Projektowanie algorytmów i metody sztucznej inteligencji

Projekt 1

Jakub Stasiak 241591

2 IV 2019

Prowadzący zajęcia: dr inż. Łukasz Jeleń
Termin zajęć: środa 7.30-9.00

1 Wprowadzenie

Tematem projektu było zapoznanie się i porównanie ze sobą trzech algorytmów sortowań:

- sortowania przez scalanie,
- sortowania szybkiego,
- sortowania introspektywnego.

Każdy z algorytmów należało zaimplementować w programie napisanym w języku c++ oraz napisać program mierzący czas sortowania tablic o następujących rozmiarach:

- 10 000,
- 50 000,
- 100 000,
- 500 000,
- 1 000 000

dla różnych rodzajów elementów całkowitoliczbowych.
W moim programie wykorzystałem typy:

- short int,
- int,
- long int,
- long long int.

Ponadto należało rozpatrzeć przypadki:

- wszystkie elementy tablicy losowe,
- $25\%, 50\%, 75\%, 95\%, 99\%, 99,7\%$ początkowych elementów tablicy jest już posortowanych,
- wszystkie elementy tablicy już posortowane ale w odwrotnej kolejności.

Dodatkowo rozpatrzyłem przypadek, gdy wszystkie elementy są już posortowane. Przyjąłem również, że sortowanie układa elementy od najmniejszej do największej wartości. Każdą kombinację sortowań przeprowadziłem stukrotnie.

2 Sortowanie przez scalanie

Sortowanie przez scalanie można przedstawić jako metodę podziału danych na jednoelementowe części i scalanie ich w coraz to większe już posortowane grupy. Jest to metoda rekurencyjna.

Każde wywołanie rekurencji dzieli zestaw danych na pół, więc maksymalna głębokość rekurencji jest równa:

$$log_2 n.$$

Dla każdego elementu jest później wykonywane porównanie i scalenie, którego złożoność obliczeniowa jest równa:

$$2n.$$

Ponadto w mojej implementacji wykonywane jest również przekopiowanie tablicy oraz przepisanie jej na końcu, więc ostateczna złożoność obliczeniowa jest równa:

$$O(2n \cdot log_2 n + 2n) = O(n \cdot log_2 n).$$

Nawet w najgorszym przypadku nie jest zmieniana asymptotyczna złożoność obliczeniowa.

3 Sortowanie szybkie

Sortowanie szybkie również jest algorytmem wykorzystującym rekurencję. W każdym kroku zostaje wybrany element zwany „flagą" lub „pivotem", który dzieli tablicę na 2 części: z elementami większymi oraz mniejszymi od flagi.

Złożoność obliczeniowa w przypadku optymistycznym oraz średnim wynosi

$$O(n \cdot log_2 n),$$

a w przypadku pesymistycznym

$$O(n^2).$$

Przypadek optymistyczny zachodzi, kiedy przy każdym podziale zestawu danych flaga jest równa medianie. Przypadek pesymistyczny zachodzi, gdy za każdym razem flagą zostaje element największy lub najmniejszy (np. kiedy zestaw jest posortowany, a flagą zostaje ostatni element). W mojej implementacji znacząco zmniejszyłem prawdopodobieństwo tego zdarzenia wybierając jako flagę środkowy element zestawu danych.

4 Sortowanie introspektywne

Sortowanie introspektywne jest algorytmem hybrydowym łączącym szybkie sortowanie z sortowaniem przez kopcowanie. Zaczynamy od sortowania szybkiego, a kiedy rozmiar zestawu danych jest równy lub prawie równy 10, przełączamy się na sortowanie przez kopcowanie. Zaletą jest dobra wydajność sortowania szybkiego przy typowych zestawach liczb oraz wydajność w pesymistycznym przypadku dzięki sortowaniu przez kopcowanie. Praktyczna wydajność we wszystkich przypadkach jest równa:

$$O(n \cdot log_2 n).$$

5 Wyniki pomiarów

Wszędzie podany uśredniony czas dla jednego sortowania.

