Nazwa kursu: Projektowanie Algorytmów i Metody Sztucznej Inteligencji

Tytuł: Projekt 1 - sortowania

Data oddania: 26.03.2020 r.

Termin zajęć: Piątek 7:30

Prowadzący: mgr inż. Marta Emirsajłow

Dane studenta: Konrad Arent 243646

1. Cel ćwiczenia

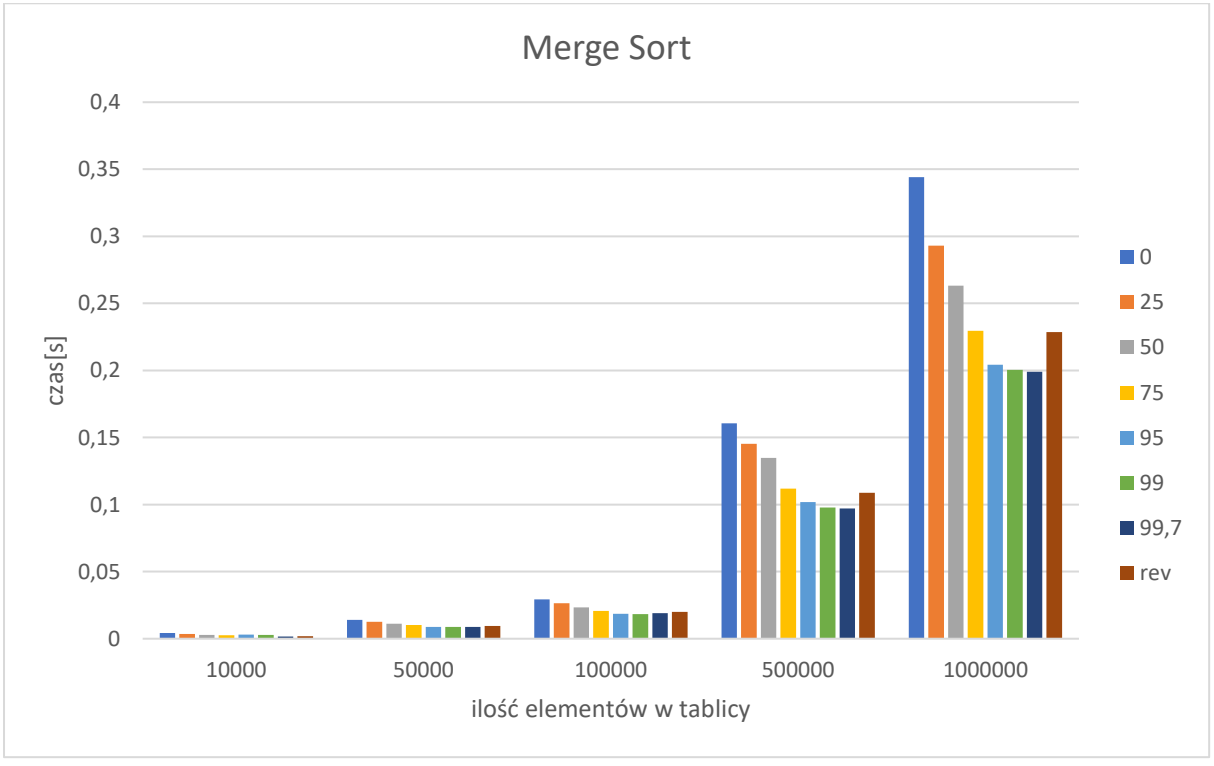
Celem projektu było zaimplementowanie trzech algorytmów (odpowiednio wybranych z tabeli w zadaniu projektowym) sortowania: sortowanie przez scalanie (Merge Sort), sortowanie przez kopcowanie (Heap Sort) oraz sortowanie Shella (Shell Sort). Należało porównać algorytmy, przeprowadzając pomiary czasu sortowania stu tablic o ilości elementów 10.000, 50.000, 100.000, 500.000 i 1.000.000 w różnych stopniach wstępnego posortowania: 0%, 25%, 50%, 75%, 95%, 99%, 99.7% oraz tablicy posortowanej, lecz w odwrotnej kolejności

