

**Analiza algorytmów – dokumentacja końcowa**  
**Wojciech Janaszek, 273689 ISI**  
**semestr 17L**

## **I. Opis problemu**

Rozpatrywany problem to rozplątywanie odcinków – mamy 2 zbiory punktów na płaszczyźnie, po N punktów każdy. Należy znaleźć taki zbiór odcinków, który połączy punkty z obu zbiorów tak, że żaden z odcinków się nie przecina, oraz odcinki łączą tylko punkty z innych grup.

## **II. Opis metod rozwiązania**

W ramach projektu zrealizowano 2 algorytmy rozwiązujące zadany problem:

1. „Brutal force” - polega na sprawdzaniu wszystkich możliwych połączeń odcinków, i odrzucaniu takich zbiorów odcinków, które nie spełniają określonych wymogów (łączą punkty z tego samego zbioru punktów, lub przecinają inny odcinek)
2. „Pomysłowy” - polega na znajdowaniu najbliższych punktów z obu grup, i łączeniu ich w odcinki. Następnie sprawdzamy tak połączone punkty, czy jakiś odcinek nie przecina innego, i jeśli tak jest, to zmieniamy połączenia 4 punktów tworzących tę parę odcinków, aby połączyć te punkty w drugi sposób. Następnie znowu sprawdzamy, czy w naszym zbiorze odcinków nie ma przecinających się odcinków. Te czynności powtarzamy, aż do uzyskania zbioru odcinków, w którym żadne 2 odcinki się nie przecinają.

## **III. Ograniczenia**

Ograniczenia wynikają z ograniczeń pamięciowych dla problemu o dużym rozmiarze.

- Algorytm 1 – maksymalna liczba wierzchołków 6 (powyżej tej wartości następuje przekroczenie dopuszczalnego rozmiaru struktury *vector* – `std::bad_alloc`)
- Algorytm 2 – maksymalna liczba wierzchołków to 1900 (powyżej tej wartości program przerwa po dłuższym czasie pracę)

## **IV. Opis wykorzystywanych struktur danych i pomocniczych algorytmów**

W projekcie niezależnie od wybranego algorytmu używane są następujące struktury danych:

- *Point2D* – reprezentująca punkt na płaszczyźnie 2D
- *Segment* – reprezentująca odcinek między dwoma punktami
- *SetOfSegments* – reprezentująca zbiór odcinków

Dodatkowo w algorytmie nr 1 wykorzystywane są następujące struktury danych:

- *Sweeper* – tzw. miotła – służąca do wykrywania, czy w zbiorze odcinków znajdują się jakiegokolwiek przecinające się odcinki