5.1 Sortowanie przez scalanie

short int

	10k	50k	100k	500k	1M
0,00%	1	7	17	90	187
25,00%	1	6	14	80	170
50,00%	1	6	12	73	152
75,00%	0	5	11	63	127
95,00%	0	4	9	53	111
99,00%	0	4	9	53	108
99,70%	0	4	9	53	107
odwrotnie	0	4	9	52	109
posortowane	0	4	9	51	107

long int

	10k	50k	100k	500k	1M
0,00%	1	7	16	93	201
25,00%	1	7	14	83	176
50,00%	1	6	12	73	153
75,00%	0	5	11	62	131
95,00%	0	4	9	54	115
99,00%	0	4	9	54	112
99,70%	0	4	9	52	112
odwrotnie	0	4	9	53	112
posortowane	0	4	9	52	112

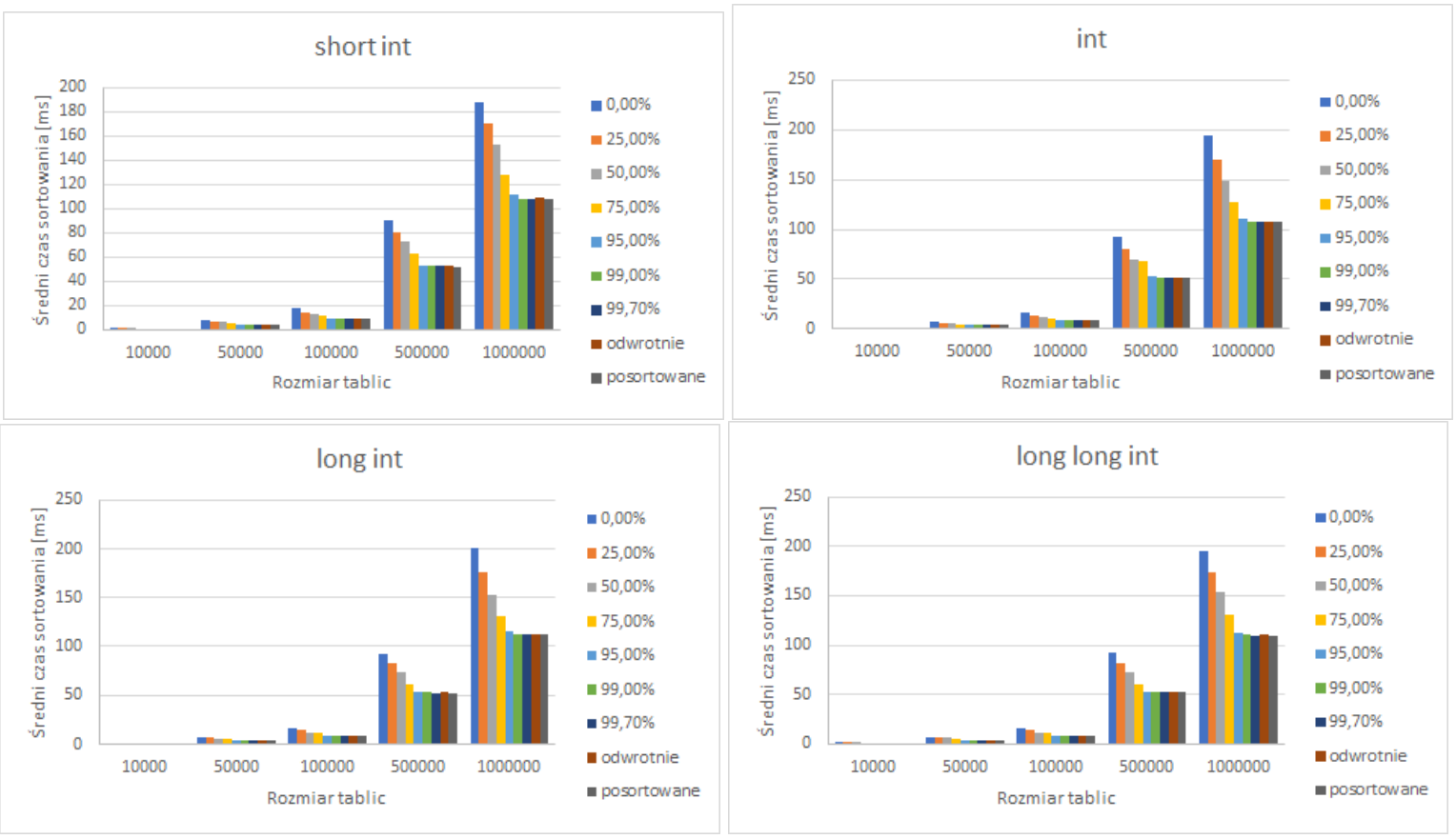
int

	10k	50k	100k	500k	1M
0,00%	1	7	16	92	194
25,00%	1	6	14	81	170
50,00%	1	6	12	70	148
75,00%	0	5	10	68	127
95,00%	0	4	9	53	110
99,00%	0	4	9	51	108
99,70%	0	4	9	51	107
odwrotnie	0	4	9	51	107
posortowane	0	4	9	51	107

long long int

	10k	50k	100k	500k	1M
0,00%	1	7	16	92	195
25,00%	1	6	14	82	173
50,00%	1	6	12	72	153
