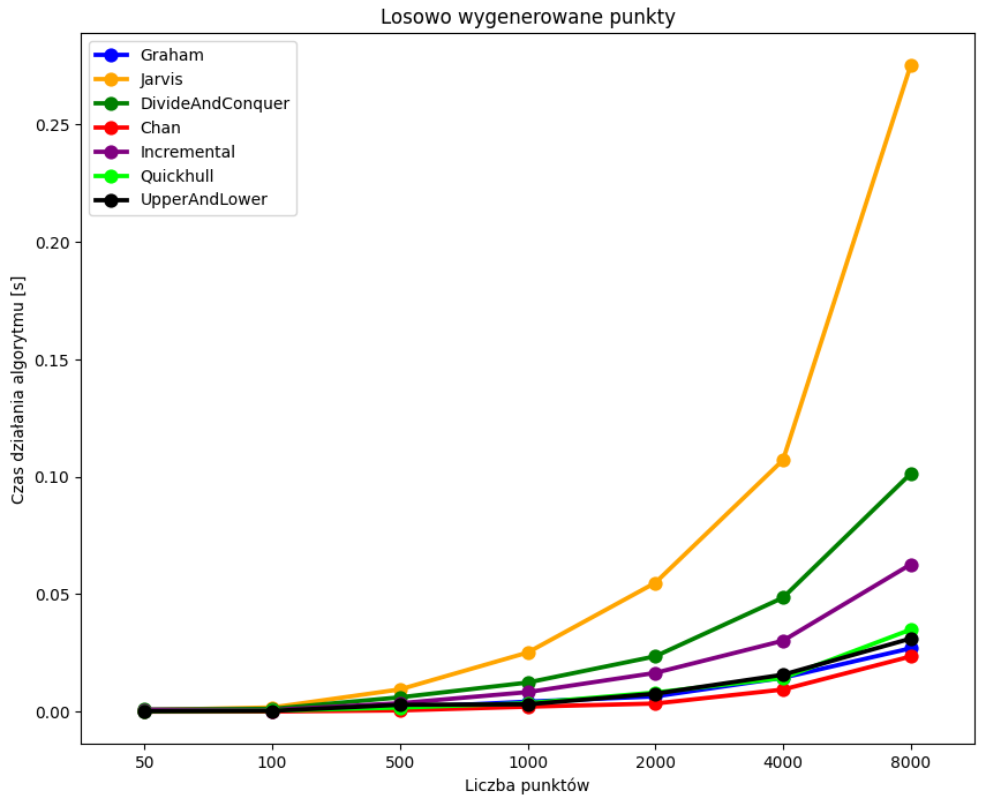
2.3 Porównanie czasowe

Do porównania czasowego algorytmów zostały wyznaczone 4 rodzaje zbiorów wejściowych:

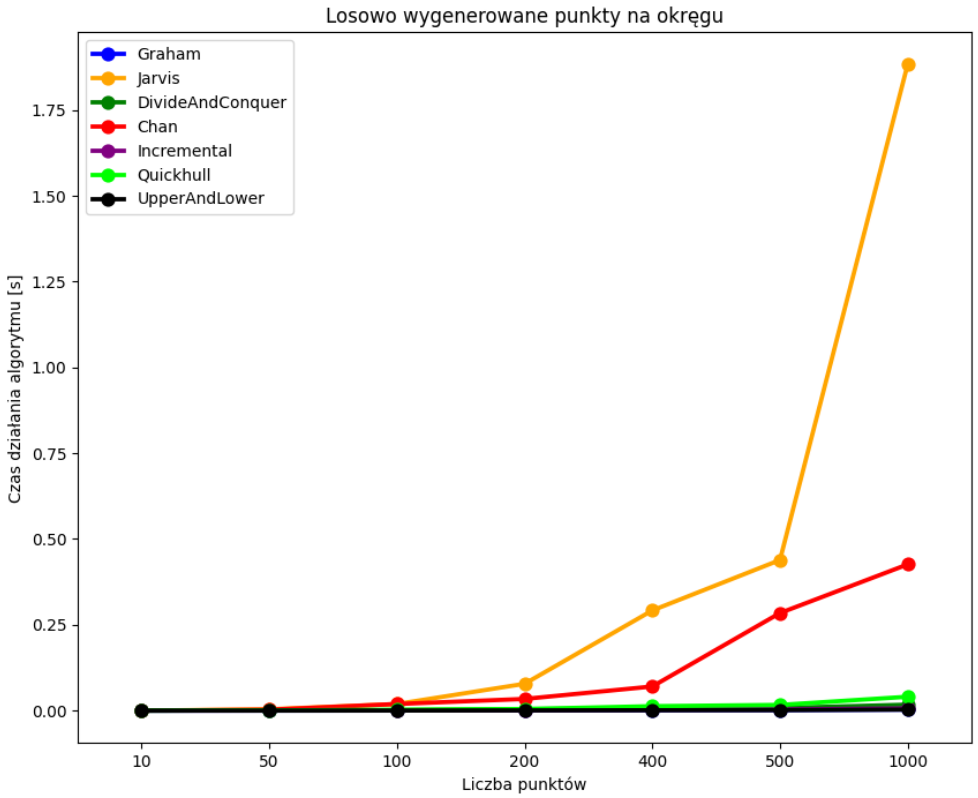
- range - punkty losowo rozmieszczone na zadanym przedziale
- circle - punkty jednostajnie rozmieszczone na okręgu
- rectangle - punkty losowo rozmieszczone na bokach prostokąta
- square - punkty losowo rozmieszczone dwóch bokach kwadratu oraz dwóch jego przekątnych

