

Otoczka wypukła dla zbioru punktów w przestrzeni dwuwymiarowej - dokumentacja do projektu

1. Sprawozdanie

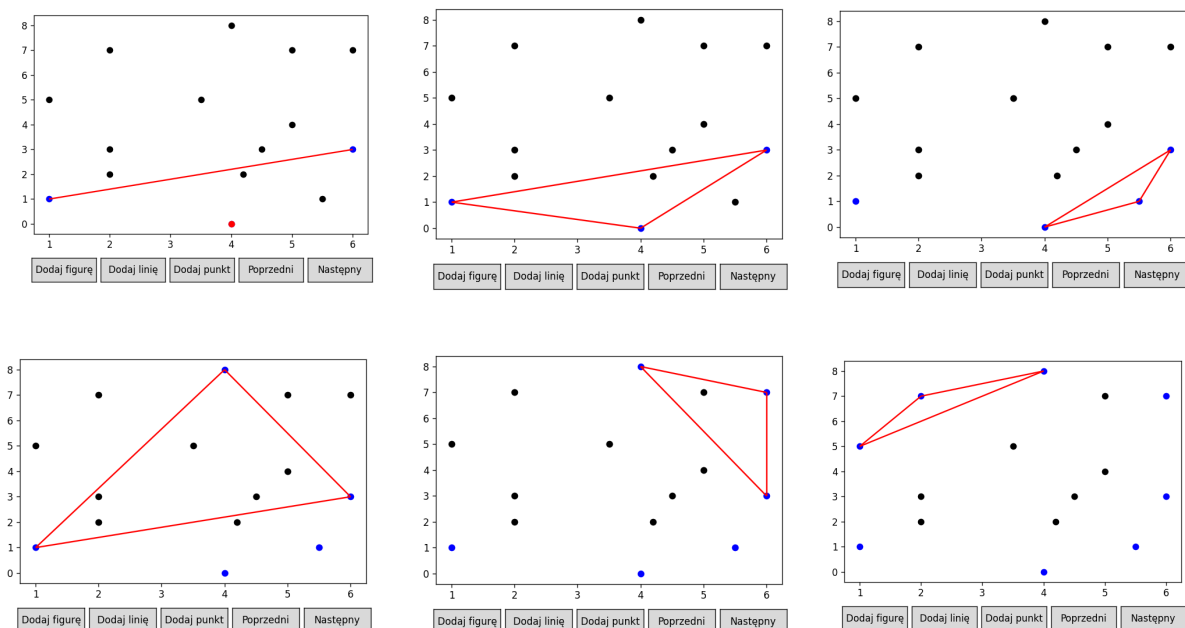
1.1. Wprowadzenie

Celem projektu było zaimplementowanie i przetestowanie wydajności algorytmów wyznaczania otoczki wypukłej dla zbioru punktów w przestrzeni dwuwymiarowej. Zaimplementowane algorytmy to:

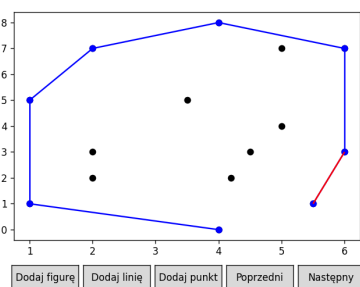
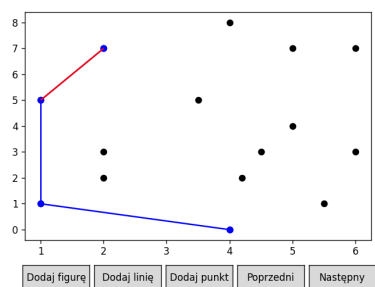
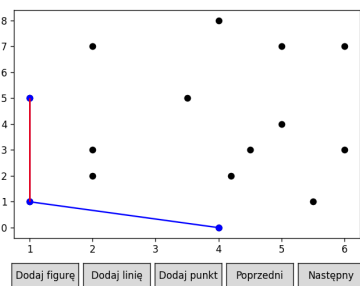
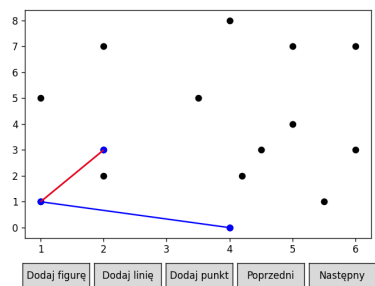
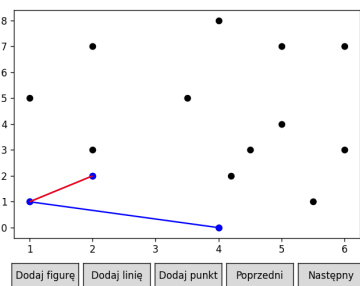
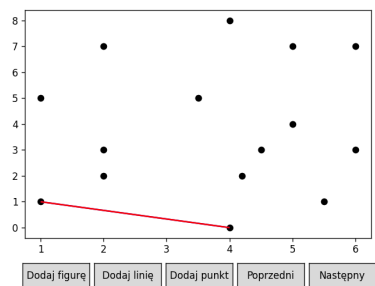
- algorytm Grahama,
- algorytm Jarvis,
- algorytm przyrostowy,
- algorytm górnej i dolnej otoczki,
- algorytm Quickhull,
- algorytm dziel i rządź,
- algorytm Chan'a

1.2. Przykładowe przebiegi działania algorytmów

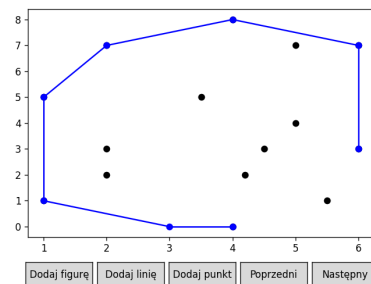
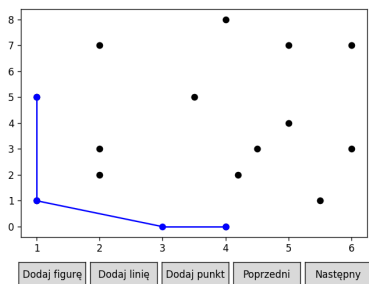
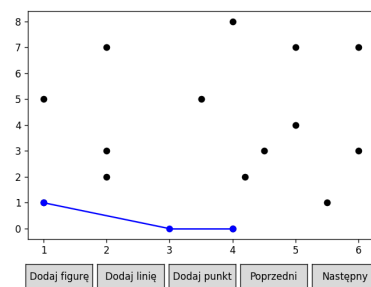
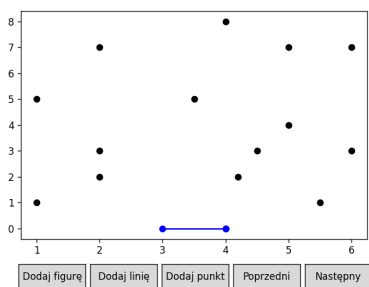
- Algorytm Quickhull



