Projektowanie algorytmów i metody sztucznej inteligencji

Prowadzący	Wykonawca	Nr. Indeksu	Data
dr inż. Łukasz Jeleń	Patryk Marciniak	248978	26.03.2020

1 Wstęp

Projekt polegał na zaimplementowaniu i przetestowaniu następujących algorytmów:

- Merge Sort (sortowanie przez scalanie)
- Quick Sort (sortowanie szybkie)
- Intro Sort (sortowanie introspektywne)

Wymienione wyżej algorytmy należą do sortowań zaawansowanych tzn. sortowań o logarytmicznej złożoności obliczeniowej. Wyjątkiem jest jednak najgorszy przypadek sortowania szybkiego, którego złożoność obliczeniowa wynosi $O(n^2)$. Z tych trzech sortowań, jedynym sortowaniem stabilnym jest Merge Sort, oznacza to, że występujące po sobie elementy o jednakowych wartościach nie zostaną zamienione miejscami w procesie sortowania. W projekcie rozważane były następujące rozmiary tablic:

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

Należało także rozważyć następujące stopnie posortowania:

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

2 Opis algorytmów

2.1 Merge Sort

Sortowanie przez scalanie jest algorytmem rekurencyjnym wykorzystujacym metodę *dziel i zwyciężaj*. Tablica w każdym kroku zostaje podzielona na dwie części aż do powstania tablic jednoelementowych. Następnie porównując pary tablic algorytm scala je układając ich elementy w odpowiedniej kolejności.Merge Sort jest algorytmem stabilnym.
Złożoność obliczeniowa i pamięciowa Merge Sorta prezentują się następująco:

$$\begin{aligned} \text{Najlepszy przypadek} &\longrightarrow O(n \times log(n)) \\ \text{Średni przypadek} &\longrightarrow O(n \times log(n)) \\ \text{Najgorszy przypadek} &\longrightarrow O(n \times log(n)) \\ \text{Złożoność pamięciowa} &\longrightarrow O(n) \end{aligned}$$

2.2 Quick Sort

Sortowanie szybkie podobnie jak merge sort jest również algorytmem rekurencyjnym działającym na zasadzie *dziel i zwyciężaj*. W każdym kroku sortowania zostaje wybrany element służacy do podziału tablicy (tzw. pivot). Na jego podstawie tablica zostaje podzielona na dwie nowe tablice, jedną z elementami mniejszymi od pivota, a drugą z większymi. Wybrany element nie bierze udziału w dalszym sortowaniu, ponieważ jest już na swojej pozycji. Quick Sort jest algorytmem niestabilnym i jego wydajność zależy m. in. od metody doboru pivota. Złożoność obliczeniowa i pamięciowa Quick Sorta wygląda następująco:

$$\begin{aligned}
&\text{Najlepszy przypadek} &&\longrightarrow O(n \times log(n))\\
&\text{Średni przypadek} &&\longrightarrow O(n \times log(n))\\
&\text{Najgorszy przypadek} &&\longrightarrow O(n^2)\\
&\text{Złożoność pamięciowa} &&\longrightarrow O(n) \text{ lub } O(log(n))
\end{aligned}$$