2. Sortowanie przez scalanie (Merge Sort)

Sortowanie przez scalanie jest algorytmem rekurencyjnym z grupy algorytmów szybkich. Tablica w każdym kroku zostaje podzielona na dwie części (dwa mniejsze podproblemy), aż do powstania tablic jednoelementowych, które są uważane za posortowane. Następnie, algorytm porównuje „lewy podproblem" oraz „prawy podproblem" i scala je (merge) w odpowiedniej kolejności. Merge-sort w każdym wywołaniu rekurencji dzieli tablicę na pół. Dla tablicy o rozmiarze 2 algorytm „zejdzie o jeden poziom rekurencji w dół", dla 4 o dwa, dla 8 o trzy. Można więc zauważyć, że maksymalna głębokość rekurencji wynosi log2 n, gdzie n to rozmiar tablicy. Dla każdego elementu wykonywane jest porównanie i scalenie (merge), więc ostateczna złożoność obliczeniowa wynosi O(nlog2 n). Rozpatrując przypadek najgorszy można zauważyć, że zwiększa się jedynie ilość potrzebnych porównań podczas scalania elementów, więc nie powoduje to zmiany złożoności obliczeniowej. Aby móc swobodnie poruszać się w tym algorytmie po elementach potrzebujemy dodatkowej tablicy o takim samym rozmiarze jak nasza tablica do posortowania wiec złożoność pamięciowa dodatkowych struktur dla sortowania przez scalanie wynosi to O(n).

Tabela 1 Merge sort

		10000	50000	100000	500000	1000000
0%	min	0,002	0,013	0,027	0,152	0,314
	max	0,02	0,019	0,036	0,214	0,478
	średnia	0,00418	0,0141	0,02933	0,16056	0,34419
25%	min	0,002	0,12	0,024	0,136	0,281
	max	0,006	0,16	0,036	0,193	0,337
	średnia	0,00352	0,0126	0,02649	0,14526	0,29297
50%	min	0,002	0,01	0,022	0,119	0,25
	max	0,005	0,016	0,032	0,188	0,318
	średnia	0,00276	0,01123	0,0233	0,13472	0,26324
75%	min	0,001	0,009	0,019	0,105	0,219
	max	0,006	0,014	0,027	0,142	0,283
	średnia	0,00257	0,01016	0,02078	0,11178	0,22953
95%	min	0,001	0,008	0,017	0,093	0,196
	max	0,015	0,012	0,024	0,14	0,226
	średnia	0,00309	0,00883	0,01849	0,10199	0,20432
99%	min	0,001	0,008	0,017	0,091	0,191
	max	0,014	0,013	0,032	0,13	0,276
	średnia	0,0028	0,00882	0,01843	0,09779	0,20038
99,7%	min	0,001	0,008	0,017	0,09	0,191
	max	0,003	0,014	0,032	0,113	0,245
	średnia	0,00174	0,00874	0,01902	0,09715	0,19898
Rev	min	0,001	0,009	0,019	0,102	0,216
	Max	0,003	0,013	0,029	0,13	0,284
	średnia	0,00181	0,0096	0,02005	0,10891	0,22855