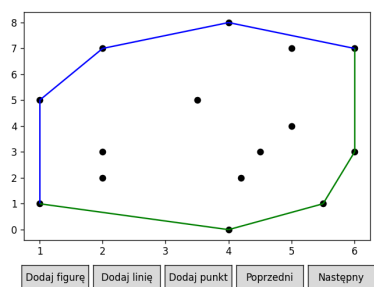
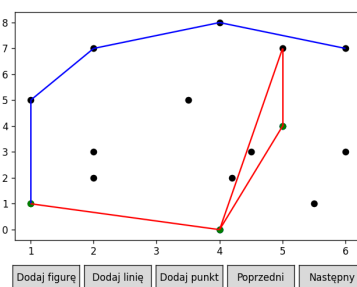
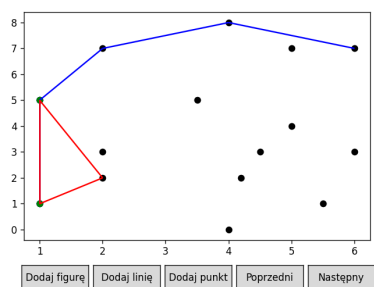
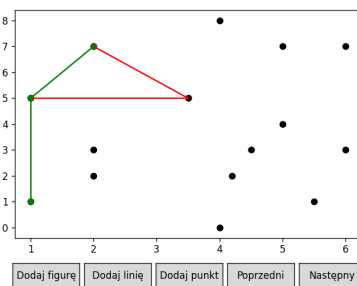
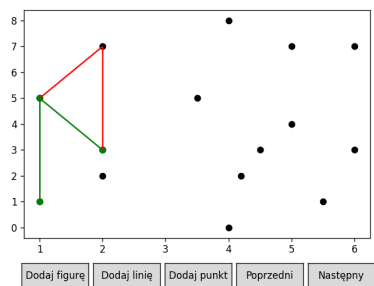
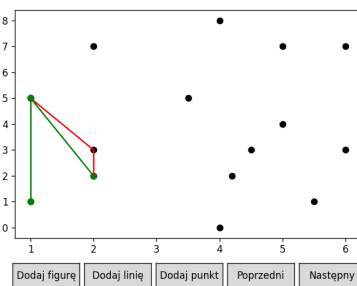
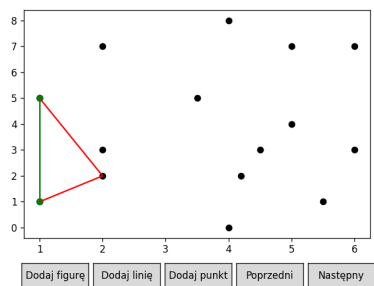
● Algorytm Grahama



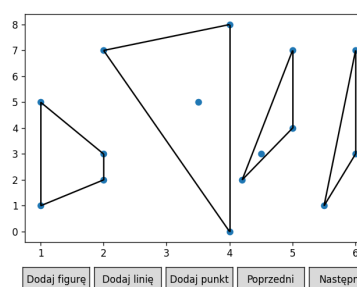
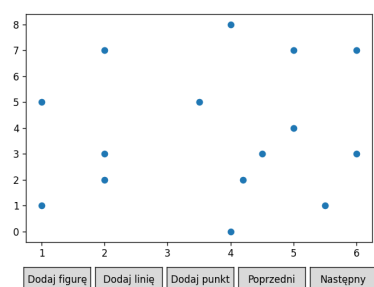
● Algorytm Jarvisa

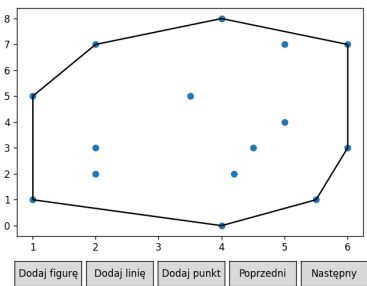
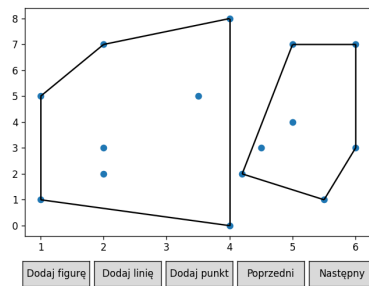


