Artur Woliński, 258952 04.04.2022r

# Projektowanie algorytmów i metody sztucznej inteligencji
# Projekt 1.

## 1. Wprowadzenie

Zadaniem projektowym było zaimplementowanie trzech różnych algorytmów sortowania oraz wykonania szeregu testów.

Wybrane algorytmy sortowania:

- Sortowanie przez scalanie
- Sortowanie szybkie
- Sortowanie introspektywne

Algorytmy testowane były dla 100 tablic o następującej ilości danych :

- 10000
- 50000
- 100000
- 500000
- 1000000

Oraz o następujących stopniach wstępnego posortowania tablic:

- 0%
- 25%
- 50%
- 75%
- 95%
- 99%
- 99,7%
- wszystkie elementy posortowane w odwrotnej kolejności

## 2. Opis wybranych algorytmów

### 2.1. Sortowanie przez scalanie

Jest to rekurencyjny algorytm sortowania, który stosuje metodę „Dziel i zwyciężaj”. Jego działanie można przedstawić w dwóch krokach:

- Dziel rekurencyjnie na połówki cały zestaw danych aż do osiągnięcia jednoelementowych podproblemów
- Scalaj z powrotem kolejne podproblemy wraz z ich sortowaniem aż do osiągnięcia posortowanego zestawu danych

Złożoność obliczeniowa tego algorytmu dla każdego przypadku wynosi tyle samo :

$$O(n * log(n))$$

## 2.2. Sortowanie szybkie

Jest to jeden z bardziej popularnych algorytmów który również stosuje metodę „dziel i zwyciężaj". W każdym kroku sortowania szybkiego zostaje wybrany element służący do podziału tablicy. Następnie algorytm porównuje wszystkie elementy tablicy z wybranym i tworzy 2 nowe tablice, jedną zawierającą elementy mniejsze, a drugą zawierającą większe. Element wybrany do podziału nie bierze dalej udziału w sortowaniu, ponieważ jest już na swojej pozycji. Kroki te są powtarzane aż do uzyskania posortowanej tablicy.

Złożoność obliczeniowa algorytmu quicksort wynosi:

- Dla przypadku optymistycznego $O(n * \log(n))$
- Dla przypadku typowego $O(n * \log(n))$
- Dla przypadku pesymistycznego $O(n^2)$

## 2.3. Sortowanie introspektywne

Jest to algorytm hybrydowy, w którym został wyeliminowany problem $O(n^2)$ występujący w najgorszym przypadku algorytmu sortowania szybkiego.

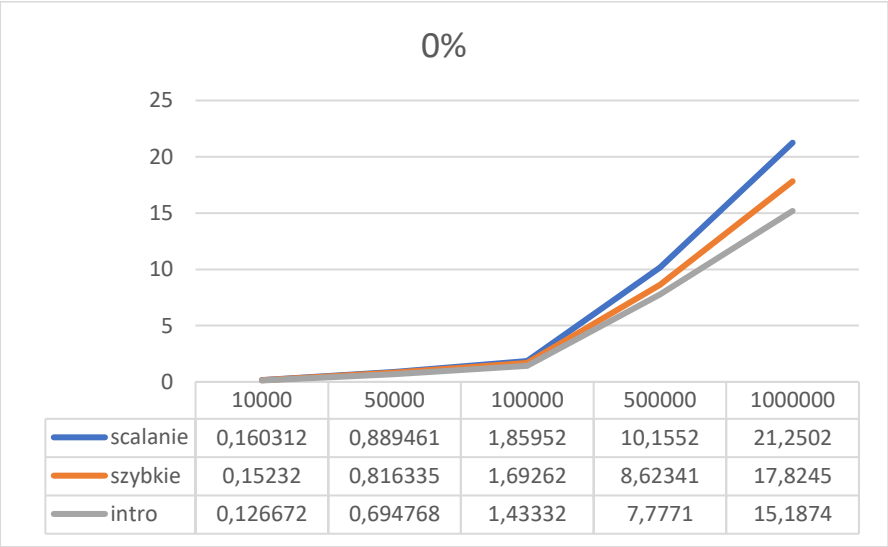
Rozwiązaniem problemu złożoności obliczeniowej O(n2) w najgorszym przypadku jest badanie głębokości rekurencji. W procedurze głównej Sortowania Introspektywnego: Hybrid Introspective Sort tworzona jest stała 'maxdepth' o wartości 2·log2n, która określa maksymalną dozwoloną głębokość wywołań rekurencyjnych. Następnie wywoływana jest procedura Intro Sort. Procedura Intro Sort przyjmuje jako dodatkowy parametr wartość 'maxdepth', która określa maksymalną dozwoloną głębokość wywołań rekurencyjnych z poziomu, na którym obecnie się znajdujemy. Jeżeli wartość parametru 'maxdepth' wynosi 0, wywołania rekurencyjne są kończone i dla podproblemu, którym obecnie się zajmujemy, wywoływana jest procedura Sortowania Przez Kopcowanie, które jest traktowane jako sortowanie pomocnicze.