2.3 Introspective Sort

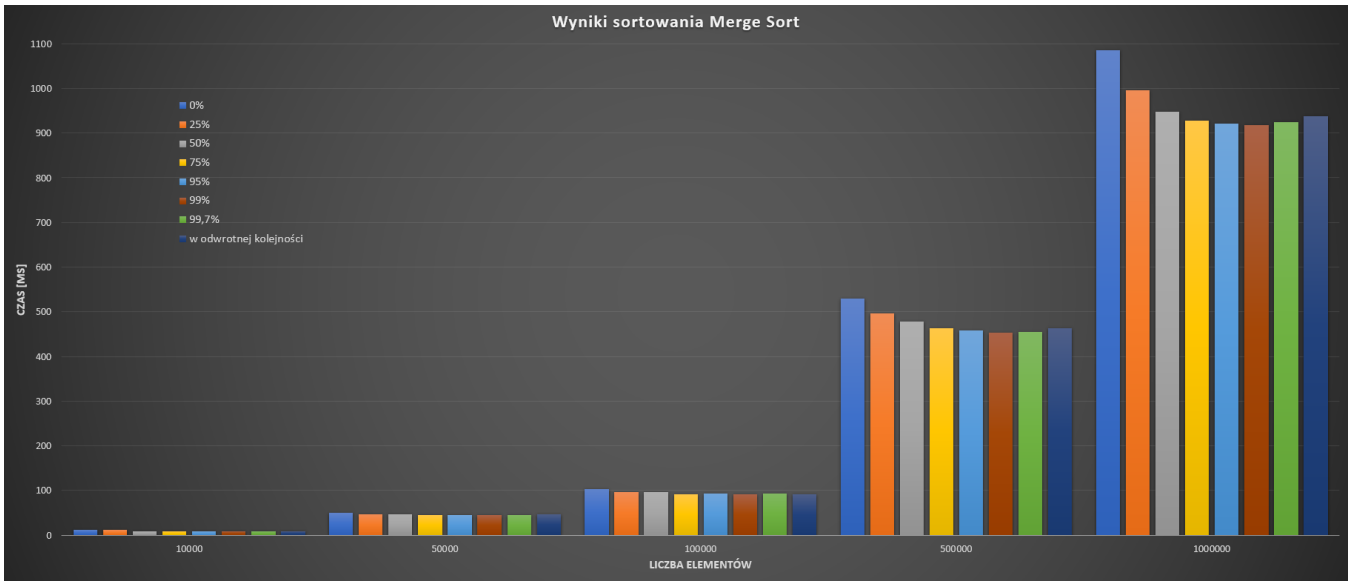
Sortowanie introspektywne jest odmianą sortowania hybrydowego, co oznacza, że zawiera w sobie więcej niż jeden algorytm sortowania. Konkretniej, w jego skład wchodzą sortowanie szybkie, sortowanie przez kopcowanie i sortowanie przez wstawianie. Intro sort dzieli tablicę podobnie do tego, jak robi to Quick Sort, jednak jeśli głębokość rekurencji będzie większa od dwukrotności logarytmu naturalnego z rozmiaru tablicy, to algorytm do sortowania wykorzysta Heap Sorta. Jeśli zaś w pozostałym fragmencie tablicy jest mniej niż 16 elementów to algorytm wykorzysta sortowanie przez wstawianie. Sortowanie introspektywne jest sortowaniem niestabilnym stworzonym, aby wyeliminować najgorszy przypadek złożoności obliczeniowej Quick Sorta (tj. $O(n^2)$).
Złożoność obliczeniowa i pamięciowa Quick Sorta wygląda następująco:

$$\begin{aligned}
&\text{Najlepszy przypadek} &&\longrightarrow O(n \times log(n))\\
&\text{Średni przypadek} &&\longrightarrow O(n \times log(n))\\
&\text{Najgorszy przypadek} &&\longrightarrow O(n \times log(n))\\
&\text{Złożoność pamięciowa} &&\longrightarrow O(log(n))
\end{aligned}$$

3 Otrzymane wyniki

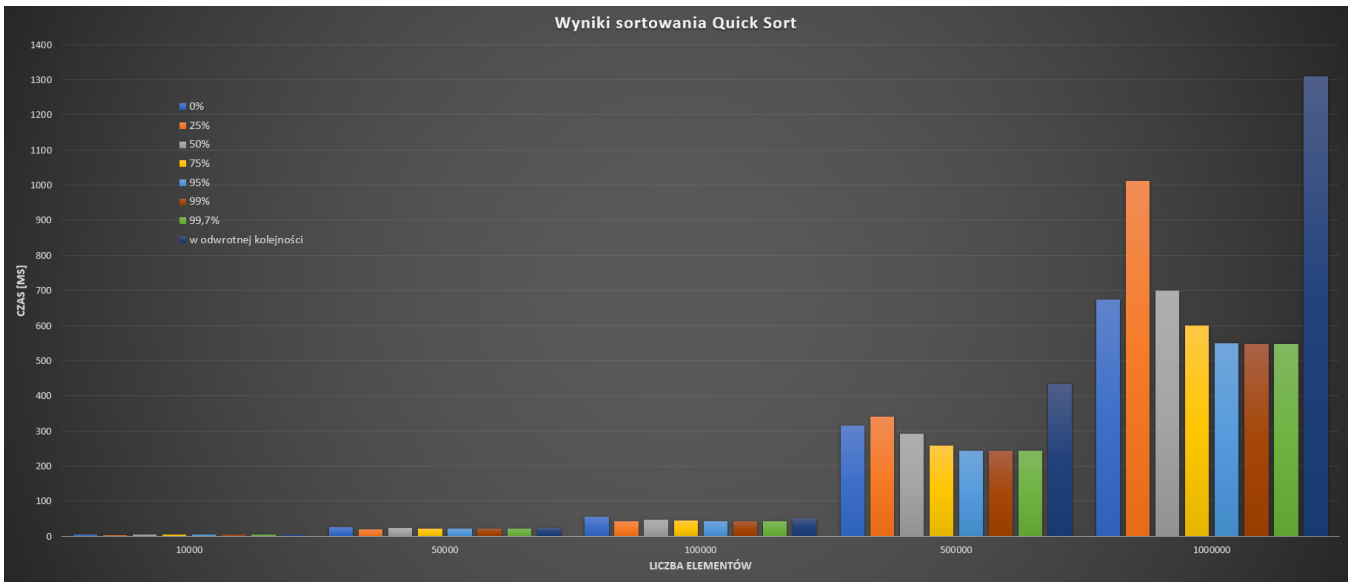
3.1 Sortowanie przez scalanie (Merge Sort)

