Algorytmy Geometryczne - Sprawozdanie z laboratorium 2

Szymon Tyburczy 420081 Wtorek 15:00-16:30 B Grupa 7

1.Użyte narzędzia oraz krótki opis.

Korzystając ze znanej książki "Wprowadzenie do algorytmów" Thomasa H. Cormena, użyłem opisanego tam algorytmu, który mówi nam o tym, czy przechodząc do sąsiedniego odcinka, skręcamy w lewo, czy w prawo. Zbiory punktów zostały losowo wygenerowane przy użyciu funkcji random.uniform() oraz random.random() z biblioteki numpy. Wykresy oraz wizualizacje zostały przygotowane przy użyciu narzędzia opracowanego przez koło naukowe BIT.

2. Specyfikacja sprzetu na ktorym dokonywane bylo ćwiczenie:

Komputer z systemem windows 11x64
Procesor: 13th Gen Intel(R) Core(TM) i7-10700KF 3.8GHz RAM:
16GB DDR4 3600Mhz
Procesor Graficzny: NVIDIA RTX 3080

3. Cel ćwiczenia oraz teoria stojaca za implementacja

Celem ćwiczenia, jest zaprogramowanie i przetestowanie algorytmu do wyznaczania otoczki wypukłej dla różnych zbiorów punktów na płaszczyźnie. Ćwiczenie służy do zrozumienia i analizy działania algorytmów wyznaczania otoczki wypukłej, ich efektywności na różnych zbiorach danych oraz umożliwienia ich wizualnej interpretacji.

Definicja: Otoczką wypukłą CH(S) dowolnego niepustego zbioru punktów S często nazywamy najmniejszy zbiór wypukły zawierający S.

Podzbiór płaszczyzny Q nazywamy wypukłym, jeśli i tylko jeśli dla dowolnej pary punktów p, q takich, że p, q $\in$ Q, odcinek pq jest całkowicie zawarty w Q.

Do określania otoczki wypukłej będziemy używać dwóch algorytmów: algorytmu Grahama oraz algorytmu Jarvisa.

Algorytm Grahama - wyjaśnienie za nim stojącej logiki oraz specyfikacja implementacji

1) Wybór punktu początkowego:
   Znajdujemy punkt z najniższą współrzędną y (jeśli kilka punktów ma tę samą współrzędną y, wybieramy ten, który ma najniższą współrzędną x). Ten punkt nazywany jest punktem początkowym otoczki wypukłej. W moim kodzie wykonuje to funkcja 'find_min_y()', która przyjmuje zbiór punktów i zwraca szukany punkt.
2) Sortowanie kątowe:
   Sortujemy pozostałe punkty według kąta, jaki tworzą z osią x i punktem początkowym. Jeśli dwa punkty tworzą ten sam kąt z osią x, wybieramy ten, który jest dalej od punktu początkowego. Proces ten odbywa się w kodzie za pomocą funkcji 'sort_points()'.

3) Budowanie otoczki:
   Przechodzimy przez posortowane punkty, używając algorytmu opartego na stosie do budowania otoczki. Dodajemy punkty do stosu, ale jeśli nowy punkt tworzy zakręt w prawo z dwoma ostatnimi punktami na stosie, w takim przypadku usuwamy ostatni punkt ze stosu. Proces ten powtarzamy, aż przejdziemy przez wszystkie punkty. Dzieje się to w moim kodzie w funkcji 'graham_algorithm()'.

4) Złożoność tego algorytmu to O(n log n), gdzie 'n' oznacza liczbę punktów w zbiorze wejściowym.

Złożoność algorytmu Grahama jest wynikiem działania kilku mniejszych algorytmów:

- Sortowania, które wykonuje się w czasie O(nlogn)
- Wyszukiwania punktu o najmniejszej współrzędnej "y" wykonującym się w czasie O(n)
- Budowania otoczki wypukłej O(n)

Algorytm Jarvisa - wyjaśnienie za nim stojącej logiki oraz specyfikacja implementacji

1) Wybór punktu początkowego:
   Wybieramy punkt początkowy, który ma najniższą współrzędną y (jeśli kilka punktów ma tę samą współrzędną y, wybieramy ten z najmniejszą współrzędną x). Ten punkt staje się pierwszym punktem otoczki wypukłej (CH). W moim kodzie, zarówno w algorytmie Jarvisa, jak i w algorytmie Grahama, funkcja 'find_min_y()' przyjmuje zbiór punktów i zwraca odpowiedni szukany punkt.