## **V. Instrukcja obsługi programu**

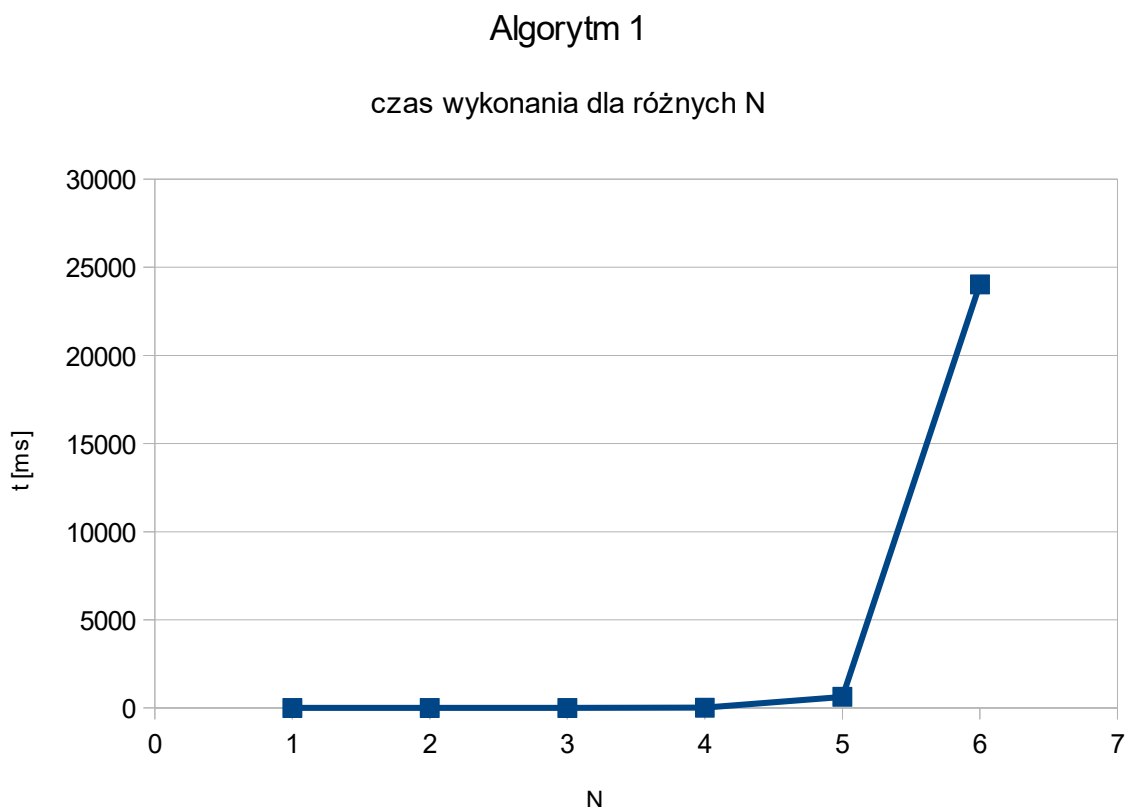
Program umożliwia 3 różne wykonania:

- pierwsze – wczytując dane testowe z klawiatury bądź z pliku; służy do weryfikacji poprawności
  - drugie – generując dane testowe z zadaniem ziarnem („seed”) lub z ziarnem domyślnym; służy do weryfikacji poprawności
  - trzecie – generacja danych testowych, pomiar czasu, i przedstawienie wyników pomiarów dla zadanej ilości pomiarów, rozmiaru problemu w tabelce
- Szczegółowa instrukcja kompilacji i uruchomienia programu znajduje się w pliku `readme.txt`