W przypadku gdy maxdepth > 0 procedura Intro Sort działa podobnie jak procedura Quick Sort. Wywoływana jest procedura partition, która dzieli tablicę na dwa rozłączne podzbiory, gromadząc w pierwszym elementy posiadające klucze o wartości mniejszej równej wartości klucza elementu rozdzielającego, a w drugim elementy o wartościach kluczy większych równych kluczowi pivota. Następnie dla obu podzbiorów wywołana jest rekurencyjnie procedura Intro Sort z parametrem maxdepth pomniejszonym o 1, w związku z czym maksymalna dozwolona głębokość rekurencyjnych wywołań z następnego poziomu jest o 1 mniejsza.

Złożoność obliczeniowa w każdym przypadku wynosi $O(n * \log(n))$

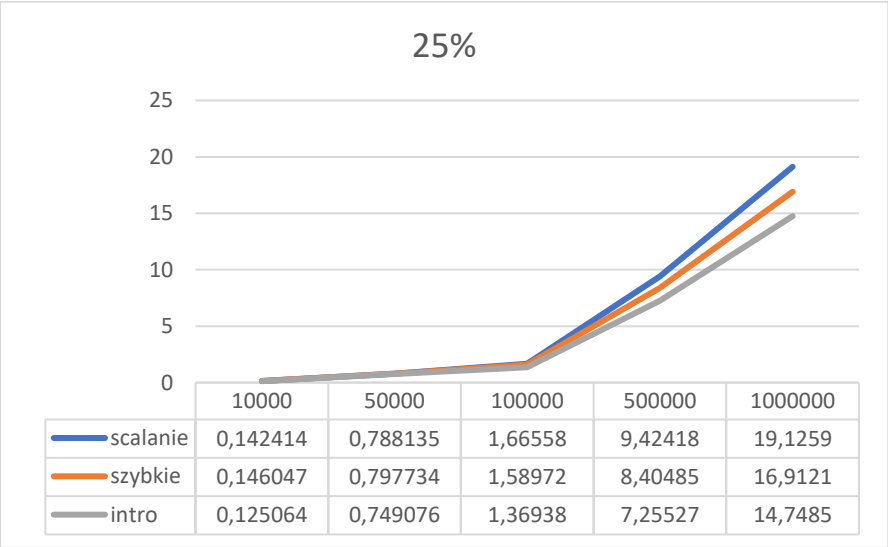
# 3. Przedstawienie wyników testów

## 3.1. Wszystkie elementy losowe.



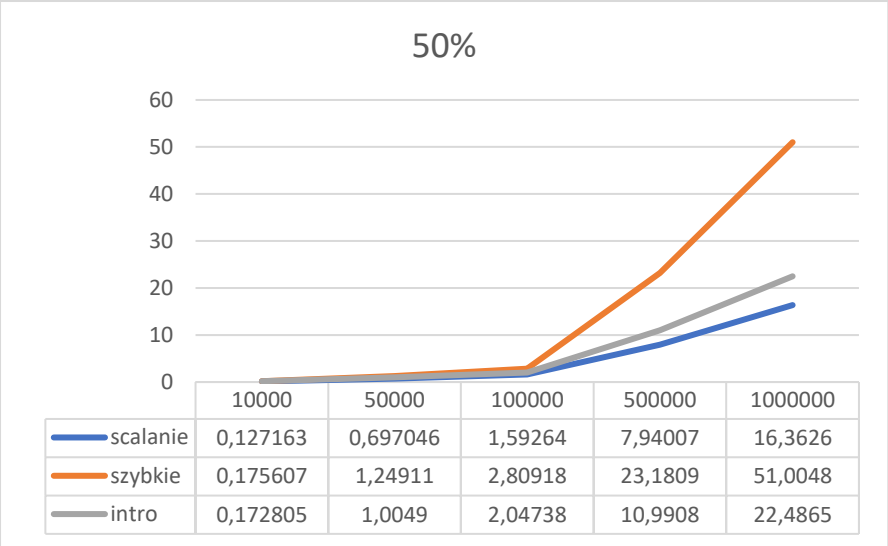

| | 10000 | 50000 | 100000 | 500000 | 1000000 |
|---|---|---|---|---|---|
