Otoczka wypukła dla zbioru punktów w przestrzeni dwuwymiarowej

Algorytmy Geometryczne

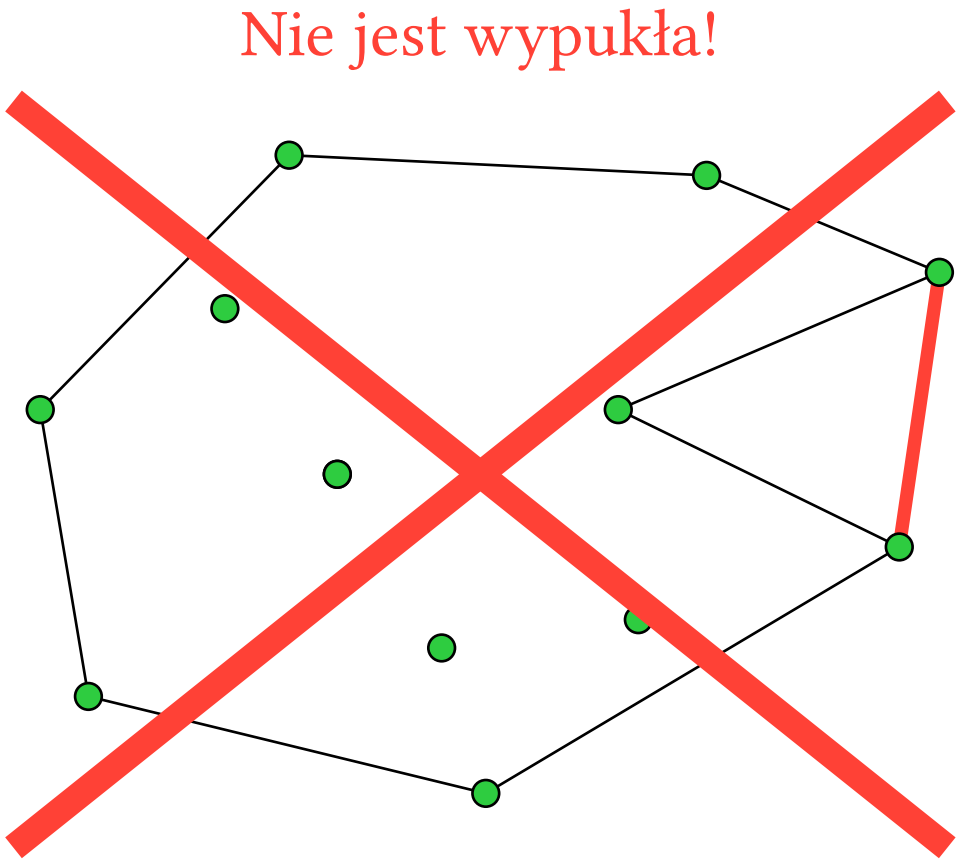
Remigiusz Babiarz · Jakub Własiewicz

Plan prezentacji

1. Otoczka wypukła

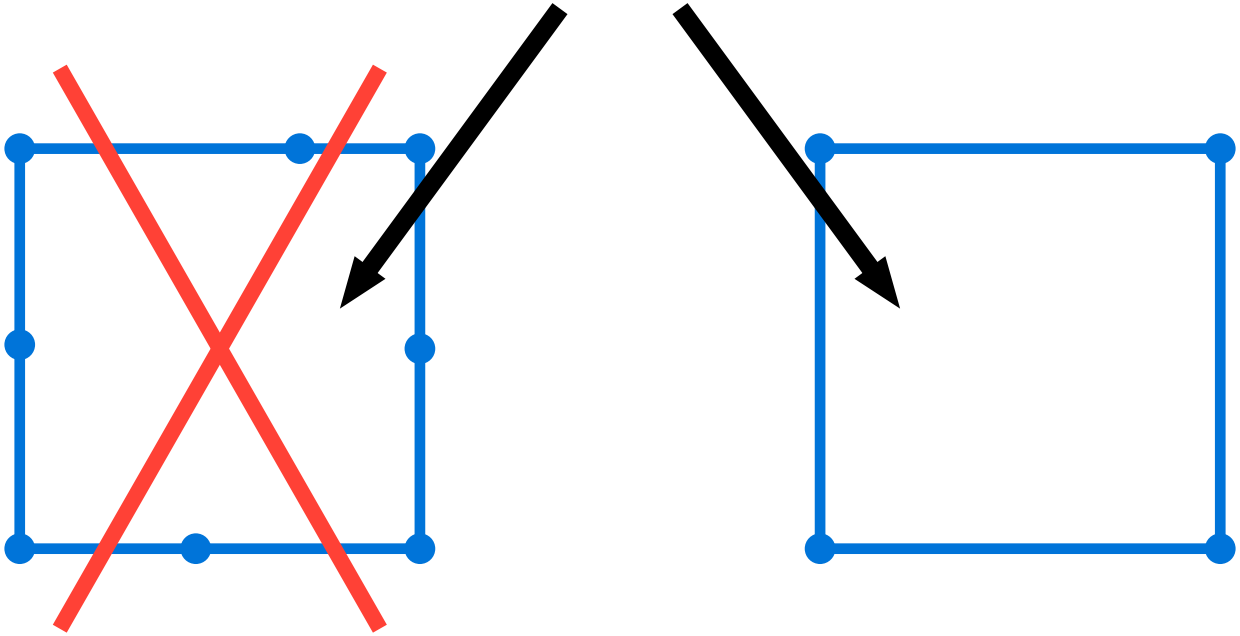
1.1 Definicja

Otoczką wypukłą zbioru punktów nazywamy najmniejszy wielokąt wypukły zawierający ten zbiór. Będziemy oznaczać liczbę punktów zbioru wejściowego jako n, a liczbę punktów otoczki jako h.



1.2 Poprawność

Otoczkę uznajemy za poprawną jeżeli jej wierzchołki są zadane w kierunku przeciwnym lub zgodnym do ruchu wskazówek zegara. W dodatku, do otoczki **nie zaliczamy wewnętrznych punktów współliniowych**.