75,00%	0	5	11	61	131
95,00%	0	4	9	53	113
99,00%	0	4	9	52	111
99,70%	0	4	9	52	110
odwrotnie	0	4	9	52	111
posortowane	0	4	9	52	110

Rysunek 1: Wyniki pomiaru czasu dla sortowania przez scalanie. Wartości podane w milisekundach



Rysunek 2: Wykresy zależności czasu od ilości danych dla sortowania przez scalanie dla różnych typów danych

5.2 Szybkie sortowanie

short int

	10k	50k	100k	500k	1M
0,00%	1	8	18	108	228
25,00%	1	8	18	110	247
50,00%	2	8	19	111	311
75,00%	3	243	28	102	212
95,00%	1	98	21	132	278
99,00%	0	25	23	189	419
99,70%	0	11	18	203	607
odwrotnie	0	4	10	61	141
posortowane	0	4	9	57	127

long int

	10k	50k	100k	500k	1M
0,00%	1	8	18	110	233
25,00%	1	8	18	93	209
50,00%	1	6	14	83	173
75,00%	0	5	12	67	145
95,00%	0	4	10	58	122
99,00%	0	4	10	56	119
99,70%	0	4	10	56	120
odwrotnie	0	4	9	57	120
posortowane	0	4	9	56	118