| scalanie | 0,160312 | 0,889461 | 1,85952 | 10,1552 | 21,2502 |
| szybkie | 0,15232 | 0,816335 | 1,69262 | 8,62341 | 17,8245 |
| intro | 0,126672 | 0,694768 | 1,43332 | 7,7771 | 15,1874 |

## 3.2. Elementy posortowane w 25%.



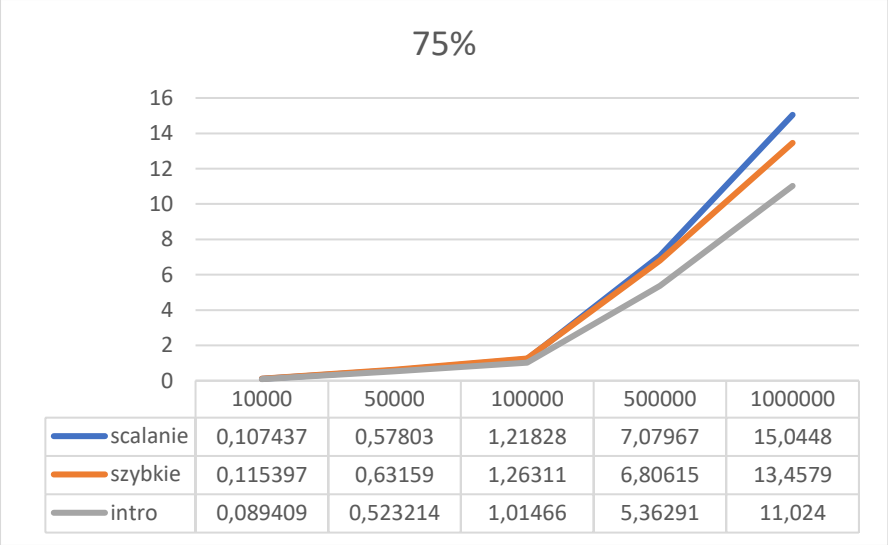

| | 10000 | 50000 | 100000 | 500000 | 1000000 |
|---|---|---|---|---|---|
| scalanie | 0,142414 | 0,788135 | 1,66558 | 9,42418 | 19,1259 |
| szybkie | 0,146047 | 0,797734 | 1,58972 | 8,40485 | 16,9121 |
| intro | 0,125064 | 0,749076 | 1,36938 | 7,25527 | 14,7485 |

## 3.3. Elementy posortowane w 50%.



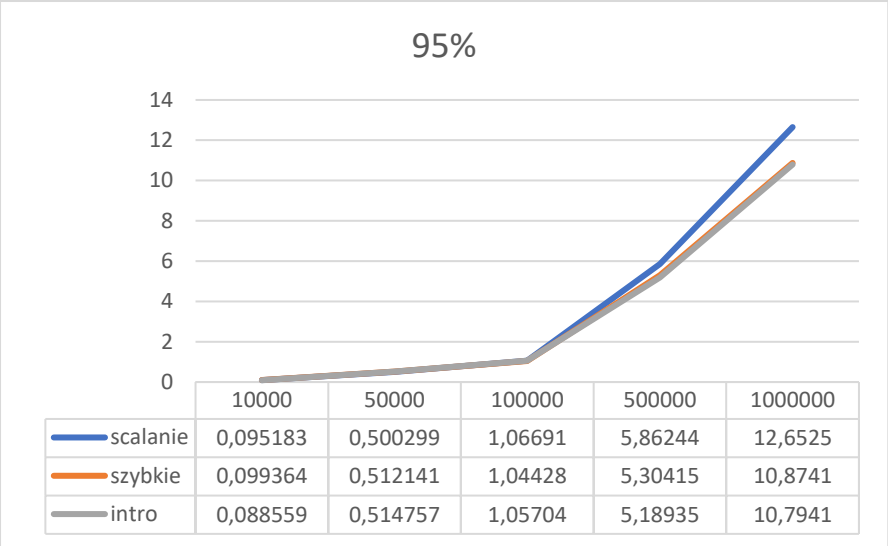

| | 10000 | 50000 | 100000 | 500000 | 1000000 |
|---|---|---|---|---|---|
| scalanie | 0,127163 | 0,697046 | 1,59264 | 7,94007 | 16,3626 |
| szybkie | 0,175607 | 1,24911 | 2,80918 | 23,1809 | 51,0048 |
| intro | 0,172805 | 1,0049 | 2,04738 | 10,9908 | 22,4865 |