Liczba punktów	Nazwa zbioru	Algorytm						
		Grahama	Jarvisa	Dziel i rządź	Chana	Przyrostowy	Quickhull	Górnej i dolnej otoczki
50	range_1	0.000187	0.000671	0.000884	0.000023	0.000842	0.000225	0.000201
100	range_2	0.000343	0.001806	0.001214	0.000076	0.000701	0.000434	0.000358
500	range_3	0.001665	0.009491	0.006115	0.000551	0.003615	0.001738	0.002944
1000	range_4	0.004279	0.025185	0.012408	0.002117	0.008334	0.003606	0.003144
2000	range_5	0.006494	0.054890	0.023585	0.003477	0.016525	0.008163	0.007669
4000	range_6	0.014444	0.107140	0.048545	0.009436	0.030216	0.014355	0.015743
8000	range_7	0.026998	0.275150	0.101342	0.023528	0.062704	0.034950	0.031131
10	circle_1	0.000052	0.000172	0.000116	0.000125	0.000095	0.000083	0.000030
50	circle_2	0.000182	0.003938	0.000656	0.002797	0.000318	0.000684	0.000159
100	circle_3	0.000315	0.019356	0.001365	0.019336	0.000732	0.002738	0.000355
200	circle_4	0.000704	0.078274	0.004613	0.034128	0.001626	0.004794	0.000737
400	circle_5	0.001455	0.291913	0.005991	0.070417	0.002892	0.012973	0.001256
500	circle_6	0.001678	0.438848	0.007662	0.284390	0.003474	0.016693	0.001566
1000	circle_7	0.003030	1.882398	0.017368	0.426035	0.006721	0.040504	0.003137
20	rectangle_1	0.000068	0.001341	0.000288	0.000007	0.000138	0.000171	0.000108
100	rectangle_2	0.000350	0.001183	0.001019	0.000070	0.000494	0.000638	0.000317
200	rectangle_3	0.000603	0.002017	0.001142	0.000186	0.000885	0.000927	0.000553
500	rectangle_4	0.001478	0.007122	0.003843	0.000584	0.002365	0.002359	0.001428
1000	rectangle_5	0.004234	0.011053	0.008562	0.002130	0.004472	0.004975	0.004319
2500	rectangle_6	0.008331	0.030960	0.017936	0.005012	0.014015	0.015076	0.007232
5000	rectangle_7	0.018040	0.065698	0.039142	0.012989	0.028421	0.029130	0.017871
94	square_1	0.000296	0.000558	0.000811	0.000030	0.000536	0.000261	0.000254
204	square_2	0.000582	0.001263	0.001746	0.000117	0.001341	0.001839	0.000658
304	square_3	0.001060	0.001869	0.002382	0.000316	0.001624	0.000848	0.000773
704	square_4	0.002078	0.006784	0.005426	0.000841	0.003792	0.002161	0.004028
2004	square_5	0.006360	0.014236	0.019870	0.003215	0.014128	0.006021	0.007553
3004	square_6	0.009172	0.024642	0.027057	0.005638	0.020012	0.011432	0.011895
4004	square_7	0.017101	0.030599	0.046344	0.011893	0.029795	0.012980	0.013443