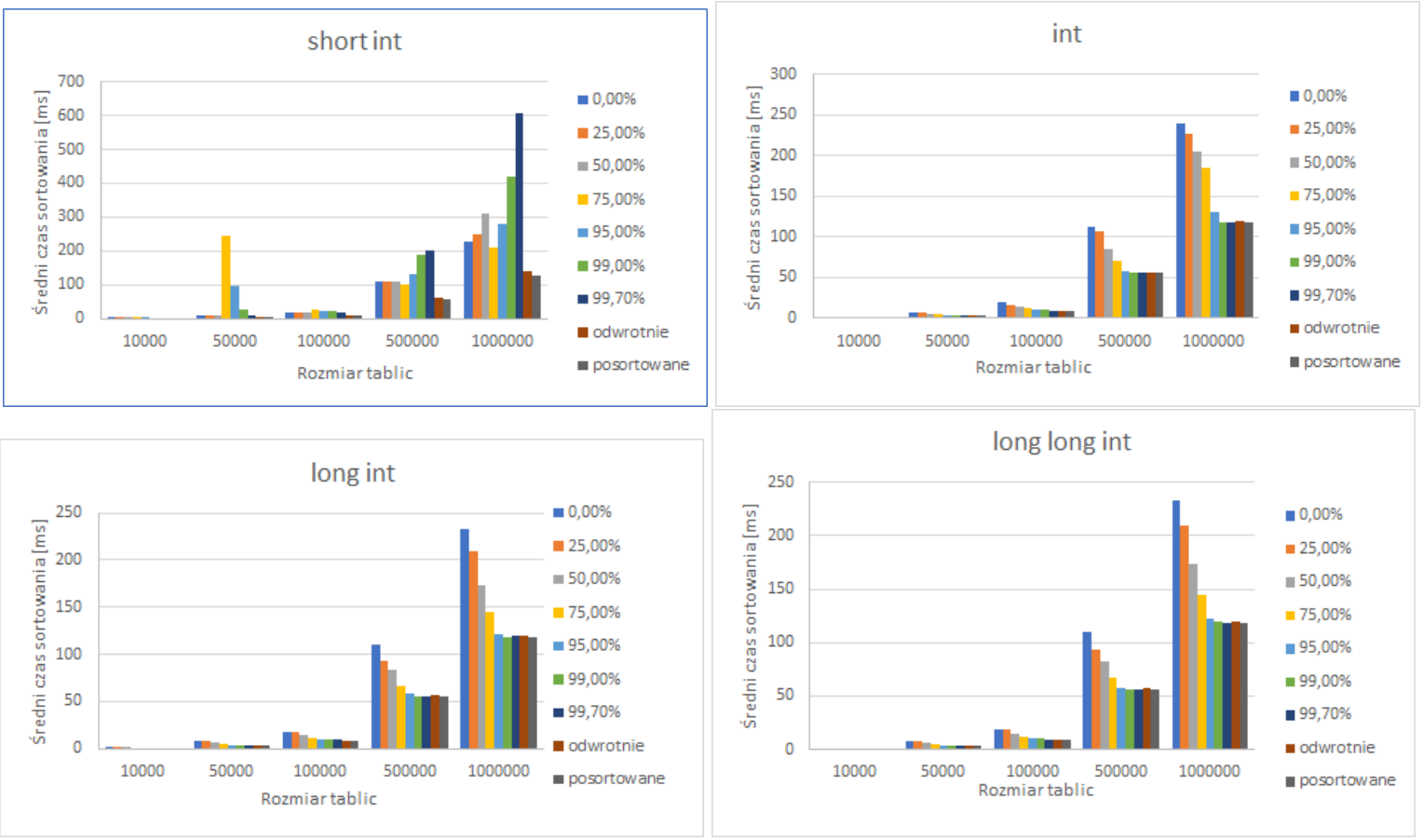
int

	10k	50k	100k	500k	1M
0,00%	1	8	19	112	239
25,00%	1	7	16	108	227
50,00%	1	6	14	86	206
75,00%	0	5	12	71	186
95,00%	0	4	10	58	131
99,00%	0	4	10	56	118
99,70%	0	4	9	56	118
odwrotnie	0	4	9	57	120
posortowane	0	4	9	57	118

long long int

	10k	50k	100k	500k	1M
0,00%	1	8	18	110	233
25,00%	1	8	18	93	209
50,00%	1	6	14	82	173
75,00%	0	5	12	67	145
95,00%	0	4	10	58	122
99,00%	0	4	10	56	119
99,70%	0	4	9	56	118
odwrotnie	0	4	9	57	120
posortowane	0	4	9	56	118

Rysunek 3: Wyniki pomiaru czasu dla sortowania szybkiego. Wartości podane w milisekundach



Rysunek 4: Wykresy zależności czasu od ilości danych dla sortowania szybkiego dla różnych typów danych

5.3 Sortowanie introspektywne

short int

	10k	50k	100k	500k	1M
0,00%	3	7	12	89	153
25,00%	0	7	11	89	129
50,00%	0	7	11	89	130
75,00%	0	7	11	89	129
95,00%	0	7	11	89	129
99,00%	0	7	11	89	129
99,70%	0	7	11	89	129
odwrotnie	0	7	11	87	144
posortowane	0	7	11	89	130

long int

	10k	50k	100k	500k	1M
0,00%	0	7	11	88	154
25,00%	0	7	11	89	154
50,00%	0	7	11	89	156
75,00%	0	7	11	89	154
95,00%	0	7	11	88	154
99,00%	0	7	11	88	153
99,70%	0	7	11	88	154
odwrotnie	0	7	11	89	153
posortowane	0	7	11	89	156