2. Algorytmy

2.1 Przyrostowy

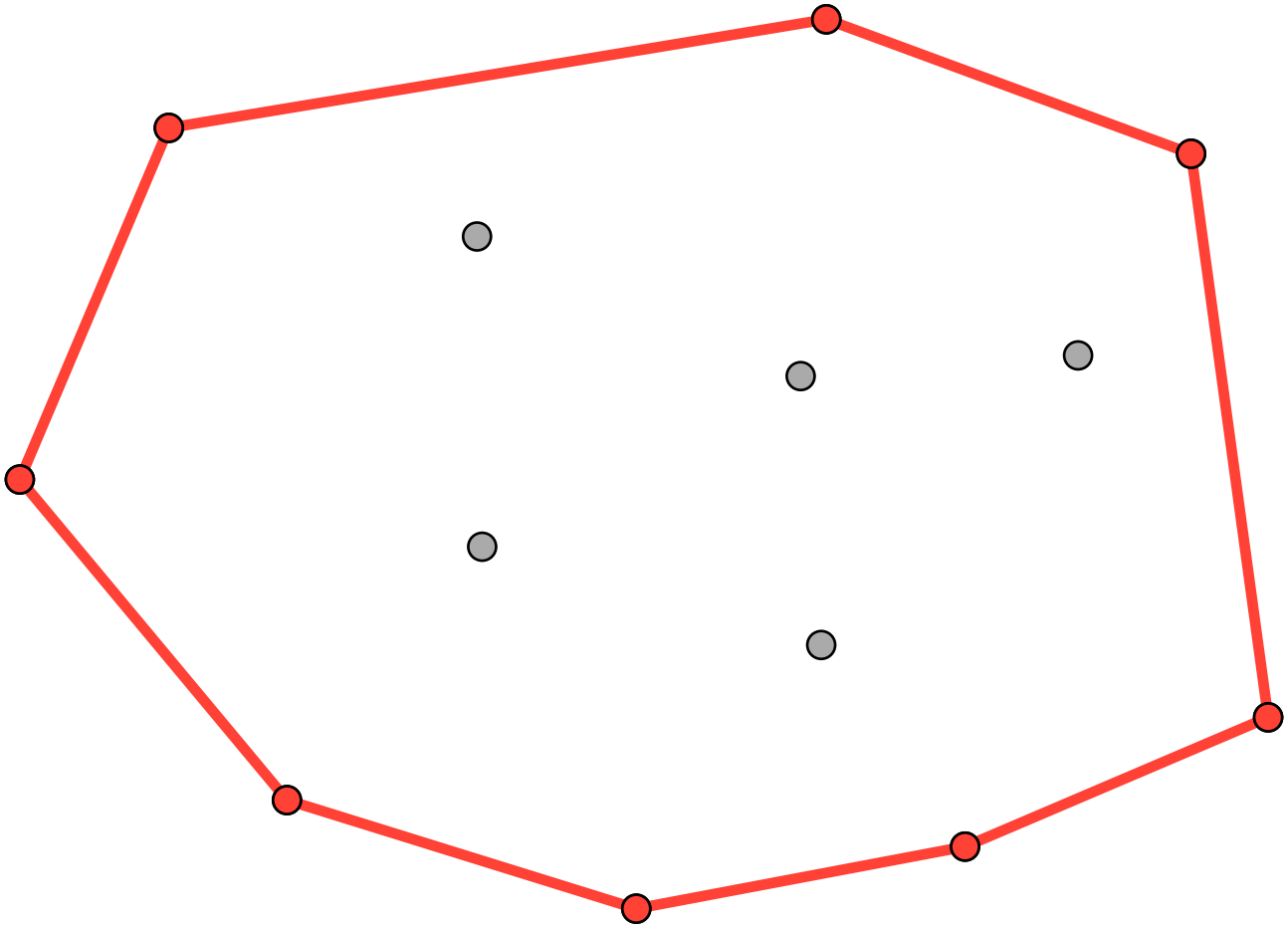
2.1.1 Działanie algorytmu

Algorytm najpierw sortuje punkty względem współrzędnej x, a następnie wybiera dwa pierwsze punkty, będące początkową otoczką. Następnie iteracyjnie dodaje kolejne punkty do otoczki, dokonane jest to poprzez znalezienie **stycznych do otoczki przechodzących przez kolejne punkty** i odpowiednie usuwanie punktów wewnątrz niej.

