Obie implementacje działają poprawnie dla wszystkich zbiorów danych.

Graham	1000	10000	100000
A	0,00879	0,113881	1,454202
B	0,00802	0,116406	1,462209
C	0,006607	0,087517	1,181027
D	0,005311	0,053903	0,79104

Jarvis	1000	10000	100000
A	0,013347	0,156888	1,707177
B	0,563325	62,78974	-
C	0,006072	0,057002	0,567672
D	0,00353	0,033237	0,331972

Czasy są podawane w sekundach.

Dane A

W zbiorze A większość wartości były rozmieszczone w dowolnie. W zależności od ilości punktów i rozpiętości zboru na którym losowaliśmy punkty zmieniała się wydajność algorytmu Jarvis ‘a.

W ogólnym przypadku algorytm Grahama był pewniejszym rozwiązaniem zapewniając przy zmiennych parametrach (ilość danych stała) podobne czasy.

Dane B

W zbiorze B wszystkie punkty powinny należeć do otoczki, konsekwencją czego jest osiągnięcie pesymistycznej złożoności czasowej algorytmu Jarvis ‘a $O(n^2)$ przez co obliczenia dla dużych zbiorów są nie praktyczne, gdyż zabierając zbyt dużo czasu. Dla algorytmu Grahama nie ma to większego znaczenia gdyż czynnikiem ograniczający jego prędkość jest głównie sortowanie, które zajmuje $O(nlogn)$.

Dane C i D

W zbiorach C i D algorytm Jarvis ’a wypadł lepiej od algorytmu Grahama. Duża liczba punków współliniowych zapewniła, że ilość punktów na otoczce była niewielka a co za tym idzie algorytm Jarvis ‘a działał w czasie $O(kn)$ dzięki czemu wykonał się szybciej. W przypadku zbioru D można nawet powiedzieć że stała $k = 4$

Dane zostały tak dobrane aby podkreślić silne i słabe strony obu algorytmów.
Zauważyć możemy, że jeśli możemy założyć rozmiar otoczki za niewielki, algorytm Jarvis ‘a będzie lepszym wyborem. Natomiast gdy nie wiemy nic o zbiorze wejściowym lepiej skorzystać z algorytmu Grahama gdyż ma on o wiele lepszą złożoność pesymistyczną.
Dodatkową zaletą algorytmu Grahama jest jego stabilność czasowa co sprawia że będzie lepszy do zastosowania w różnego rodzaju systemach czasu rzeczywistego czy np. grach. Również fakt, że największym ograniczeniem czasowym w algorytmie Grahama jest sortowanie sprawia, że może on posłużyć do szybkiego obliczania nowej otoczki, po dodaniu punktów. Zakładając że punkt względem którego dane były posortowane nie uległ zmianie ponowne uruchomienie algorytmu zajmie
$O(n)$ – wstawienie punktu do posortowanej tablicy
$O(n)$ – wyznaczenie punktów otoczki na nowo