2) Inicjalizacja otoczki:
   Algorytm zaczyna od dodania punktu początkowego, znalezionego wcześniej za pomocą funkcji 'find_min_y()', do listy punktów otoczki. Punkt ten jest także używany jako punkt odniesienia do porównania z innymi punktami w celu znalezienia następnego punktu otoczki.

3) Wybór kolejnych punktów:
   Algorytm przechodzi iteracyjnie przez pozostałe punkty, wybierając kolejny punkt otoczki, który tworzy obrót w lewo w stosunku do aktualnego punktu. W tym celu funkcja orientation() sprawdza orientację trzech punktów: aktualnego punktu p, sprawdzanego punktu i oraz kandydata na kolejny punkt q. Jeśli orientacja wskazuje na obrót w prawo (wartość 2) lub gdy punkty są współliniowe, ale i jest dalej od p niż q, aktualizowany jest indeks q. Funkcja distance() oblicza odległość w celu porównania punktów współliniowych.

4) Budowanie otoczki:
   Punkt p jest aktualizowany do wartości q po każdej iteracji, a nowy punkt jest dodawany do listy hull. Proces jest kontynuowany, aż algorytm wróci do punktu początkowego, zamykając otoczkę.

5) Złożoność:
   Złożoność czasowa algorytmu Jarvisa wynosi O(nh), gdzie 'n' to liczba punktów w zbiorze wejściowym, a 'h' to liczba punktów na otoczce wypukłej.”

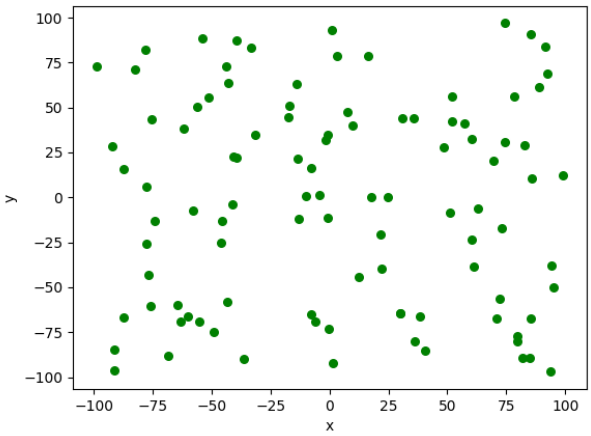
Złożoność algorytmu Jarvisa jest wynikiem działania kilku mniejszych algorytmów:

- Wyszukiwanie punktu o najmniejszej współrzędnej y wykonujące się w czasie O(n).
- Algorytm Jarvisa wykonuje operację wyboru punktów na otoczce wypukłej h razy, a w każdym kroku musi sprawdzić wszystkie n punktów, co daje złożoność O(n) dla każdego kroku.
- Zatem całkowita złożoność algorytmu Jarvisa to, jak już wspomniano, O(nh).
- Warto zauważyć, że w najgorszym przypadku złożoność tego algorytmu może wynieść $O(n^2)$, dlatego w niektórych przypadkach algorytm Grahama może okazać się przytłaczająco szybszy.