● Algorytm górnej i dolnej otoczki

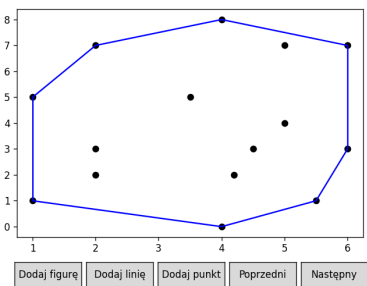
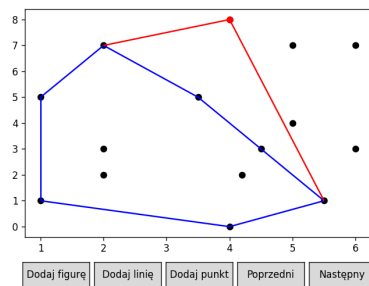
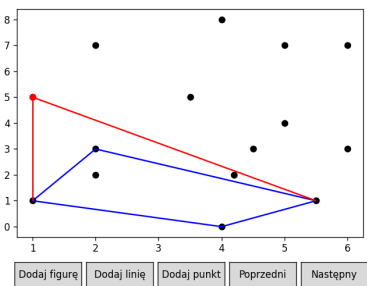
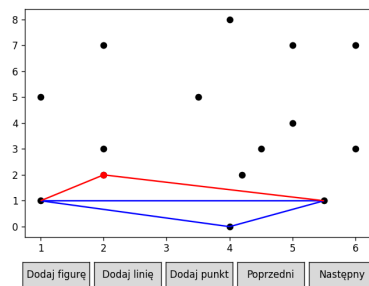
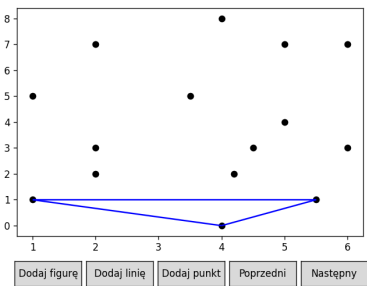
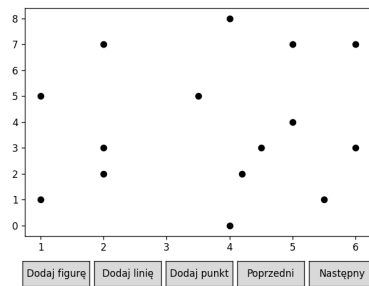


● Algorytm dziel i rządź

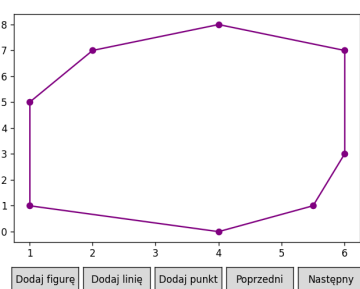
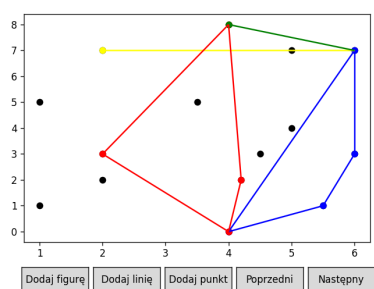
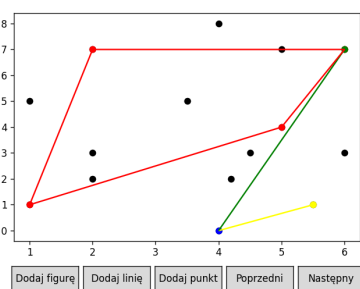
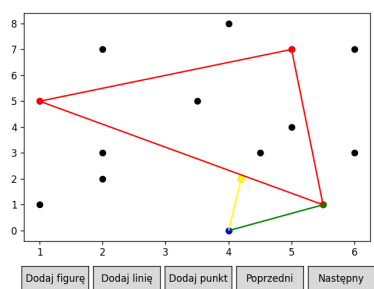
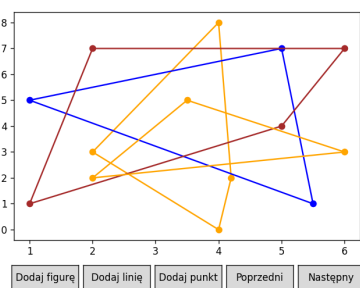
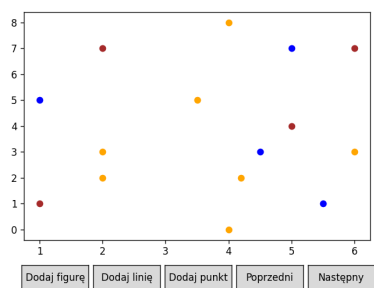