3.4. Elementy posortowane w 75%



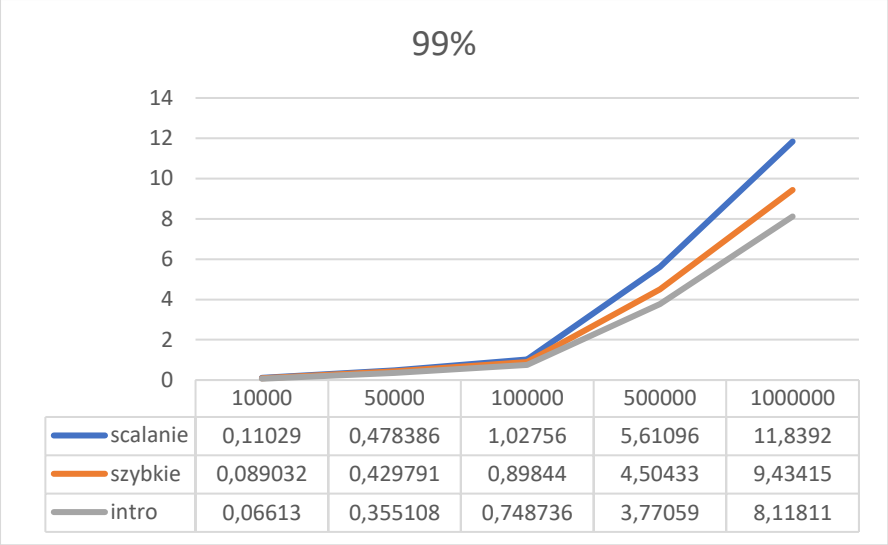

| | 10000 | 50000 | 100000 | 500000 | 1000000 |
|---|---|---|---|---|---|
| scalanie | 0,107437 | 0,57803 | 1,21828 | 7,07967 | 15,0448 |
| szybkie | 0,115397 | 0,63159 | 1,26311 | 6,80615 | 13,4579 |
| intro | 0,089409 | 0,523214 | 1,01466 | 5,36291 | 11,024 |

3.5. Elementy posortowane w 95%



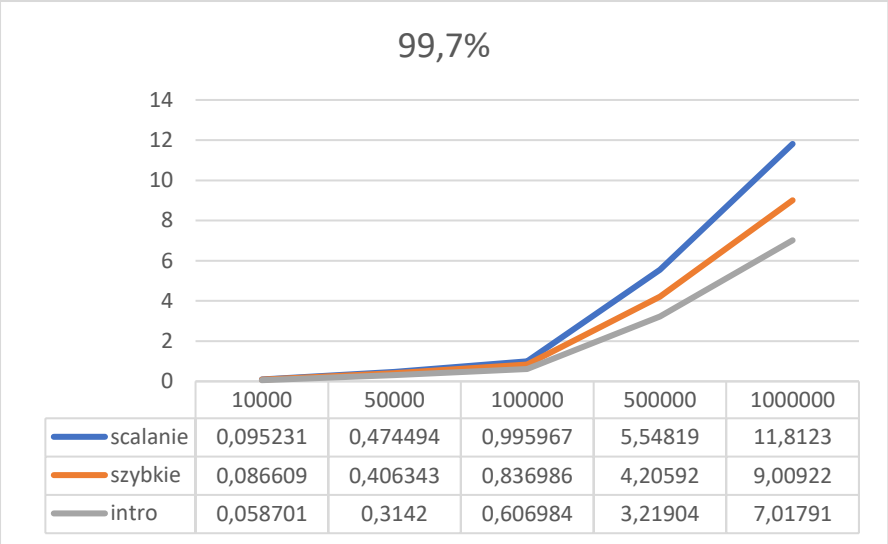

| | 10000 | 50000 | 100000 | 500000 | 1000000 |
|---|---|---|---|---|---|
| scalanie | 0,095183 | 0,500299 | 1,06691 | 5,86244 | 12,6525 |
| szybkie | 0,099364 | 0,512141 | 1,04428 | 5,30415 | 10,8741 |
| intro | 0,088559 | 0,514757 | 1,05704 | 5,18935 | 10,7941 |