4.Plan wykonania ćwiczeń

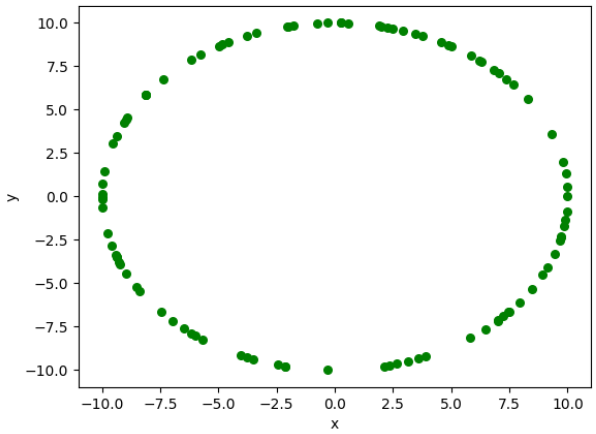
Na początku należało przygotować następujące zbiory:

a) Zbiór A zawierający 100 losowo wygenerowanych punktów o współrzędnych z przedziału [-100, 100].



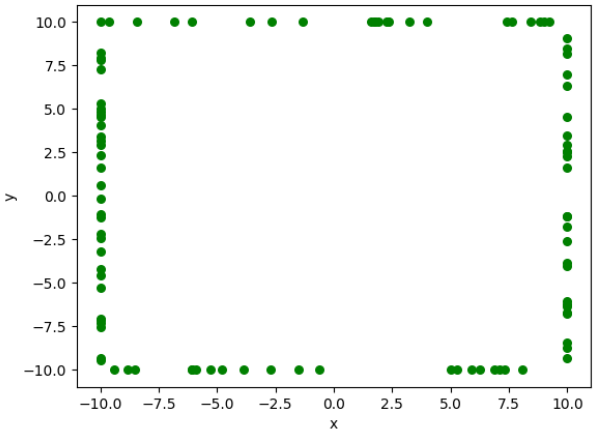
Rys. 4.1 Zbioru A

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



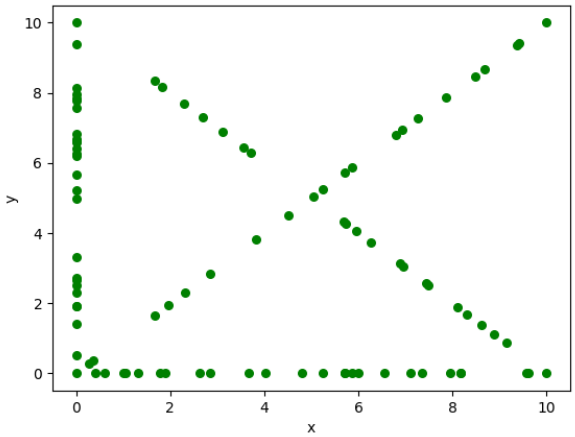
Rys. 4.2 Zbioru B

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



Rys. 4.3 Zbioru C

d) Zbiór 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.