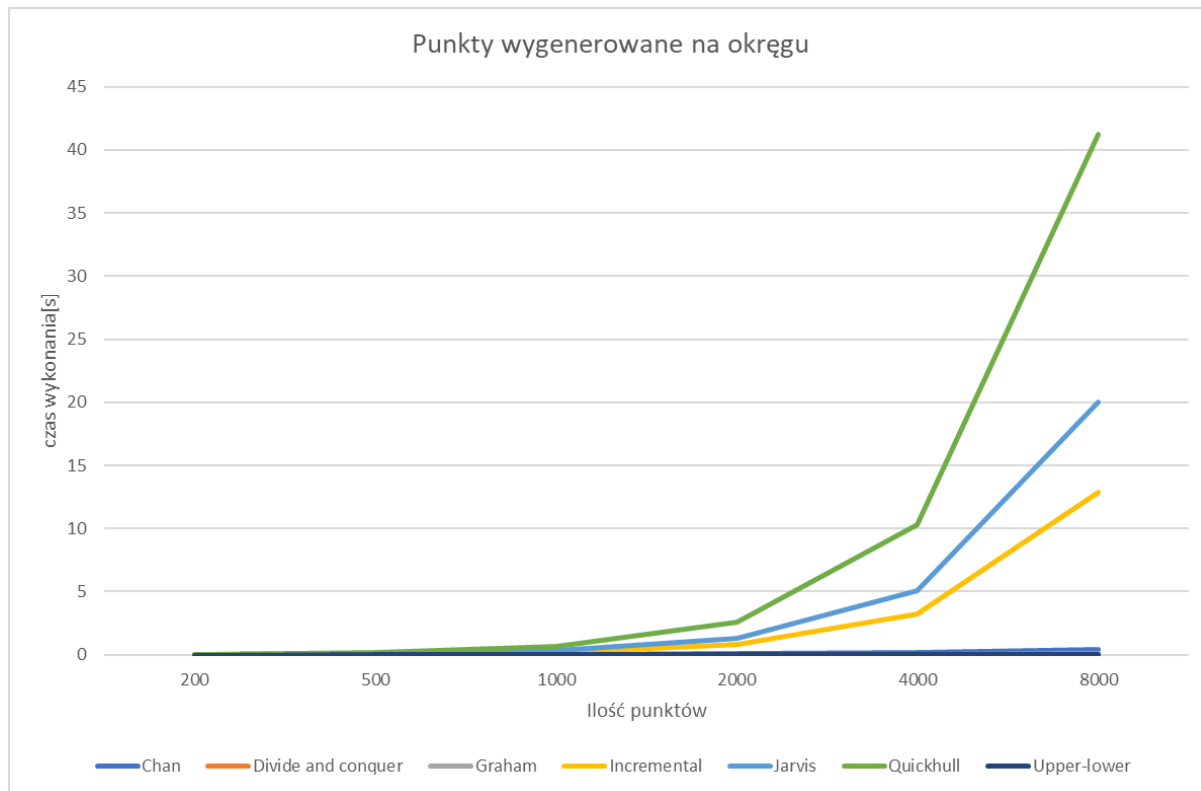