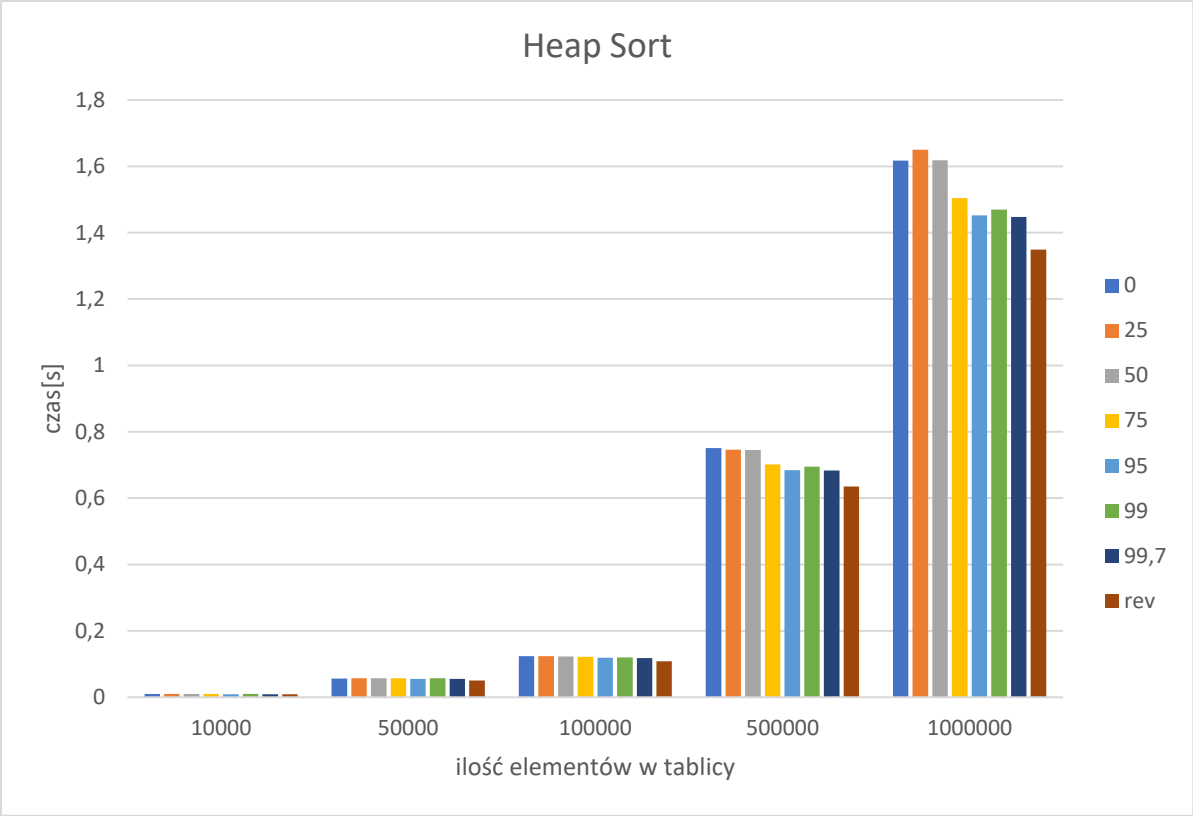
Rysunek 1 Wykres zależności czasu od ilości elementów tablicy Merge sort

3. Sortowanie przez kopcowanie (Heap Sort)

Sortowanie przez kopcowanie jest jednym z algorytmów szybkich. Do sortowania wykorzystuje tzw. kopiec, czyli kompletne drzewo binarne (jest ono zapełnione elementami maksymalnie do lewej strony). Pierwszym elementem algorytmu jest stworzenie kopca maksymalnego – takiego, w którym każdy korzeń ma wartość większą od każdego ze swoich dwóch poddrzew. W drugim etapie algorytmu następuje sortowanie właściwe. Polega ono na usunięciu wierzchołka kopca, zawierającego element maksymalny, a następnie wstawieniu w jego miejsce elementu z końca kopca i odtworzenie porządku kopca. W zwolnione w ten sposób miejsce, zaraz za końcem zmniejszonego kopca wstawia się usunięty element maksymalny. Operacje te powtarza się aż do wyczerpania elementów w kopcu. Każde wstawienie nowego elementu podczas budowania maksymalnego kopca może wymagać tyle przesunięć, ile jest węzłów kopca na ścieżce od miejsca wstawiania do korzenia drzewa. Łatwo więc można zauważyć, że liczba ta jest równa lub mniejsza od wysokości kopca, czyli log2n. Ponieważ operację tę wykonujemy (n - 1) razy, to dostajemy górne oszacowanie właśnie O(n log n).