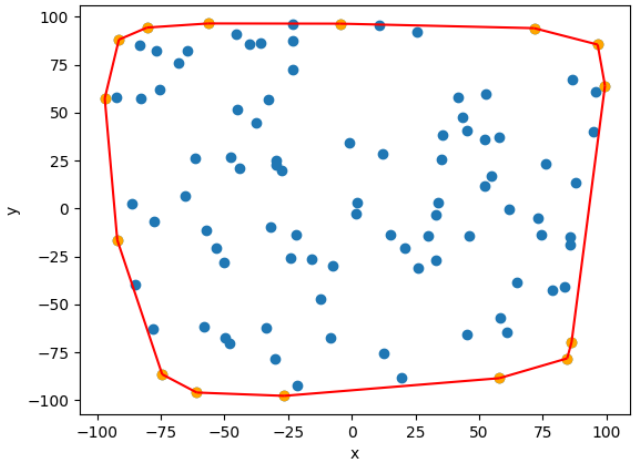
Rys. 4.4 Zbioru D

5.Analiza wyników

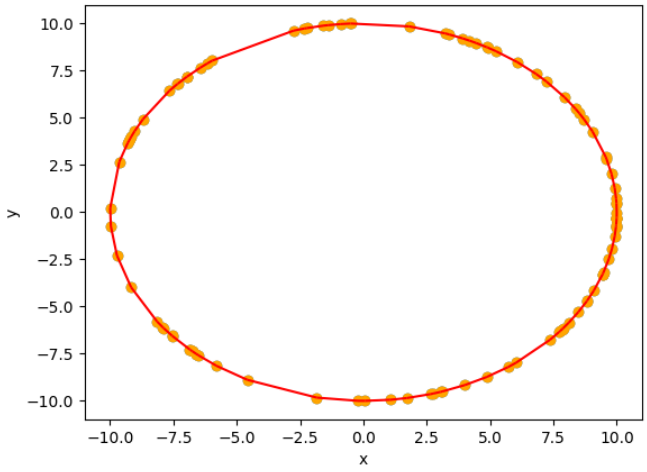
Zbiór punktów	Liczba wierzchołków w otoczce weług algorytmu Grahama	Liczba wierzchołków w otoczce weług algorytmu Jarvisa
A	15	15
B	100	100
C	8	8
D	4	4

Tabela 5.1

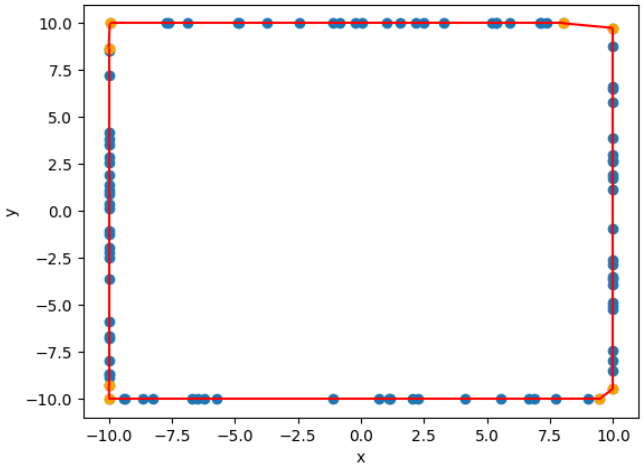
- Tabela 5.1 przedstawia ile punktów opisane wcześniej 2 algorytmy wybierają do stworzenia otoczki wypukłej na zbiorach A, B, C, D.
- Oba algorytmy tworzą dokładnie te same otoczki wypukłe składające się z tych samych punktów.
- Zamieszczone ilustracje ukazują stworzone przez algorytmy otoczki wyupkłe. Kolorem niebieskim oznaczone są punkty w otoczce wypukłej, pomarańczowym punkty na otoczce (należące do niej), natomiast linie łączące punkty na otoczce są koloru czerwonego.



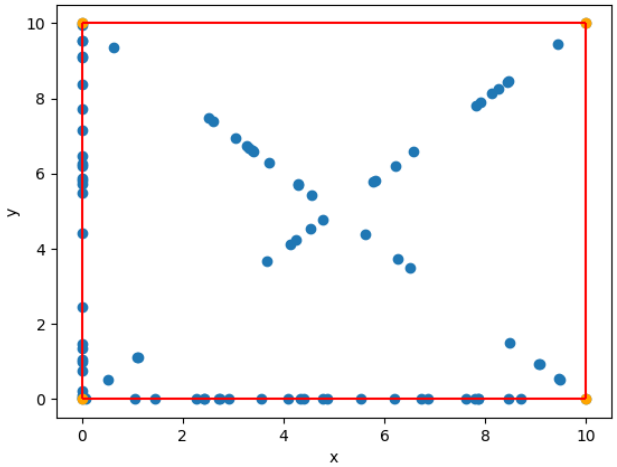
Rys 5.1 Otoczka wypukła zbioru A dla obu algorytmów