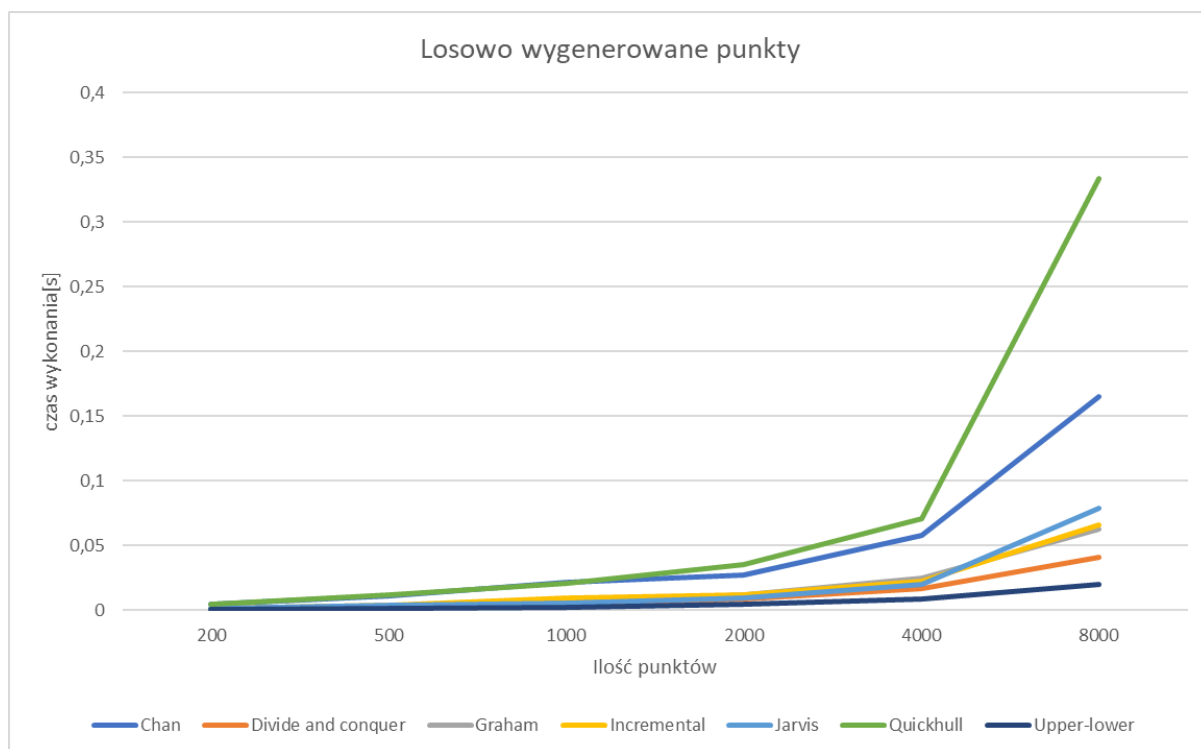
● Algorytm przyrostowy