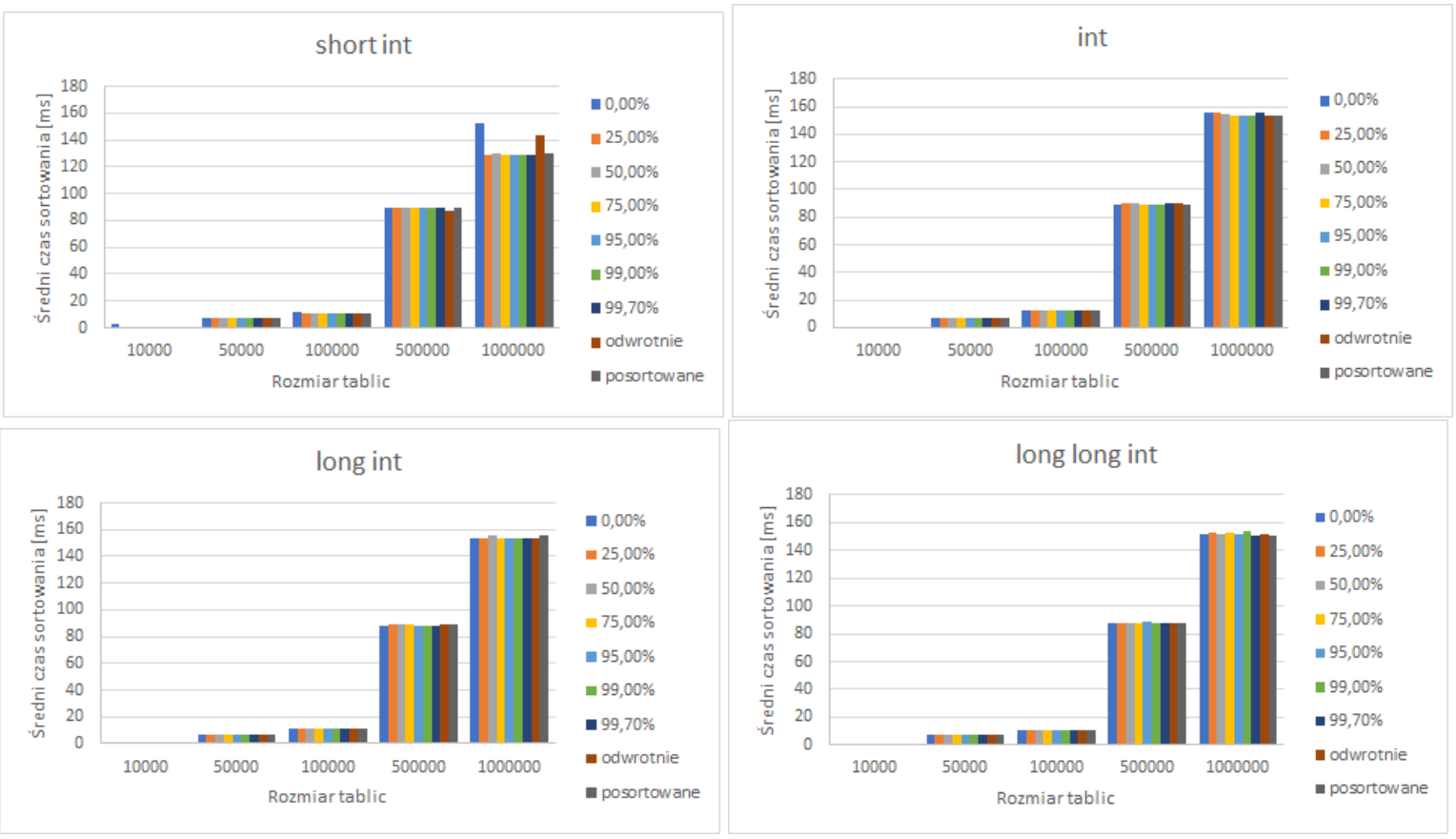
int

	10k	50k	100k	500k	1M
0,00%	0	7	12	89	156
25,00%	0	7	12	90	156
50,00%	0	7	12	90	155
75,00%	0	7	12	89	153
95,00%	0	7	12	89	154
99,00%	0	7	12	89	154
99,70%	0	7	12	90	156
odwrotnie	0	7	12	90	154
posortowane	0	7	12	89	154

long long int

	10k	50k	100k	500k	1M
0,00%	0	7	11	88	152
25,00%	0	7	11	88	153
50,00%	0	7	11	88	152
75,00%	0	7	11	88	153
95,00%	0	7	11	89	152
99,00%	0	7	11	88	154
99,70%	0	7	11	88	151
odwrotnie	0	7	11	88	152
posortowane	0	7	11	88	151