Złożoność czasowa algorytmu to $O(n \log n)$

2.1 Przyrostowy

2.1.2 Przykład



2.2 Grahama

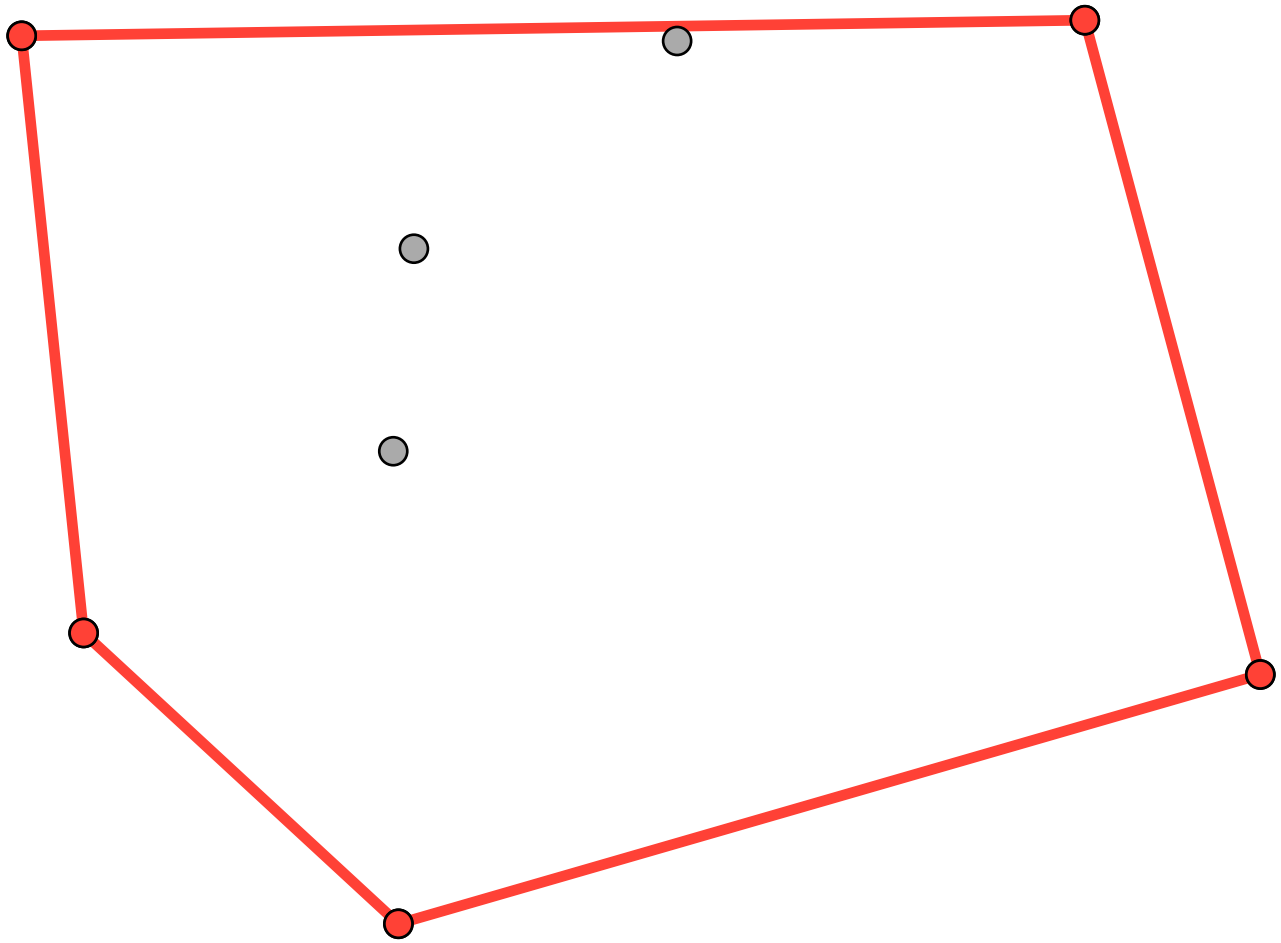
2.2.1 Działanie algorytmu

Algorytm wyszukuje najniższy punkt względem y, sortuje punkty na podstawie kąta jaki tworzy odcinek przez najniższy punkt oraz kolejny punkt z osią OX. Usunięte są również punkty współliniowe. Tworzy stos, dla każdego punktu usuwa punkty ze stosu aż dwa ostatnie z wybranym punktem przestaną tworzyć skręt w prawo i dodaje ten punkt na stos.

Złożoność czasowa algorytmu to $O(n \log n)$

2.2 Grahama

2.2.2 Przykład



2.3 Jarvisa

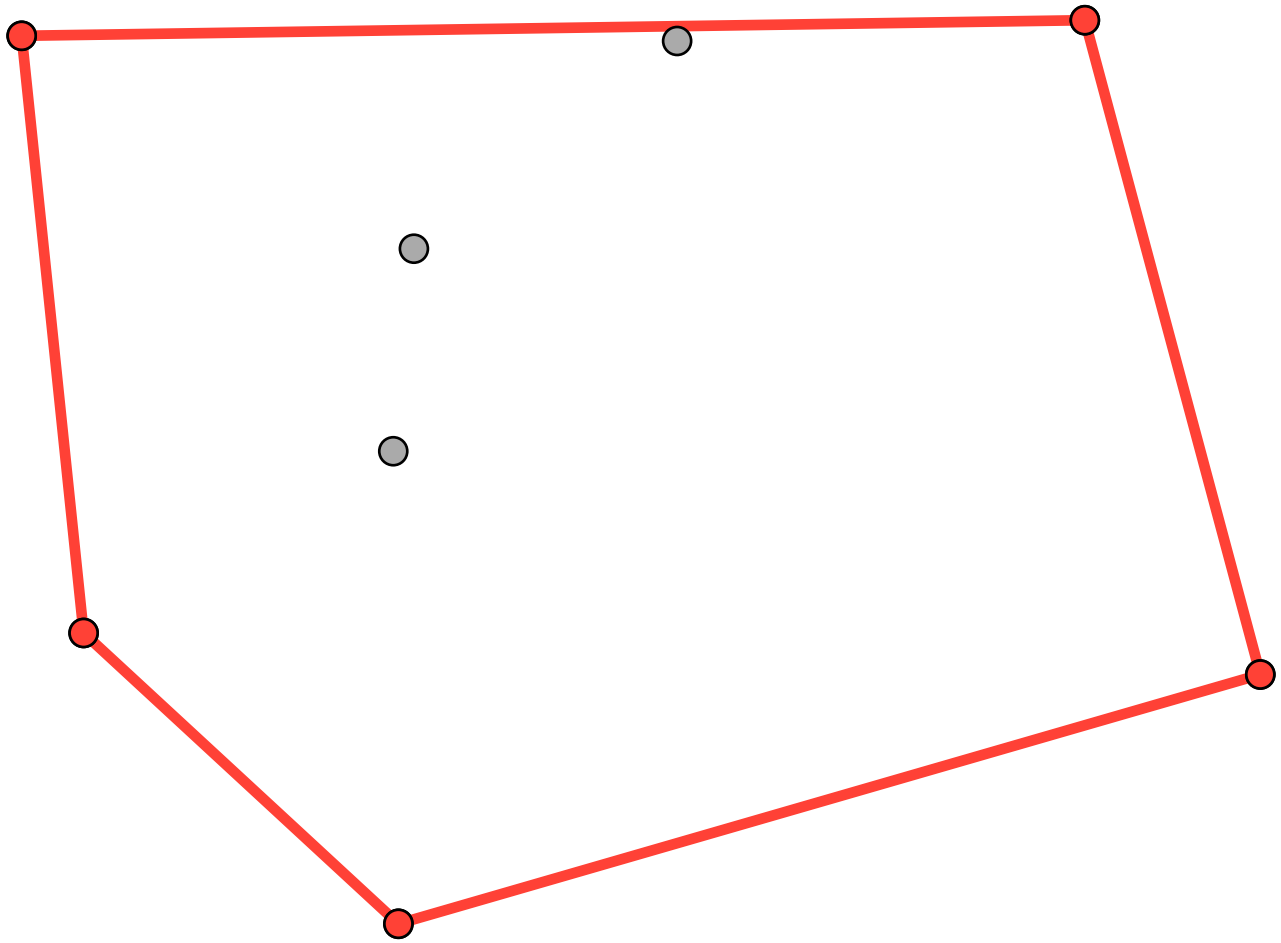
2.3.1 Działanie algorytmu

Algorytm rozpoczyna od znalezienia najniższego punktu, następnie iterując po wszystkich punktach, znajduje taki którego odcinek tworzony z ostatnim punktem otoczki spełnia warunek, że wszystkie punkty znajdują się po jego lewej stronie i dodaje go do otoczki. Algorytm kontynuuje do momentu spotkania początkowego punktu.