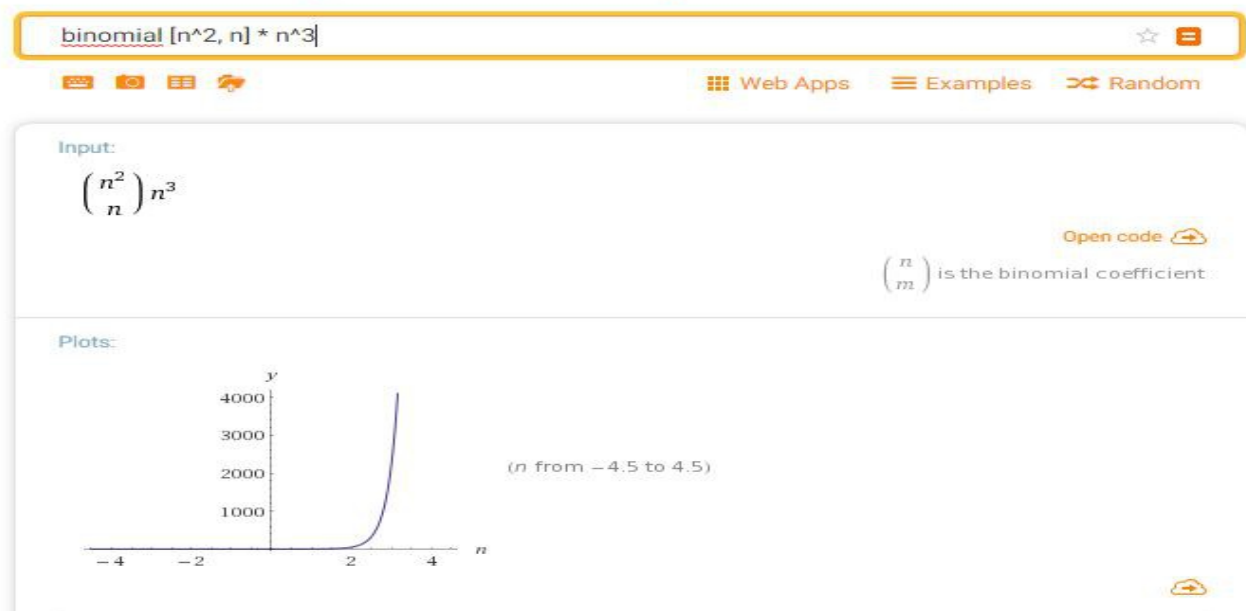
## **VI. Złożoność obliczeniowa**

### Algorytm 1 (brutal force)

W związku z ogromną ilością kombinacji, jakie tworzą się dla możliwych zbiorów odcinków, okazało się niemożliwe ostateczne potwierdzenie zakładanej złożoności, gdyż dla  $N > 6$  algorytm wyrzucał wyjątek związany z przekroczeniem maksymalnego rozmiaru vectora z możliwymi zbiorami odcinków. Poniżej wykres wykonania czasu algorytmu dla różnych rozmiarów problemu  $N$ :