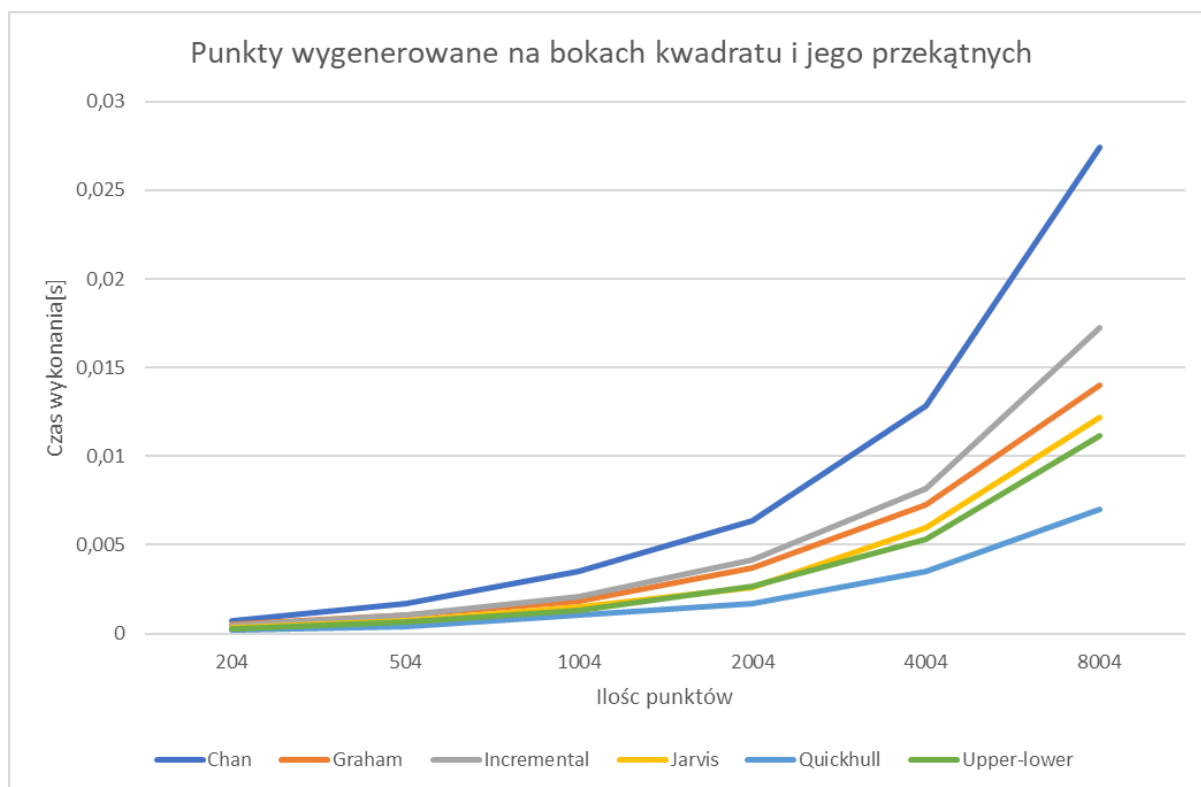
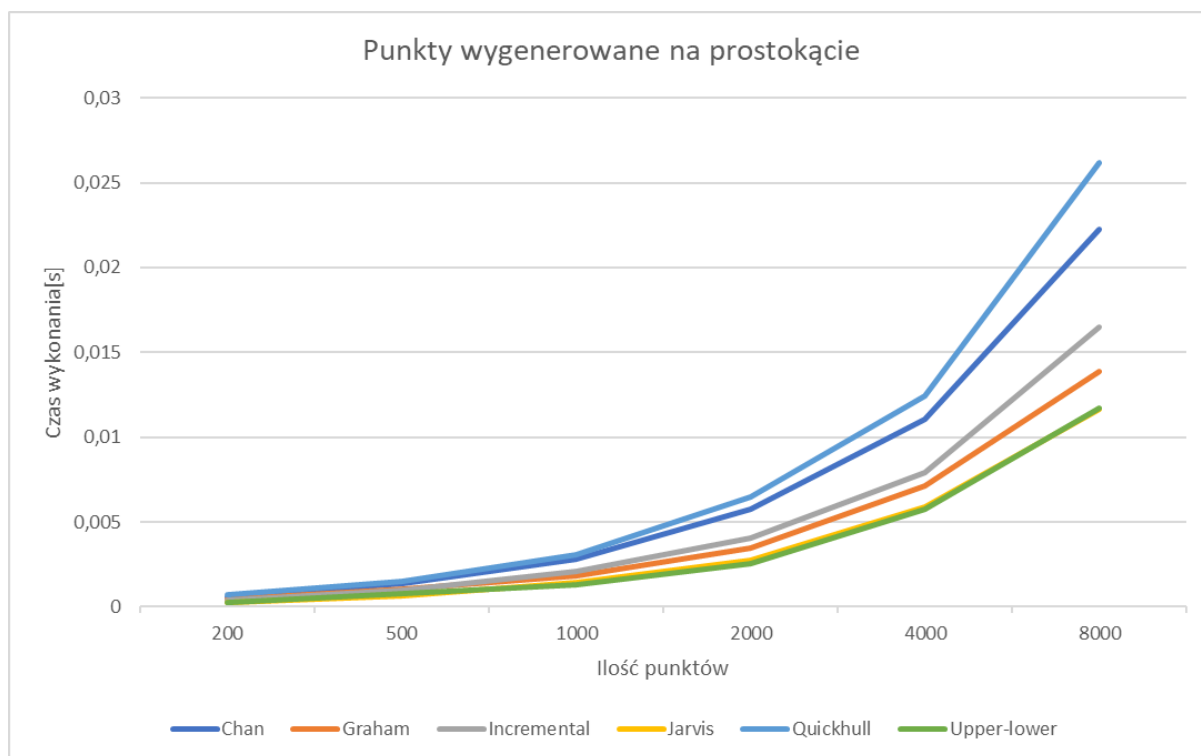