Złożoność czasowa algorytmu to $O(nh)$

2.3 Jarvisa

2.3.2 Przykład



2.4 Górna i dolna otoczka

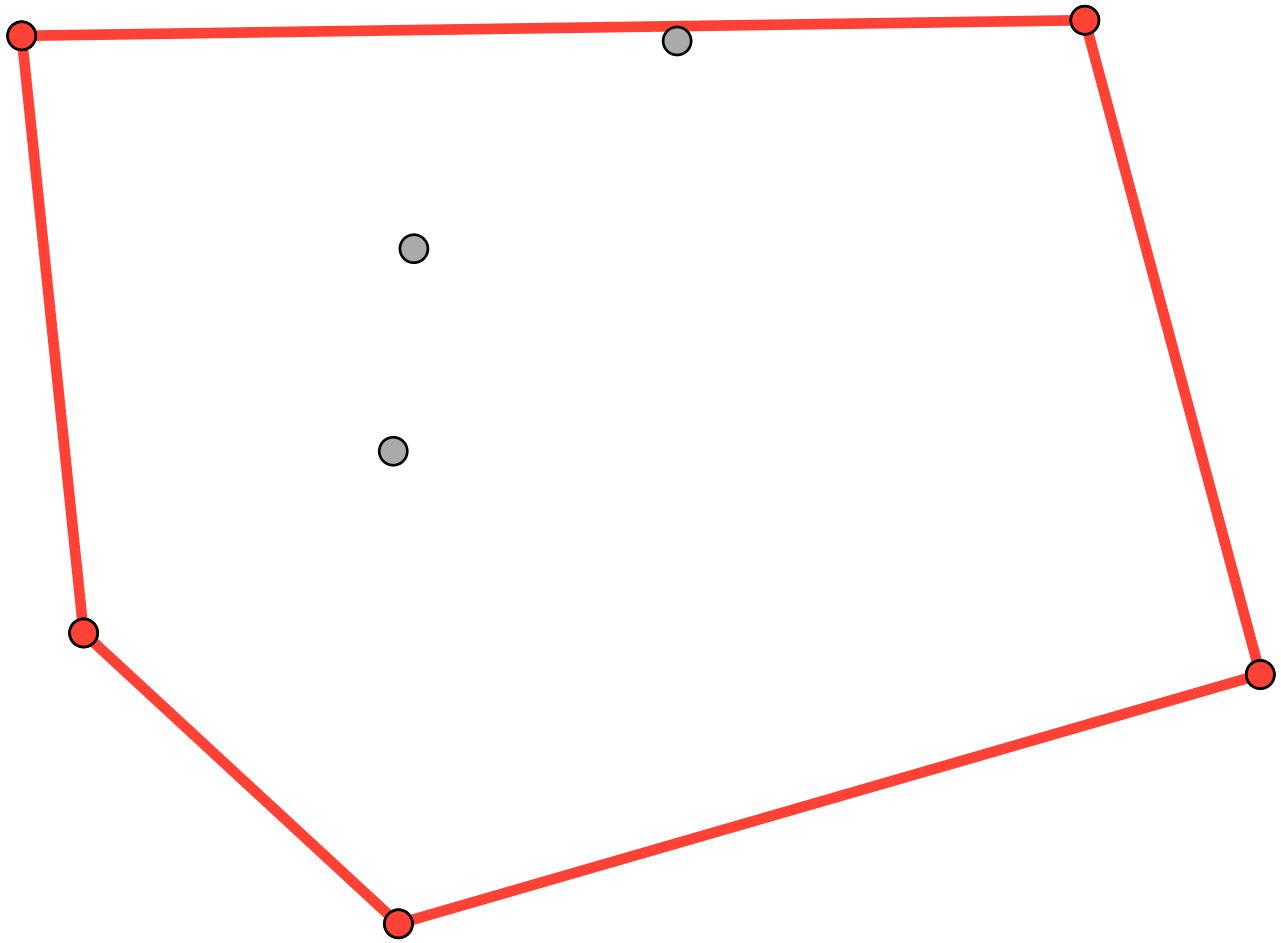
2.4.1 Działanie algorytmu

Algorytm sortuje punkty względem współrzędnej x. Pierwsze dwa punkty są początkiem górnej otoczki, iteracyjnie dodajemy kolejne punkty do niej, zachowując warunek wypukłości, podobnie jak w algorytmie Grahama. Analogicznie tworzymy dolną otoczkę, ostatecznie łącząc je w jedną, wynikową otoczkę.

Złożoność czasowa algorytmu to $O(n \log n)$

2.4 Górna i dolna otoczka

2.4.2 Przykład



2.5 Quickhull

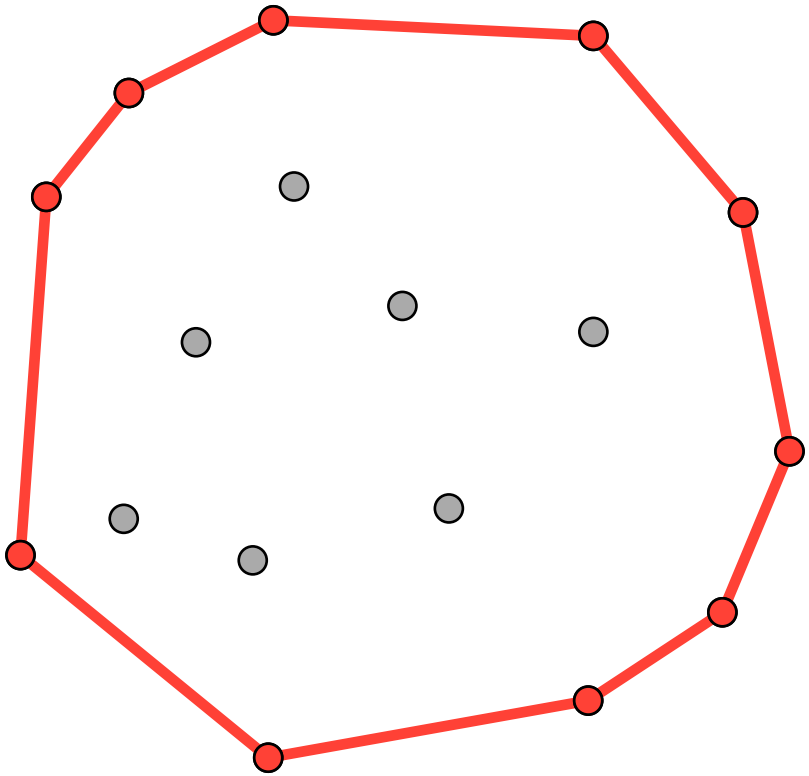
2.5.1 Działanie algorytmu

Algorytm wyznacza 4 skrajne punkty zbioru, tworząc wielokąt. Usuwane są wszystkie punkty wewnątrz tego wielokąta, a następnie na każdym z boków wywoływana jest rekurencyjna funkcja, która tworzy trójkąt tworzony przez dany bok jako podstawę oraz najbardziej oddalony od niej punkt znajdujący się po konkretnej stronie. Zwracamy ten najdalszy punkt oraz najdalsze punkty zwrócone poprzez rekurencyjne wywołanie na dwóch pozostałych bokach trójkąta.