Tabela 2 Heap Sort

		10000	50000	100000	500000	1000000
0	min	0,009	0,053	0,115	0,708	1,574
	max	0,015	0,066	0,169	0,97	1,761
	średnia	0,00972	0,05681	0,12356	0,75131	1,61698
25	min	0,008	0,053	0,115	0,714	1,591
	max	0,014	0,076	0,142	0,83	2,266
	średnia	0,00967	0,05749	0,12376	0,74575	1,65037
50	min	0,009	0,053	0,115	0,712	1,577
	max	0,013	0,078	0,14	0,96	1,86
	średnia	0,00959	0,05736	0,12305	0,74504	1,61863
75	min	0,008	0,053	0,113	0,672	1,464
	max	0,014	0,081	0,169	0,804	1,617
	średnia	0,00959	0,05711	0,12171	0,70133	1,50448
95	min	0,008	0,052	0,111	0,66	1,419
	max	0,015	0,074	0,146	0,765	1,721
	średnia	0,00942	0,05562	0,11925	0,68474	1,45234
99	min	0,008	0,052	0,111	0,657	1,415
	max	0,014	0,094	0,173	1,265	2,396
	średnia	0,00958	0,05744	0,11962	0,69535	1,46991
99,7	min	0,008	0,052	0,111	0,659	1,417
	max	0,016	0,071	0,136	0,915	1,568
	średnia	0,00949	0,05579	0,11848	0,68348	1,44774
Rev	min	0,007	0,047	0,101	0,612	1,314
	Max	0,018	0,064	0,123	0,717	1,669
	średnia	0,00886	0,05079	0,10826	0,63505	1,34946



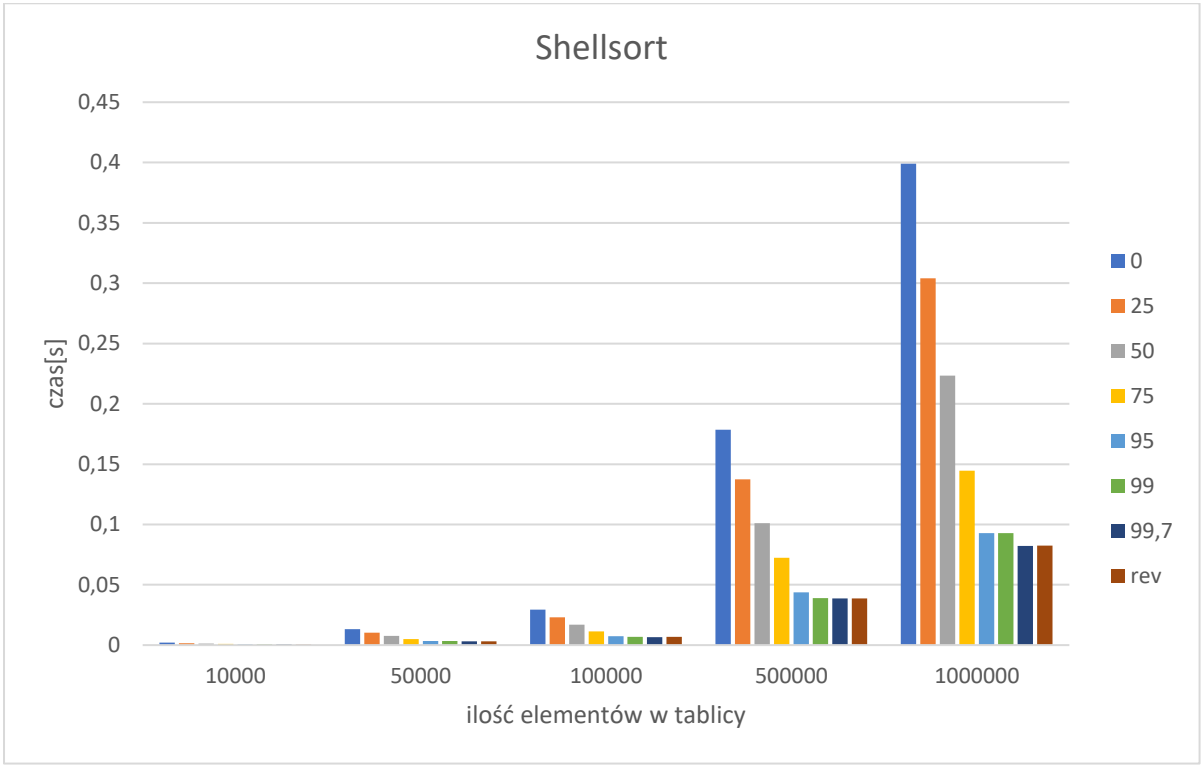
Rysunek 2 Wykres zależności czasu od ilości elementów tablicy Heap sort