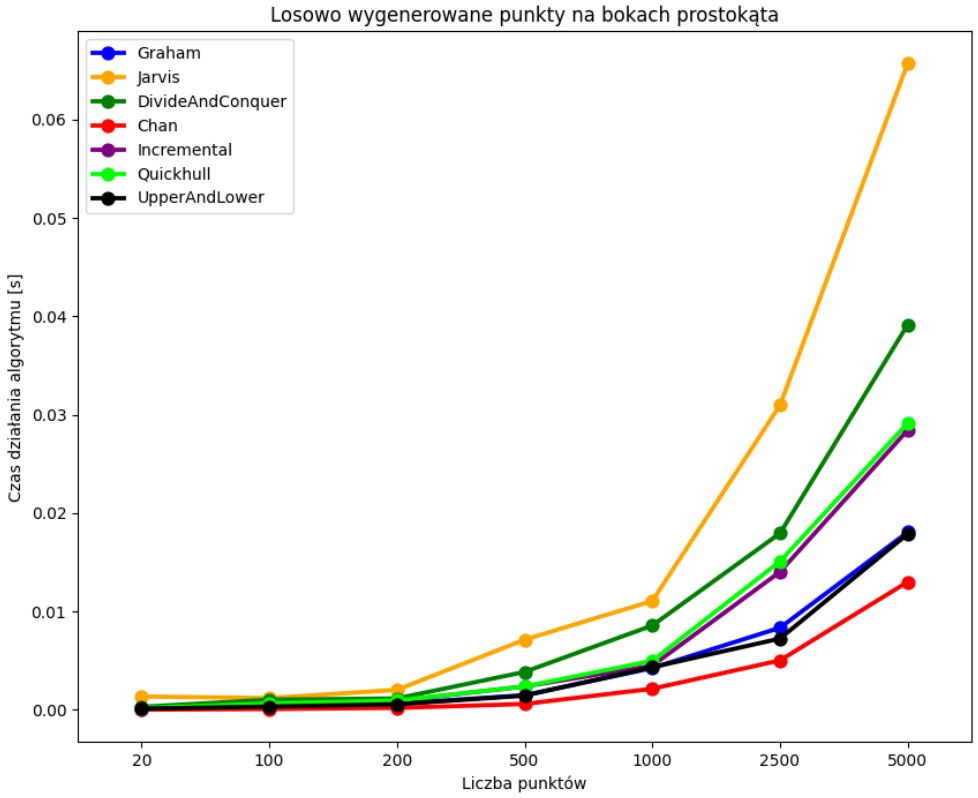
Tabela 1: Wyniki czasowe w sekundach poszczególnych algorytmów dla danych zbiorów



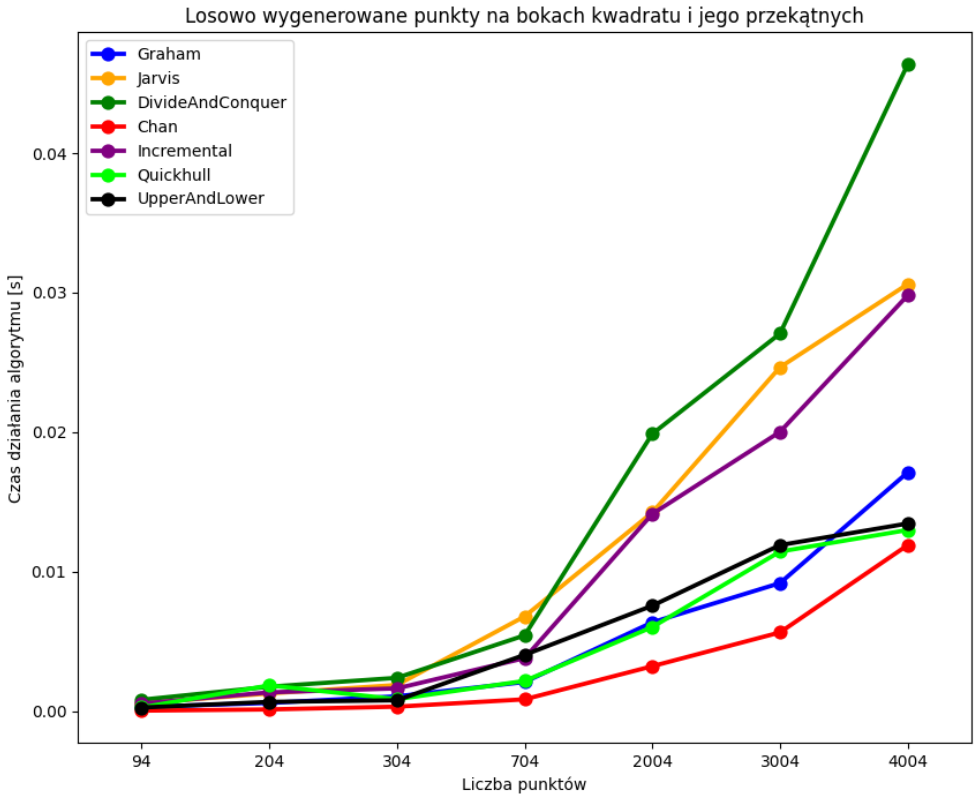
Rysunek 29: Wykres przedstawiający pomiar czasów działania algorytmów dla zbiorów typu range



Rysunek 30: Wykres przedstawiający pomiar czasów działania algorytmów dla zbiorów typu circle



Rysunek 31: Wykres przedstawiający pomiar czasów działania algorytmów dla zbiorów typu rectangle



Rysunek 32: Wykres przedstawiający pomiar czasów działania algorytmów dla zbiorów typu square

Literatura

[1] https://www.geeksforgeeks.org/ (29/12/2023)

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest *Introduction to Algorithms, fourth edition*

[3] T. M. Chan *Optimal output-sensitive convex hull algorithms in two and three dimensions*

[4] Wykład z kursu algorytmy geometryczne na Akademii Górniczo-Hutniczej w Krakowie