Złożoność czasowa algorytmu to $O(n \log n)$

2.5 Quickhull

2.5.2 Przykład



2.6 Dziel i rządź

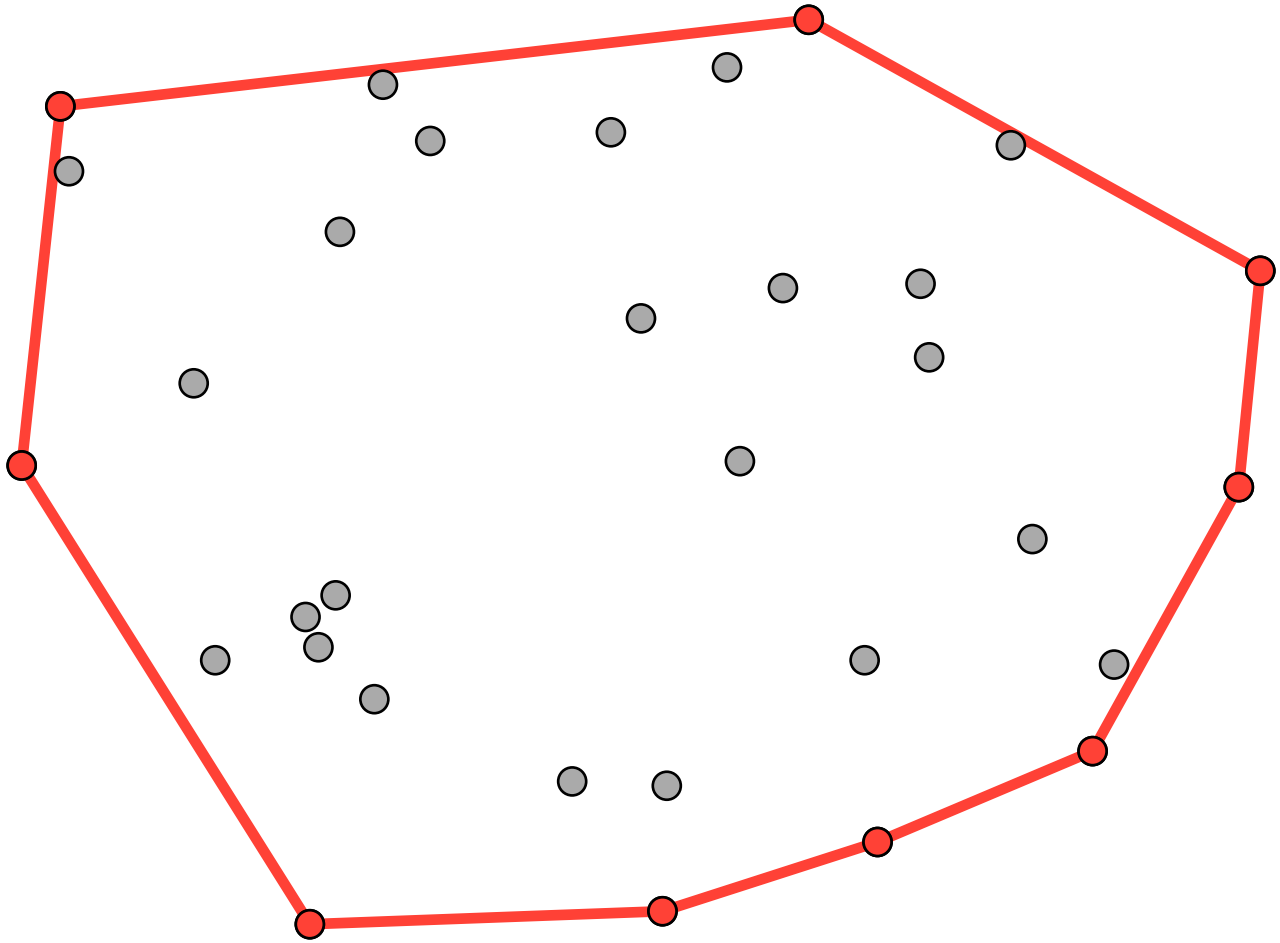
2.6.1 Działanie algorytmu

Algorytm sortuje punkty względem współrzędnej x, następnie dzieli zbiór na grupy względem mediany, aż do momentu gdy liczebność każdej z nich będzie mniejsza lub równa parametrowi algorytmu k. Algorytm Grahama wyznacza otoczkę dla każdej z grup, dzięki czemu możemy połączyć wszystkie mniejsze otoczki w jedną, wynikową. Łączenie sąsiednich otoczek polega na znajdowaniu stycznych.

Złożoność czasowa algorytmu to $O(n \log n)$

2.6 Dziel i rządź

2.6.2 Przykład



2.7 Chana

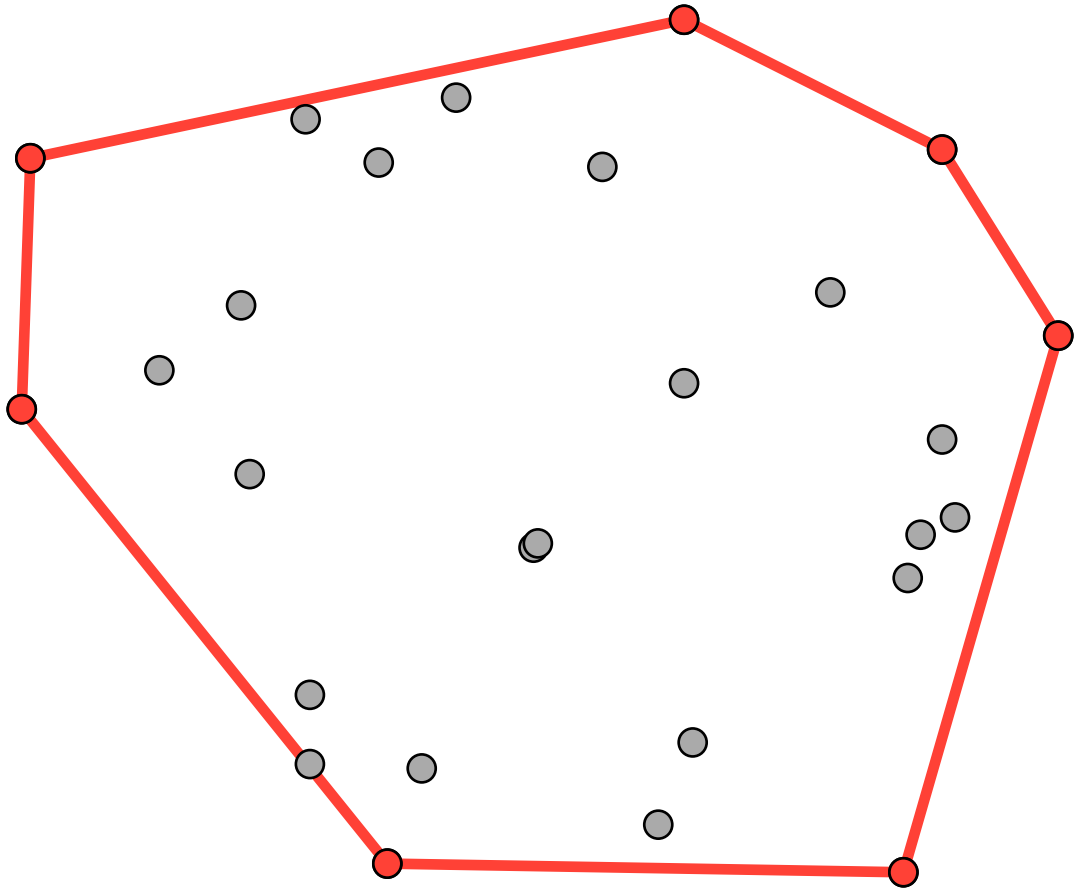
2.7.1 Działanie algorytmu

Algorytm łączy algorytmy Grahama i Jarvisa, zakładamy, że rozmiar otoczki będzie równy m, dzielimy zbiór na grupy o rozmiarze m i wyznaczamy ich otoczki algorytmem Grahama. Następnie znajdujemy skrajny punkt będący początkiem otoczki i szukamy wsród tych otoczek taki punkt, który maksymalizuje kąt. Robimy tak maksymalnie m razy. Ponieważ próba może się nie powieść, wywołujemy algorytm ustalając $m := \min\left(2^{2^t}, n\right)$, gdzie początkowo $t = 0$, do otrzymania prawidłowej otoczki.