4. Sortowanie Shella (Shell Sort)

Sortowanie Shella, inaczej nazywane sortowaniem przez wstawianie z malejącym odstępem, jeden z algorytmów sortowania działających w miejscu i korzystających z porównań elementów. Jest to tak naprawdę usprawnienie sortowania przez wstawianie. Różnica jest taka, że porównywane elementy (element i jego sąsiad) nie leżą obok siebie, lecz są oddalone o pewną odległość. Odległością zaproponowaną przez Shella jest n 2 k gdzie n jest liczbą elementów, a k to numer kolejnej iteracji (jest to nic innego jak dzielenie dystansu przez dwa przy każdej iteracji). Po wybraniu w pierwszym etapie tego algorytmu dystansu, wykonujemy sortowanie przez wstawianie w każdej iteracji, zmniejszając odległość zgodnie z wybraną regułą aż do dystansu równego 1. Złożoność czasowa sortowania Shella zależna jest od wybranej reguły przyjmowanie odstępów. Każdy z nich charakteryzuje się inną złożonością obliczeniową. Nie da się określić dokładnie matematycznie średniej złożoności obliczeniowej Shell Sorta, jednak w zależności od wyboru dystansu może wynosić od $O(n$ 2) do $n * (\log n)$ 2 .

Tabela 3 Shell Sort

		10000	50000	100000	500000	1000000
0%	min	0,001	0,012	0,027	0,167	0,372
	max	0,003	0,016	0,034	0,214	0,507
	średnia	0,00206	0,01327	0,02944	0,17842	0,39908
25%	min	0,001	0,009	0,021	0,129	0,288
	max	0,002	0,017	0,028	0,16	0,386
	średnia	0,0016	0,01042	0,02309	0,13727	0,30404
50%	min	0,001	0,007	0,016	0,094	0,21
	max	0,002	0,01	0,021	0,119	0,302
	średnia	0,00136	0,00776	0,01704	0,10099	0,22326
75%	min	0	0,004	0,01	0,062	0,135
	max	0,002	0,007	0,015	0,119	0,164
	średnia	0,00095	0,00507	0,01133	0,07235	0,14464
95%	min	0	0,003	0,007	0,04	0,086
	max	0,001	0,005	0,01	0,062	0,119
	średnia	0,00076	0,0034	0,0075	0,04374	0,09279
99%	min	0	0,003	0,006	0,036	0,078
	max	0,001	0,005	0,009	0,049	0,108
	średnia	0,00068	0,00344	0,00684	0,03903	0,09279
99,7%	min	0	0,002	0,006	0,036	0,077
	max	0,001	0,005	0,009	0,049	0,096
	średnia	0,00063	0,00314	0,00669	0,03867	0,08233
Rev	min	0	0,003	0,006	0,036	0,077
	Max	0,002	0,005	0,009	0,049	0,103
	średnia	0,00061	0,00315	0,00689	0,03867	0,08246



Rysunek 3 Wykres zależności czasu od ilości elementów tablicy Shell Sort

5. Specyfikacja komputera, na którym prowadzono pomiary

Procesor: Intel Core i7-4510U (2 GHz, 3.1 GHz Turbo, 4MB Cache)

Ram: 8GB

6. Wnioski

Wszystkie algorytmy są szybkie. Najszybsze z całej trójki jest sortowanie przez scalanie (Merge sort), na drugim miejscu pod względem szybkości jest sortowanie Shella, a na trzecim sortowanie przez kopcowanie (Heap sort). Wolniejsze działanie sortowania przez kopcowanie może wynikać z nie do końca optymalnej implementacji algorytmu.

Szybkość sortowania zależy od rozmiaru tablicy oraz stopnia posortowania początkowych elementów tablicy. Im mniejszy jest zakres tablicy oraz im większy jest stopień posortowania początkowych elementów tablicy, tym czas sortowania jest mniejszy, co jest oczekiwanym rezultatem.

W przypadku tablic posortowanych w odwrotnej kolejności najkrótsze czasy sortowania uzyskało sortowanie Shella.

7. Literatura

https://pl.wikipedia.org/wiki/Sortowanie_Shella
https://pl.wikipedia.org/wiki/Sortowanie_przez_kopcowanie
https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie
https://eduinf.waw.pl/inf/alg/003_sort/index.php
http://www.algorytm.org/algorytmy-sortowania/