● Algorytm Chan'a



1.3. Wykresy pomiarów czasów działania algorytmów





2. Część techniczna

Implementacje algorytmów zostały napisane w języku Python. Główna część wizualizująca działanie algorytmów znajduje się w pliku Jupyter Notebook, który importuje algorytmy z plików. W projekcie zostały użyte biblioteki takie jak pandas oraz numpy aby w przejrzysty sposób zaprezentować wyniki pomiarów czasów dla algorytmów. Wykresy dotyczące czasów działania zostały zrobione przy pomocy biblioteki Seaborn. Do porównania czasów działania algorytmów została użyta funkcja z biblioteki time - perf counter. Wiedza na temat wszystkich algorytmów była czerpana w większości z wykładu.

Funkcje pomocnicze zastosowane w programie:

- `create_lines(points)` - funkcja jako argument dostaje listę punktów wielokąta i zwraca jego krawędzie w postaci krotek.
- `orientation(a, b, c, epsilon=10 ** (-12))` - funkcja określa położenie punktu c względem wektora $[a, b]$ (po lewej, po prawej lub współliniowy). Robi to na podstawie wyznacznika.
- `random_points_on_the_range(num_of_points, ranges)` - funkcja generuje punkty (w ilości `num_of_points`) o współrzędnych wartości odciętych $[0, \dots, \text{ranges}[0]]$, oraz rzędnych $[0, \dots, \text{ranges}[1]]$.
- `random_points_on_the_circle(num_of_points, center, R)` - funkcja generuje punkty (w ilości `num_of_points`) rozmieszczone równomiernie na okręgu o środku w `center` i promieniu `R`.
- `random_points_on_the_rectangle(num_of_points, vertices)` - funkcja generuje punkty (w ilości `num_of_points`) rozmieszczone równomiernie na krawędziach prostokąta o wierzchołkach z `vertices`.
- `random_points_on_the_square(side_num_of_points, diag_num_of_points, vertices)` - funkcja generuje punkty na dwóch krawędziach kwadratu pokrywającymi się z osiami x i y oraz na przekątnych kwadratu i w jego wierzchołkach.
- `visualisation_tool.py` zawiera narzędzie do wizualizacji udostępnione na kursie.

Program został wykonany na systemie operacyjnym Linux Ubuntu 20.04 oraz na procesorze Intel Core i5-7300HQ 2.50GHz.

3. Część użytkownika

Aby skorzystać z programu należy uruchomić plik "ConvexHullProject" w Jupyter Notebook oraz uruchamiać po kolei komórki, w których zawarty jest kod. Wszystkie algorytmy zawarte w `algorithms_with_visualisation` mogą zostać uruchomione podając im listę punktów, dla których chcemy wyznaczyć otoczkę wypukłą. Algorytm zwróci otoczkę w postaci listy punktów oraz listę scen, które możemy zwizualizować przy pomocy narzędzia graficznego. Podając w argumentach algorytmów `"write_to_file=True"` możemy zapisać wynik wyznaczania otoczki do pliku.