Złożoność czasowa algorytmu to $O(n \log h)$

2.7 Chana

2.7.2 Przykład



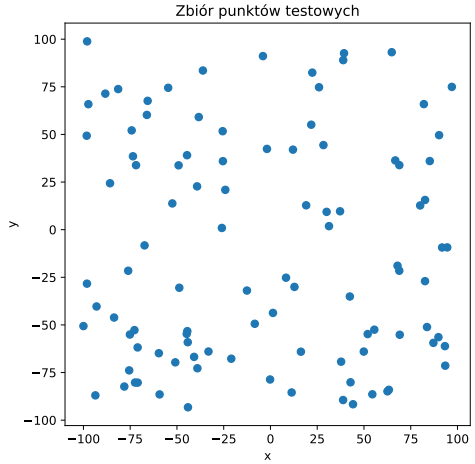
3. Zbiory testowe

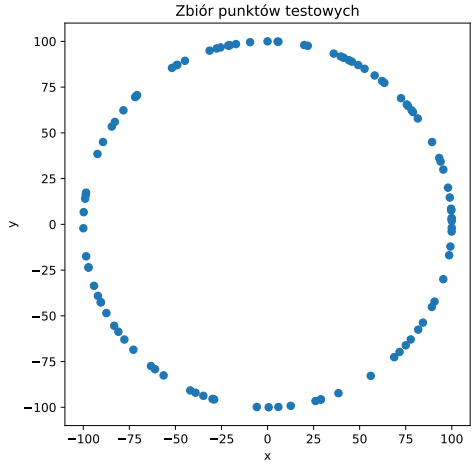
3.1 Wybrane zbiory testowe

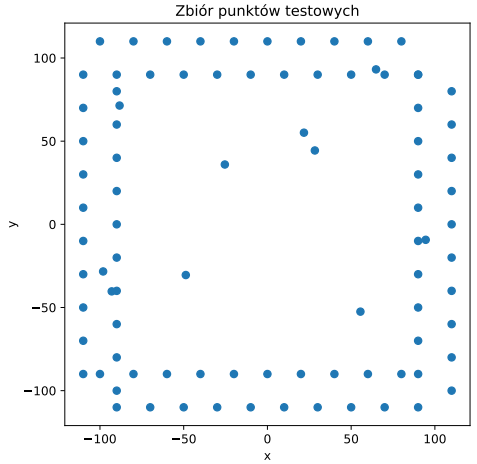
Aby sprawdzić poprawność oraz wydajność zaimplementowanych algorytmów, użyliśmy następujących zbiorów testowych:

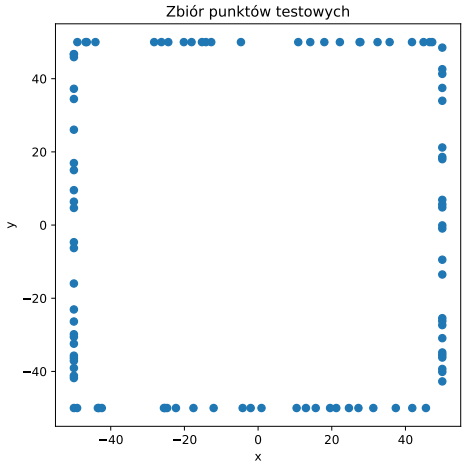
- **Zbiór 1** - punkty losowe z zakresu $[-100, 100]$,
- **Zbiór 2** - punkty na okręgu o promieniu 100,
- **Zbiór 3** - punkty wewnątrz kwadratu otoczonego obramówką kształtu zygzaka
- **Zbiór 4** - punkty na obramówce kwadratu
- **Zbiór 5** - punkty na obramówce kwadratu oraz jego przekątnych