Rysunek 5: Wyniki pomiaru czasu dla sortowania introspektywnego. Wartości podane w milisekundach



Rysunek 6: Wykres zależności czasu od ilości danych dla sortowania introspektywnego dla różnych typów danych

6 Podsumowanie

We wszystkich przypadkach tablic o rozmiarze $n = 10000$ średni czas sortowania jest bliski $t = 0ms$, więc trudno jest wyciągać z tej części danych jakiekolwiek wnioski, gdyż dla tak krótkich czasów trudno o dokładność.

6.1 Sortowanie przez scalanie

Dla sortowania przez scalanie czas znacząco skracał się wraz ze wstępnym posortowaniem tablic. Pomiędzy przypadkiem 0% oraz wstępnie posortowanym zestawem różnica jest nawet dwukrotna. Nie występują tu znaczące różnice pomiędzy typami danych. Ponadto wydajność we wszystkich przypadkach wskazuje na złożoność obliczeniową niewiele większą od $O(n)$.

6.2 Szybkie sortowanie

Wystąpiła tutaj anomalia dla rozmiaru zestawu: $n = 50000$ dla typu "short int". Dla przypadku wstępnie posortowanych części: 75% oraz 95% czas sortowania znacząco się wydłużył. Nie jestem pewny czym zostało to spowodowane, ale podejrzewam, że ma to związek z algorytmem wyboru flagi. Ponadto zauważalne są różnice pomiędzy typem „short int" oraz pozostałymi. Możliwe, że jest to związane z ograniczoną maksymalną wartością elementów, a przez to wielokrotnym występowaniem tej właśnie wartości w zestawie. Znalazłem informację, że w takim wypadku sortowanie szybkie traci na wydajności. Dla typów „long int" oraz „long long int" różnice w czasie sortowania są już niezauważalne co również wskazuje na związek wydajności tej metody oraz powtarzania się wartości elementów w zestawie danych. Pomijając typ „short int" coraz to większe wstępne posortowanie zestawu w zasadzie przyspieszało czas sortowania z wykorzystaniem tej metody.

6.3 Sortowanie introspektywne

Zgodnie z teoretycznym modelem okazało się, że jest to najbardziej niezależna od wstępnego posortowania oraz typu danych metoda sortowania. Jest ona szybsza od metody scalania w przypadkach 0% oraz 25%, tak samo szybka w przypadku 50% oraz wolniejsza w pozostałych przypadkach. Ponadto jest ona szybsza od szybkiego sortowania w prawie wszystkich przypadkach dla typu „short int" oraz dla pozostałych typów w przypadkach $0\%, 25\%$ oraz 50%, a także dodatkowo od typu „int" w przypadku 75%.

7 Literatura

Podczas wykonywania projektu wspierałem się nastepującymi stronami:

1. `https://en.wikipedia.org/wiki/Introsort`
2. `https://pl.wikipedia.org/wiki/Sortowanie_szybkie`
3. `https://en.wikipedia.org/wiki/Quicksort`
4. `https://en.wikipedia.org/wiki/Merge_sort`
5. `https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie`
6. `https://pl.wikipedia.org/wiki/Sortowanie_introspektywne`
7. `https://pl.wikipedia.org/wiki/Sortowanie_przez_kopcowanie/`
8. `https://forum.pasja-informatyki.pl/274147/sortowanie-merge-sort-c`

9. http://www.algorytm.org/algorytmy-sortowania/sortowanie-przez-scalanie-mergesort/merge-c.html
10. https://stackoverflow.com/
11. http://cpp0x.pl/kursy/Kurs-C++/
12. https://www.p-programowanie.pl/
13. http://www.cplusplus.com/
14. https://4programmers.net
15. https://en.cppreference.com/
16. https://www.geeksforgeeks.org/