[ms]		0%	25%	50%	75%	95%	99%	99,7%	od końca
10 tys.	min	9,5004	9,0616	8,7913	8,6509	8,6014	8,6108	8,6199	8,6385
	max	19,4139	16,9259	9,8267	18,0779	12,892	10,4536	9,7504	11,1371
	avg	12,4826	11,5354	8,94191	8,98932	8,7469	8,77536	8,79134	8,7062
50 tys.	min	49,0087	46,2659	44,8182	44,0218	43,7558	43,7541	43,8002	43,9253
	max	53,9845	51,5456	55,1101	53,031	83,3197	53,4064	49,044	53,1565
	avg	49,9977	47,5536	47,0209	45,5407	45,9104	45,6847	45,6165	46,1545
100 tys.	min	99,7409	94,0385	90,9228	89,153	88,5061	88,4593	88,6083	88,9592
	max	110,084	103,376	149,266	101,966	164,709	109,14	129,842	98,5005
	avg	103,392	97,2636	96,0226	92,19	92,4035	91,229	93,1158	92,0327
500 tys.	min	511,247	478,504	460,406	451,072	447,562	447,471	447,088	452,007
	max	555,618	523,32	633,664	596,98	567,923	473,387	483,903	497,06
	avg	529,418	496,137	477,912	463,634	458,4	453,738	454,137	463,972
1 mil.	min	1044,17	971	934,505	915,505	915,495	906,419	906,415	919,606
	max	1133,45	1051,56	982,259	958,971	963,581	964,105	982,726	1154,09
	avg	1086,08	996,498	948,344	928,261	920,271	918,405	924,421	938,74



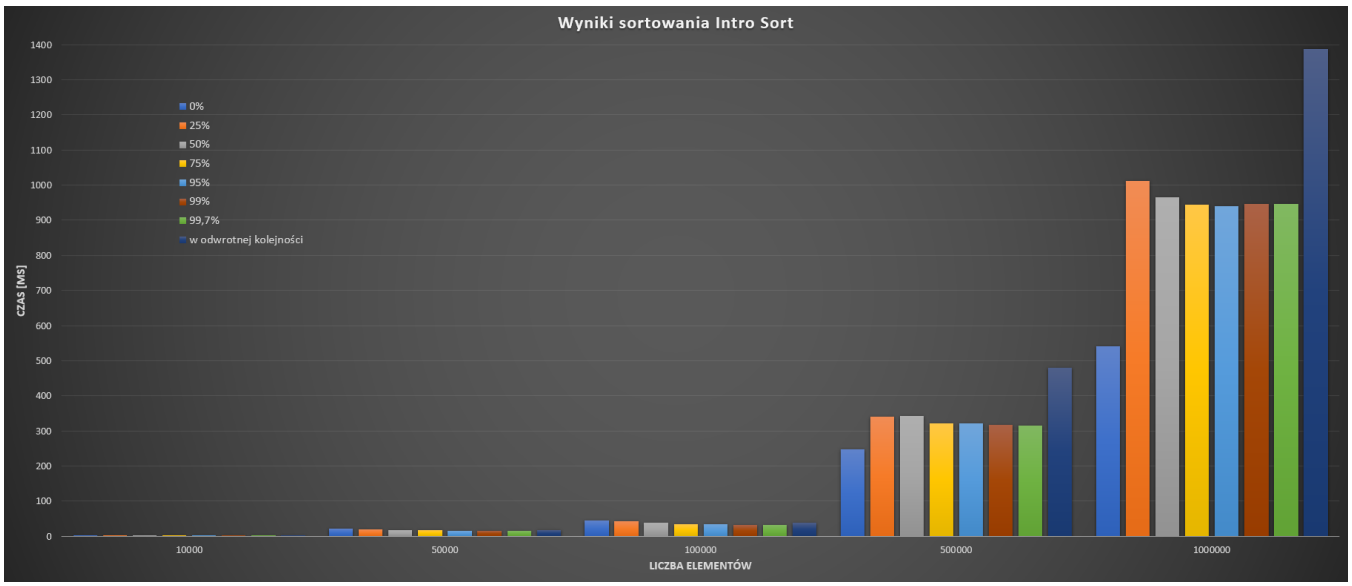
3.2 Sortowanie szybkie (Quick Sort)