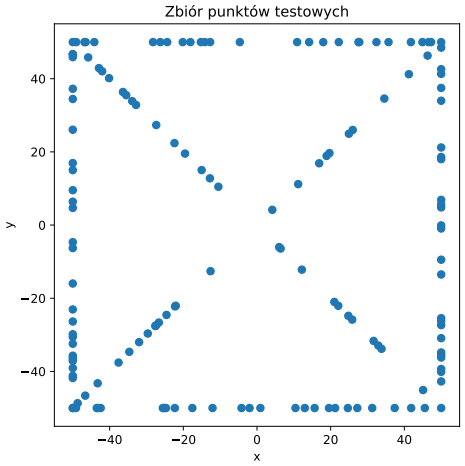
3.1 Wybrane zbiory testowe





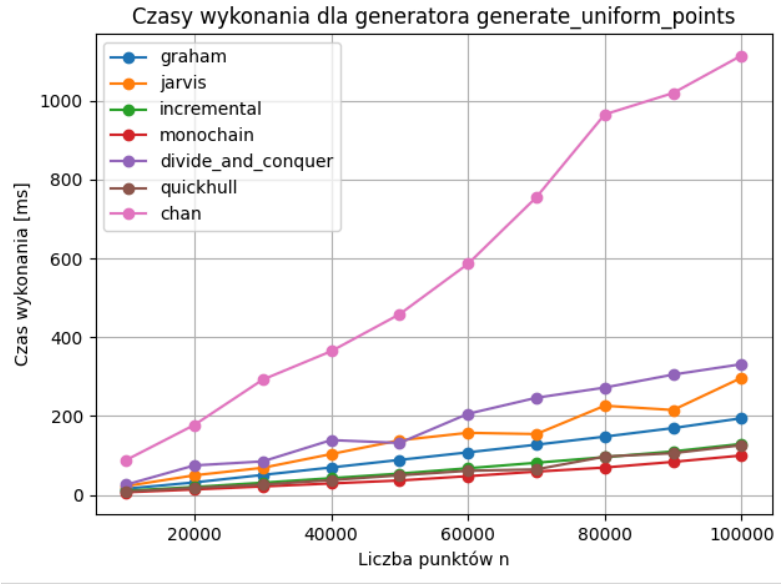






3.2 Wyniki testów wydajności i poprawności

3.2.1 Zbiór 1

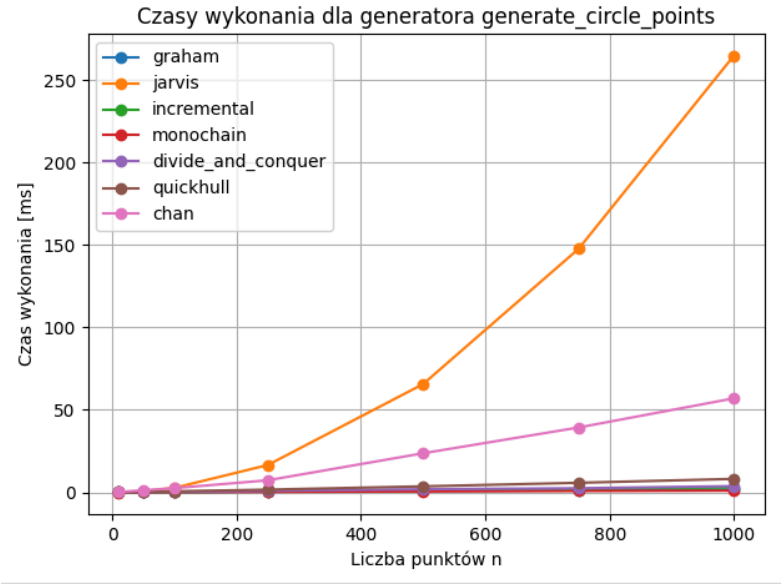


Liczba punktów	Graham [ms]	Jarvis [ms]	Przyrostowy [ms]	Górnej i dolnej otoczki [ms]	Dziel i rządź [ms]	Quickhull [ms]	Chan [ms]
20000	33.34	53.28	22.6	15.65	74.28	18.53	176.74
40000	67.39	95.16	44.4	29.1	113.4	35.45	379.74
60000	106.15	160.03	71.36	46.06	205.38	58.97	548.89
80000	146.23	231.49	105.18	70.33	269.23	86.72	811.12
100000	193.13	266.97	136.79	94.12	331.78	121.46	1045.29

Dla zbioru w pełni losowego, bez przewidywalnej liczby punktów otoczki najszybsze okazjują się algorytmy górnej i dolnej otoczki, przyrostowy oraz Quickhull. Pozostałe algorytmy jednak znacząco od nich nie odbiegają z wykluczeniem algorytmu Chana. Może wydawać się, że algorytm Chana nie osiąga oczekiwanej złożoności, jednak rozbieżność może wynikać z charakterystyki algorytmu i dużej stałej nie branej pod uwagę podczas opisywania prędkości poprzez notację dużego O.

3.2 Wyniki testów wydajności i poprawności

3.2.2 Zbiór 2

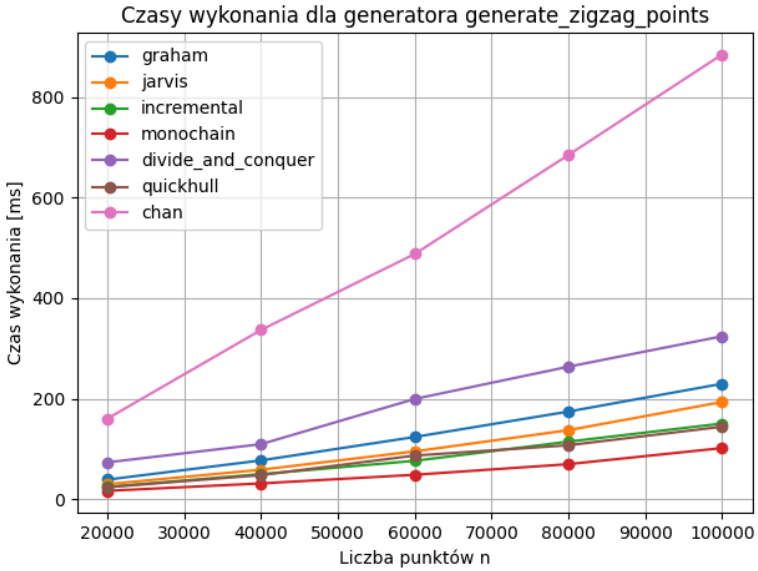


Liczba punktów	Graham [ms]	Jarvis [ms]	Przyrostowy [ms]	Górna i dolna [ms]	Dziel i rządź [ms]	Quickhull [ms]	Chan [ms]
10	0.03	0.04	0.03	0.02	0.07	0.04	0.21
50	0.09	0.68	0.08	0.05	0.17	0.23	1.22
100	0.17	2.7	0.17	0.11	0.32	0.57	2.52
250	0.44	16.54	0.44	0.26	0.89	1.62	7.28
500	0.86	65.65	1.07	0.53	1.88	3.62	23.68
750	1.3	147.64	1.95	0.8	2.49	5.77	39.3
1000	1.76	264.87	3.03	1.03	3.74	8.14	57

Dla zbioru, w którym każdy punkt należy do otoczki można zaobserwować wyniki zgodne z oczekiwaniami. Czasy procesora algorymów Jarvisa oraz Chana, których złożoność obliczeniowa degraduje przy takich danych do kolejno $O(n^2)$ oraz $O(n \log n)$ odbiegają znacząco od czasów pozostałych algorytmów.

3.2 Wyniki testów wydajności i poprawności

3.2.3 Zbiór 3

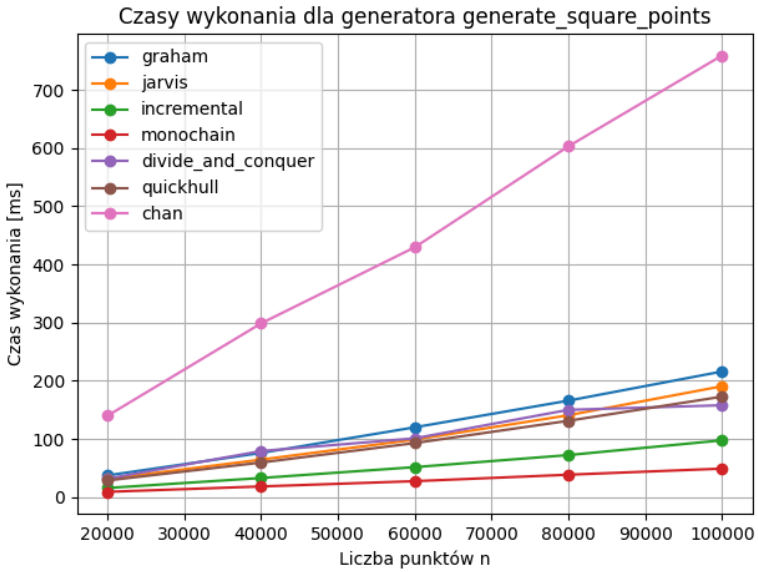


Liczba punktów	Graham [ms]	Jarvis [ms]	Przyrostowy [ms]	Górna i dolna [ms]	Dziel i rządź [ms]	Quickhull [ms]	Chan [ms]
20000	38.56	29.53	24.05	16.19	72.96	23.48	160.24
40000	76.84	58.33	49.54	31	109.25	47.56	336.87
60000	123.54	94.85	76.01	48.19	199.31	86.49	488.16
80000	174.17	137.08	114.23	69.36	263.54	107.07	685.34
100000	229.51	193.2	150.31	101.41	324.14	144	885.59

Zbiór 3 jest bardzo podobny do zbioru 1, jedyne co go odróżnia to obramówka. Dzięki niej liczba punktów otoczki jest stała i wynosi 8. Zmiana ta daje się zauważyć poprzez mniejszy czas wykonywania algorytmów Jarvisa oraz Chana. Czasy pozostałych programów są takie same, lub nieznacznie większe ze względu na dodatkowe punkty na obramówce.