Rys 5.2 Otoczka wypukła zbioru B dla obu algorytmów



Rys 5.3 Otoczka wypukła zbioru C dla obu algorytmów



Rys 5.4 Otoczka wypukła zbioru D dla obu algorytmów

5.1 Analiza - Złożoności obliczeniowe

- W celu zobrazowania różnic w czasach obliczania otoczki wypukłej, przygotowałem zestaw danych o większej liczności punktów, aby dokładniej przeanalizować efektywność algorytmów Grahama i Jarvisa. Przypomnę, że złożoność czasowa algorytmu Grahama wynosi O(nlogn), a algorytmu Jarvisa O(nh), gdzie 'n' to liczba punktów w zbiorze, a 'h' to liczba punktów należących do otoczki wypukłej. W najgorszym przypadku, gdy h = n złożoność algorytmu Jarvisa staje się kwadratowa, czyli $O(n^2)$.
- Dla zbioru B, w którym wszystkie punkty należą do otoczki wypukłej (zatem h = n), algorytm Jarvisa osiąga złożoność $O(n^2)$. Oznacza to, że przy większych licznościach punktów czas działania tego algorytmu gwałtownie wzrasta, co wyraźnie kontrastuje z czasem działania algorytmu Grahama, który dzięki efektywniejszej złożoności O(nlogn) lepiej radzi sobie ze wzrostem liczności danych.
- Przygotowanie zbiorów o różnych licznościach pozwala na dokładniejsze porównanie efektywności obu algorytmów i lepsze zrozumienie wpływu struktury danych na ich działanie.
- Aby lepiej uchwycić różnice w czasach obliczeń, zwiększyłem liczność zbiorów punktów. Dotychczas analizowałem dane o liczności 100 punktów, ale teraz rozszerzam zbiory do większych rozmiarów: n = [50000, 100000, 250000, 500000].

o Zbiór A to n losowych liczb z przedziału [-2000, 2000]

o Zbiór C zawiera n losowo wygenerowanych punktów leżących na bokach prostokąta o wierzchołkach (-300, 50), (-300,-200), (100,50), (100,-200).

o Zbiór D zawiera wierzchołki kwadratu (0, 0), (200, 0), (0, 200), (200, 200)
oraz punkty wygenerowane losowo w sposób następujący: po x punktów na dwóch bokach kwadratu leżących na osiach i po y punktów na przekątnych kwadratu, gdzie x = 0.3*n - 1
oraz y = 0.2*n - 1.

- Zgdonie z wcześniejszą analizą sensownie spodziewać się, że największe różnice w czasach obliczeń wystąpią w przypadku zbioru B, który jest okręgiem.
Dla zbioru B rozszerzam liczność punktów do n = [1000, 5000, 10000, 25000] oraz
zmieniam promień na R = 700.
Środek okręgu nadal pozostaje w punkcie środka układu współrzędnych tzn. (0, 0)

Tabele czasów dla wcześniej rozważanych zbiorów:

Zbiór punktów	Liczność n			
	50000	100000	250000	500000
Nowe A	0.7630s	1.6463s	4.3602s	9.6926s
Nowe C	1.1879s	2.3158s	6.4901s	13.8425s
Nowe D	1.1949s	2.5682s	7.0726s	15.0897s

Tabela 5.1 Czasy wykonania algorytmu Grahama dla poszczególnych moców zbiorów dla nowych zbiorów A, C oraz D.

Zbiór punktów	Liczność n			
	50000	100000	250000	500000
Nowe A	0.4127s	0.9249s	3.1156s	5.0825s
Nowe C	0.0999s	0.2821s	0.6121s	1.2298s
Nowe D	0.0787s	0.1618s	0.5679s	1.1117s

Tabela 5.2 Czasy wykonania algorytmu Jarvisa dla poszczególnych moców zbiorów dla nowych zbiorów A, C oraz D.

Zbiór punktów	Liczność n			
	1000	5000	10000	25000
Nowe B	0.0090s	0.0551s	0.1147s	0.3203s

Tabela 5.3 Czasy wykonania algorytmu Grahama dla poszczególnych moców zbiorów dla zbioru nowego B.