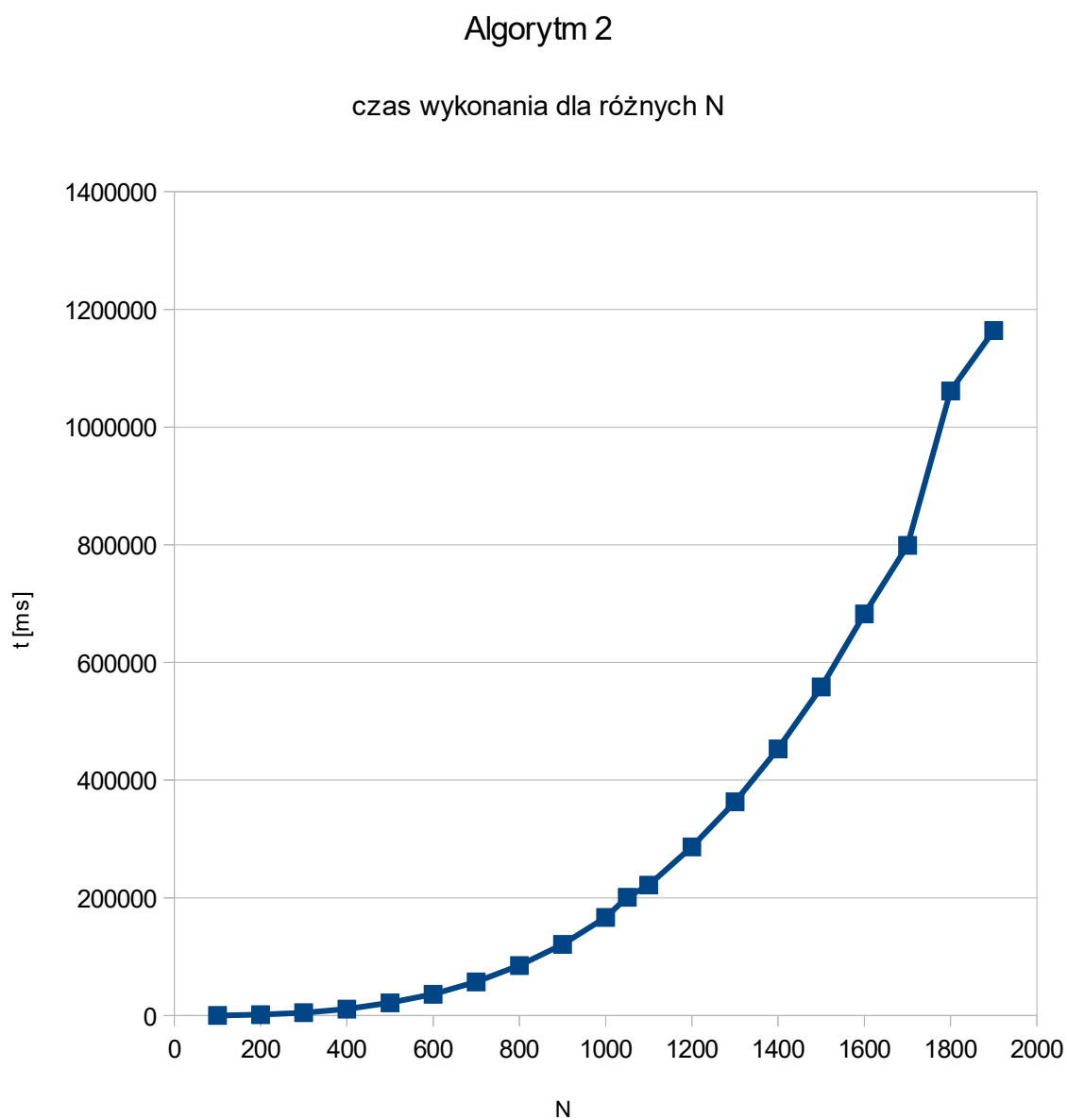


Oszacowanie złożoności z koncepcji wstępnej wydaje się być poprawne – bardzo szybko rośnie czas wykonania algorytmu już dla pojedynczych wartości. Podsumowując, algorytm ma złożoność około  $O\left(\binom{N^2}{N} \cdot N^3\right)$ . Jest to złożoność pesymistyczna. Dla porównania wykres z *Wolfram Alpha*:



## Algorytm 2

Poniżej został przedstawiony wykres czasu wykonania algorytmu dla różnych rozmiarów problemu  $N$ :



Przeprowadzono też pomiary z obserwacją parametru  $q(n)$  – dla mniejszych wartości oszacowanie złożoności wydaje się być poprawne ( $q(n)$  oscyluje wokół wartości 1.0):

```
mediana = 450, N = 100
Rozpoczynam pomiary
optimal
```

n	t(n) [ms]	q(n)
100	183	1.0844
200	1385	1.02588
300	4596	1.00868
400	10815	1.00135
450	15378	1
500	21219	1.00589
600	36172	0.992331
700	64016	1.10594
800	91214	1.05567

Jednak dla większych rozmiarów problemu pojawiają się błędy – zatem jest to przybliżone, ale nie dokładne oszacowanie złożoności algorytmu, które wynosi  $O(N^3)$  :

```

mediana = 1050, N = 100
Rozpoczynam pomiary
optimal

```

n	t(n) [ms]	q(n)
100	189	1.08774
200	1520	1.09349
300	4879	1.03999
400	11102	0.998353
500	21782	1.00288
600	36194	0.964375
700	57276	0.96104
800	84979	0.955223
900	120823	0.953861
1000	166809	0.960025
1050	201143	1
1100	221688	0.958577
1200	286588	0.954503
1300	363005	-0.995811
1400	453168	-1.68159
1500	558333	-3.49288
1600	682294	-19.7357
1700	798791	7.43849
1800	1.06152e+006	3.97471
1900	1.16426e+006	-3.87107

## VII. Końcowe uwagi

W projekcie jest błąd, który dla wartości SEED = 10, i  $N > 450$  powoduje nagłe przerwanie działania programu. Prawdopodobnie chodzi o zarządzanie pamięcią dynamicznie przydzielaną. Zostało to przeze mnie przeanalizowane, ale ze względu na brak czasu nie zostało to naprawione. Dla innych wartości parametru SEED i N program działa bez zarzutu.

W wersji na Linuksa po zbudowaniu i uruchomieniu programu pojawił się błąd – glibc detected (naruszenie ochrony pamięci). Co dziwniejsze, pod Windowsem nic takiego oprócz wymienionego wcześniej błędu dla jednej wartości się nie działo. Pomogło zakomentowanie operacji „delete ...”. W związku z brakiem dostępnego czasu to również nie zostało naprawione i dokładniej zbadane. Aczkolwiek jest to bardzo zagadkowe.