3.2 Wyniki testów wydajności i poprawności

3.2.4 Zbiór 4

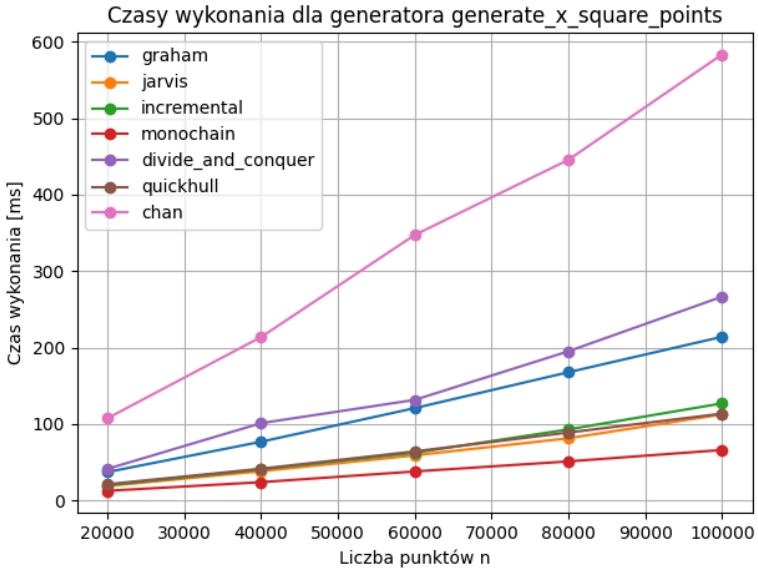


Liczba punktów	Graham [ms]	Jarvis [ms]	Przyrostowy [ms]	Górna i dolna [ms]	Dziel i rządź [ms]	Quickhull [ms]	Chan [ms]
20000	37.36	31.3	15.72	9.1	31.75	28.67	139.93
40000	75.99	64.53	32.89	18.38	79.29	59.51	298.72
60000	120.05	98.9	51.59	27.46	101.1	92.93	429.49
80000	165.86	140.75	72.3	38.54	150.06	131.26	602.92
100000	215.95	190.67	97.63	49.03	157.94	172.72	759.16

W zbiorze 4 liczba punktów otoczki ponownie wynosi 8, co ponownie pozwala na zaobserowowanie krótszego czasu wykonywania algorytmu Jarvisa oraz Chana. Ponadto każda podotoczka tworzona przez algorytm dziel i rządź ma dokładnie 4 punkty, co pozwala na szybkie ich łączenie i powoduje, że algorytm działa w przybliżeniu 2 razy szybciej niż w przypadku zbiorów 1 oraz 3. Podobnie algorytm górnej i dolnej otoczki dzięki dużej liczbie punktów współliniowych i małym rozmiarze stosu otoczki działa w przybliżeniu dwukrotnie szybciej.

3.2 Wyniki testów wydajności i poprawności

3.2.5 Zbiór 5



Liczba punktów	Graham [ms]	Jarvis [ms]	Przyrostowy [ms]	Górna i dolna [ms]	Dziel i rządź [ms]	Quickhull [ms]	Chan [ms]
20000	38.24	19.06	19.9	12.33	41.83	20.23	105.38
40000	77.58	38.48	40.69	23.96	101.14	42.01	214.73
60000	120.98	60.35	62.85	36.22	130.84	63.74	343.87
80000	168.15	87.75	88.43	50.92	195.57	88.32	443.88
100000	269.45	114.11	117.44	67.63	267.27	117.77	579.92

Tym razem liczba punktów otoczki zawsze wynosi 4, ze względu na obecność wierzchołków kwadratu w zbiorze 5. Algorytm Jarvisa wedle oczekiwań działa w przybliżeniu 2 razy szybciej. Czas działaniu algorytmu Chana również spadł znacząco, co w połączeniu z brakiem różnicy w liczbie punktów względem zbioru 4 potwierdza jego teoretyczną złożoność $O(n \log k)$. Ponadto obecność przekątnych spowodowała, że algorytmy dziel i rządź oraz górnej i dolnej otoczki nie były już takie szybkie jak w zbiorze 4 i ich czasy pracy zbliżyły się do tych na zbiorze 1.

- slajdy z wykładu,
- materiały do laboratoriów,
- https://en.wikipedia.org/wiki/Convex_hull_algorithms#Akl%E2%80%93Toussaint_heuristic (7.01.2026),
- https://en.wikipedia.org/wiki/Gift_wrapping_algorithm (7.01.2026),
- https://en.wikipedia.org/wiki/Graham_scan (7.01.2026),
- https://en.wikipedia.org/wiki/Quickhull (7.01.2026),
- https://gist.github.com/tixxit/252229 (7.01.2026),
- https://www.youtube.com/watch?v=O_K5_whYhoU (7.01.2026),
- https://www.youtube.com/watch?v=NH6WbP3lDac (7.01.2026).

5. Dodatek

5.1 Analiza złożoności algorytmu Chana

```python
def chan_hull(points, m):
    n = len(points)
    hulls = list(map(graham, map(list, batched(points, n=m))))
    p1 = max(points, key=lambda p: (p[0], p[1])) #najbardziej na prawo
    last = p1
    L = [last]
    for j in range(m):
        best = points[0] if points[0] != last else points[1]
        for i in range(len(hulls)):
            q = rtangent(hulls[i], last)
            d = det(last,best,q)
            if d < -eps or (d < eps and dist(last,best) < dist(last,q)):
                best = q
        L.append(best)
        last = best
        if best == p1:
            L.pop()  # usuwamy powtorzony punkt startowy
            return L
    return None
```

Python

$O(m \log m) \times \frac{n}{m} = O(n \log m)$

$O(\log m)$

$O(\log m) \times \frac{n}{m} \times m = O(n \log m)$

5.1 Analiza złożoności algorytmu Chana

```python
def chan(points):
    t = 0
    n = len(points)
    while True:
        m = min(n, 2**(2**t))
        result = step_chan(points, m)
        if result != None:
            return result
        t += 1
```

$O(n \log m) = O\left(n \log 2^{2^t}\right)$

$\times$ $O(\log \log h)$

$$\sum_{t=0}^{\log \log h} O\left(n \log 2^{2^t}\right) = O(n) \sum_{t=0}^{\log \log h} O\left(2^t\right) \leq O(n) \cdot O\left(2^{\log \log h + 1}\right)$$

$$= O(n) \cdot O(\log h) = O(n \log h)$$