[ms]		0%	25%	50%	75%	95%	99%	99,7%	od końca
10 tys.	min	4,3981	4,0692	3,8085	3,6353	3,5521	3,504	3,5115	3,864
	max	5,9617	6,0045	4,525	4,3651	6,3387	6,2951	4,3605	4,6478
	avg	4,8119	4,42157	4,08575	3,93029	3,86825	3,86107	3,82481	4,14303
50 tys.	min	24,8959	22,9879	21,332	20,6231	19,8081	20,0695	20,0845	21,5064
	max	29,2974	27,4155	27,6354	24,7293	22,5598	24,2554	22,4353	27,6242
	avg	26,4608	24,6673	23,0982	21,8041	21,1572	21,1272	21,0028	23,2966
100 tys.	min	52,8717	48,8261	45,0969	42,9159	41,7074	41,9002	41,8251	46,4592
	max	63,0162	57,646	51,8902	49,6942	47,5699	47,7549	46,6631	56,6523
	avg	55,8115	51,715	47,7196	44,9121	43,4725	43,3596	43,3818	50,7675
500 tys.	min	300,905	297,166	276,926	251,292	237,901	235,986	237,489	373,73
	max	338,418	361,668	323,364	275,266	250,887	251,408	257,44	486,291
	avg	314,8	320,739	291,386	258,726	243,797	243,511	243,483	432,229
1 mil.	min	649,809	699,437	653,821	576,54	536,801	532,428	534,926	1153,36
	max	717,067	859,901	773,11	630,752	564,31	564,346	563,688	1475,2
	avg	674,508	770,394	700,079	599,146	549,833	547,396	546,876	1309,69



3.3 Sortowanie introspektywne (Intro Sort)

[ms]		0%	25%	50%	75%	95%	99%	99,7%	od końca
10 tys.	min	2,937	2,6183	2,3608	2,1551	2,1188	2,1337	2,0981	2,2909
	max	7,4163	5,9133	3,5599	3,4948	3,4431	3,0895	3,3239	3,551
	avg	3,46748	3,03474	2,78361	2,58249	2,51763	2,49668	2,52895	2,70912
50 tys.	min	18,4084	16,1908	15,2301	14,0329	13,7586	13,5621	13,9788	14,9015
	max	24,2017	23,4194	24,8262	20,6855	19,745	19,1188	19,226	22,7278
	avg	20,4854	19,1879	17,838	16,684	15,6366	15,6986	15,7686	17,3141
100 tys.	min	39,8065	34,9179	31,9389	29,7732	29,2502	29,841	28,8099	32,2943
	max	54,9287	54,2222	44,7511	40,0371	41,8766	39,4288	42,3329	55,3987
	avg	43,6718	41,5591	37,4761	33,7111	33,2439	32,776	32,8858	39,3477
500 tys.	min	228,132	267,74	282,176	261,866	256,125	263,784	249,868	393,889
	max	302,346	486,821	412,904	392,07	402,831	383,912	380,582	608,642
	avg	247,525	339,601	341,995	322,059	320,333	317,851	315,185	479,212
1 mil.	min	499,405	881,642	880,826	848,888	860,757	864,984	848,789	1225,87
	max	646,38	1245,5	1089,99	1054,09	1026,86	1038,98	1156	1498,27
	avg	541,49	1011,31	964,259	944,32	939,285	945,619	945,426	1388,61



4 Podsumowanie i wnioski

Zgodnie z założeniami najwolniej z testowanych algorytmów wypada *Merge Sort*, a *Quick Sort* jest generalnie wolniejszy od *Intro Sorta*. W przypadku wszystkich trzech algorytmów, wcześniejsze posortowanie części tablicy skraca ogólny czas sortowania, jednak w przypadku sortowania szybkiego i introspektywnego, dla tablic o większej ilości elementów, gdzie posortowane jest pierwsze 25% z nich, można zauważyć nagły wzrost w czasie potrzebnym na sortowanie. Podobne wyniki uzyskuje się dla tablic posortowanych w odwrotnej kolejności. Te wyniki można wytłumaczyć niezbyt optymalną implementacją tych algorytmów lub chociażby nienajlepszą metodą dobierania *pivota* dzielącego tablicę na dwie części. Mimo tych anomalii, wszystkie algorymy zdają się spełniać założenie o swoich złożonościach obliczeniowych - tempo wzrostu czasów sortowań jest zbliżone do *nlog(n)*.
Powyższe wyniki mniej więcej potwierdzają założenia o badanych algorytmach i ich złożoności obliczeniowej oraz poprawność ich implementacji, mimo podejrzeń o suboptymalności algorytmów.

5 Literatura

- https://www.geeksforgeeks.org/merge-sort/
- https://en.wikipedia.org/wiki/Merge_sort
- https://www.tutorialspoint.com/cplusplus-program-to-implement-merge-sort
- https://pl.wikipedia.org/wiki/Sortowanie_szybkie
- https://www.geeksforgeeks.org/heap-sort/
- https://en.wikipedia.org/wiki/Introsort
- https://www.geeksforgeeks.org/introsort-or-introspective-sort/