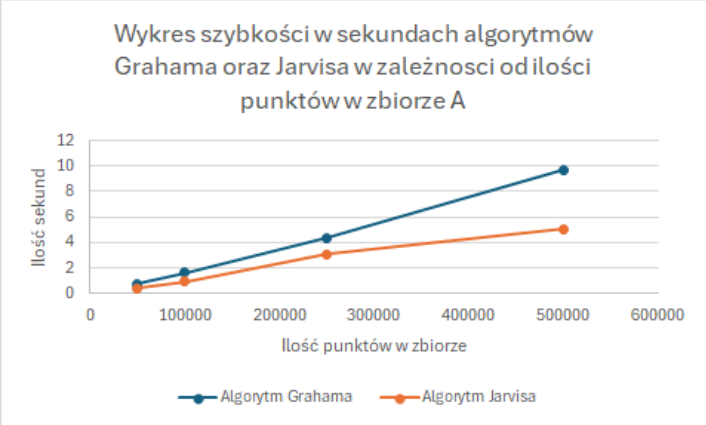
Zbiór punktów	Liczność n			
	1000	5000	10000	25000
Nowe B	0.2654s	6.7759s	27.3280s	174.3573s

Tabela 5.4 Czasy wykonania algorytmu Jarvisa dla poszczególnych moców zbiorów dla zbioru nowego B.

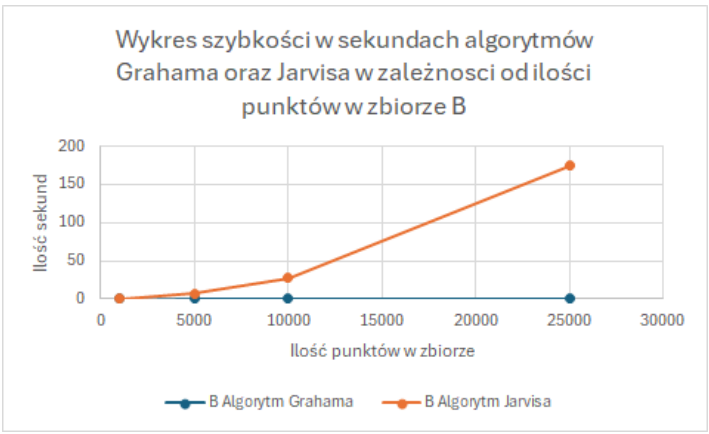
Wnioski dla tabel 5.1 - 5.4

- Dla zbioru A różnice w szybkości działania obu algorytmów są niewielkie. Zbiór A jest najbardziej "neutralnym", gdzie liczba punktów na otoczce jest umiarkowana, co powoduje podobne czasy wykonania obu algorytmów.
- Algorytm Jarvisa działa bardzo nieefektywnie w przypadku zbioru B (okrąg), gdzie każdy punkt należy do otoczki. Złożoność $O(n^2)$ sprawia, że czas działania szybko rośnie z liczbą punktów, co prowadzi do dużych różnic w porównaniu z algorytmem Grahama. Różnice w czasie dla coraz to większych zbiorów są kolosalne - nic dziwnego skoro algorytm się "ukwadratawia".
- Dla zbiorów C i D różnica czasowa wynika z małej liczby punktów na otoczce. Dla zbioru D, gdzie h jest małe, algorytm Jarvisa może działać w czasie O(n), co daje przewagę w takich przypadkach. Zbiór D pokazuje, jak algorytm Jarvisa może stać się bardziej efektywny, gdy liczba punktów na otoczce jest bardzo mała.

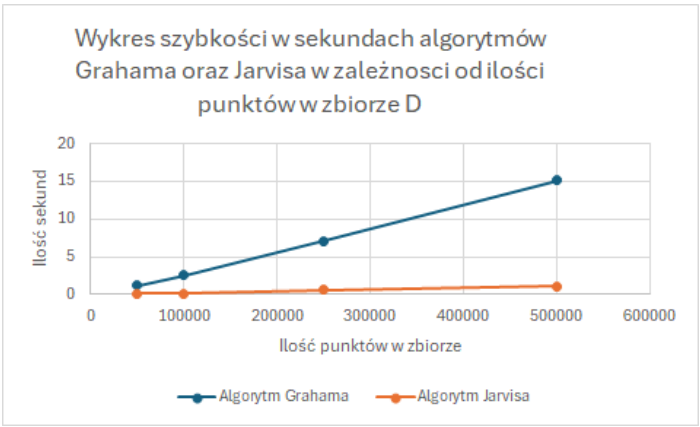
6. Wykresy szybkości algorytmów w zależności od wielkości zbiorów.