3.6. Elementy posortowane w 99%



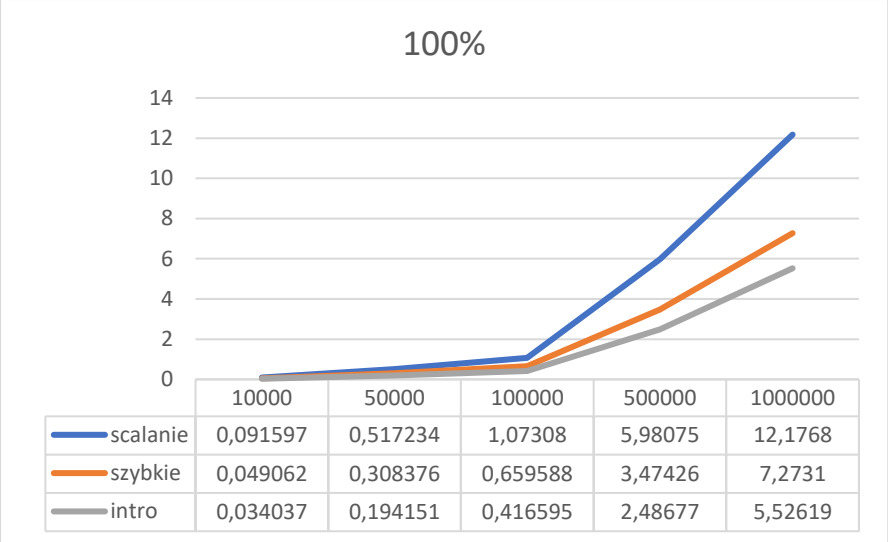

| | 10000 | 50000 | 100000 | 500000 | 1000000 |
|---|---|---|---|---|---|
| scalanie | 0,11029 | 0,478386 | 1,02756 | 5,61096 | 11,8392 |
| szybkie | 0,089032 | 0,429791 | 0,89844 | 4,50433 | 9,43415 |
| intro | 0,06613 | 0,355108 | 0,748736 | 3,77059 | 8,11811 |

3.7. Elementy posortowane w 99,7%




| | 10000 | 50000 | 100000 | 500000 | 1000000 |
|---|---|---|---|---|---|
| scalanie | 0,095231 | 0,474494 | 0,995967 | 5,54819 | 11,8123 |
| szybkie | 0,086609 | 0,406343 | 0,836986 | 4,20592 | 9,00922 |
| intro | 0,058701 | 0,3142 | 0,606984 | 3,21904 | 7,01791 |

3.8. Elementy posortowane w odwrotnej kolejności




| | 10000 | 50000 | 100000 | 500000 | 1000000 |
|---|---|---|---|---|---|
| scalanie | 0,091597 | 0,517234 | 1,07308 | 5,98075 | 12,1768 |
| szybkie | 0,049062 | 0,308376 | 0,659588 | 3,47426 | 7,2731 |
| intro | 0,034037 | 0,194151 | 0,416595 | 2,48677 | 5,52619 |

# 4. Wnioski

- Z powyższych testów wynika, że w większości przypadków najszybszy jest algorytm sortowania introspektywnego.
- W przypadku tablicy wstępnie posortowanej w 50% najszybszy okazał się algorytm sortowania przez scalanie jest to spowodowane sposobem implementacji algorytmów quicksort i mergesort. Wybierają one pivot podczas wykonywania rekurencji jako środkowy fragment danych.
- Pomiędzy algorytmami quicksort i introsort nie ma dużych różnic w czasie wykonywania, natomiast najbardziej odbiegającym algorytmem jest mergesort, który jest najwolniejszy w większości przypadków

## 5. Literatura


- https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie
- https://pl.wikipedia.org/wiki/Sortowanie_szybkie
- https://pl.wikipedia.org/wiki/Sortowanie_introspektywne
- https://www.geeksforgeeks.org/introsort-or-introspective-sort/
- https://en.cppreference.com/w/cpp/chrono