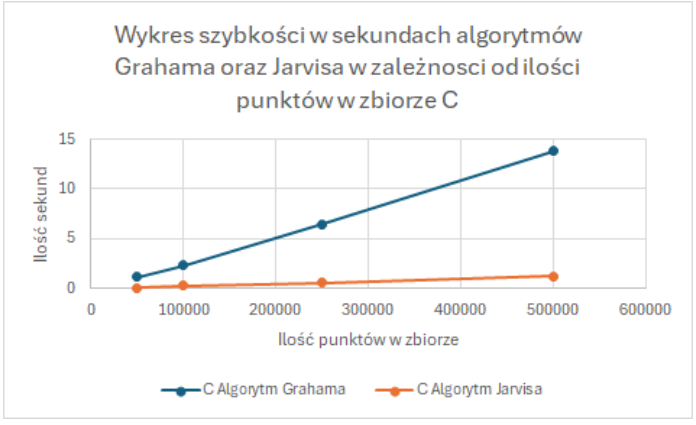
Wykres 6.1
Wykres porównuje czas obliczania otoczki wypukłej dla Algorytmu Grahama oraz Jarvisa dla zbioru A w zależności od jego liczności



Wykres 6.2
Wykres porównuje czas obliczania otoczki wypukłej dla Algorytmu Grahama oraz Jarvisa dla zbioru B w zależności od jego liczności



Wykres 6.3
Wykres porównuje czas obliczania otoczki wypukłej dla Algorytmu Grahama oraz Jarvisa dla zbioru C w zależności od jego liczności



Wykres 6.4
Wykres porównuje czas obliczania otoczki wypukłej dla Algorytmu Grahama oraz Jarvisa dla zbioru D w zależności od jego liczności

7. Ostateczne wnioski

- Algorytmy Grahama oraz Jarvisa dobrze ilustrują znaczenie złożoności obliczeniowej w informatyce oraz przewagę jednego algorytmu nad drugim w zależności od zbioru, na którym się działa.

- Dobór zbiorów był celowy i przemyślany. Zbiór A został wybrany w taki sposób, że liczba punktów należących do otoczki była nieznacznie oddalona od O(log n). To spowodowało, że czasy dla obu algorytmów były porównywalne. Zbiór A był najbardziej zbalansowanym zbiorem.

- Zbiór B został specjalnie dobrany tak, by pokazać wyraźną różnicę na korzyść algorytmu Grahama oraz złożoności O(n log n) nad O(n h). W sytuacji, gdy h = n, algorytm Jarvisa osiąga złożoność kwadratową $O(n^2)$. Dla zbioru B różnice były ogromne, w szczególności dla większych zbiorów, co pokrywało się z przewidywaniami.

- Może również wystąpić sytuacja, w której $\log_2 n > h$. W takim przypadku mamy warunek $n > 2h$, co teoretycznie powinno sprzyjać algorytmowi Jarvisa.
  Dla zbioru C widoczne są zatem różnice na korzyść algorytmu Jarvisa.

- Przygotowany zbiór D powinien zawierać dużą liczbę wierzchołków współliniowych, w związku z tym ogólna złożoność algorytmu Jarvisa powinna być liniowa O(n). W takiej sytuacji algorytm Grahama o złożoności O(n log n) powinien być znacznie wolniejszy, co widać w tabeli 5.4 oraz wykresie 6.4.

- Kluczowym aspektem w realizacji obu algorytmów jest sposób ich implementacji. Optymalizacja kodu, wykorzystanie odpowiednich struktur danych, takich jak stos w przypadku algorytmu Grahama, oraz odpowiednia organizacja algorytmu, mają bezpośredni wpływ na wydajność. Nawet jeśli algorytm teoretycznie posiada określoną złożoność obliczeniową, to różnice w implementacji, takie jak sposób przechowywania danych, mogą znacząco wpłynąć na czas wykonania. Staranność w implementacji jest więc równie istotna, co sam wybór algorytmu, zwłaszcza w przypadkach, gdy dane wejściowe są duże lub wymagają specyficznych optymalizacji.