Projektowanie Algorytmów i Metody sztucznej inteligencji

Projekt 1

Mateusz Broszczak, 253988

29.03.2021, poniedziałek, 15:15-16:10

Prowadząca - Mgr inż. Marta Emirsajłow

1 Wprowadzenie

Projekt polegał na zaimplementowaniu trzech wybranych algorytmów sortowania oraz przeanalizowaniu ich efektywności w poszczególnych przypadkach. Wybrane algorytmy to: sortowanie przez kopcowanie (heap sort), sortowanie szybkie (quicksort), sortowanie przez scalenie (merge sort).

2 Algorytmy

2.1 Heap sort

Z sortowanej tablicy tworzony jest kopiec binarny. W korzeniu kopca znajduje się element największy. Element ten zamieniany jest z ostatnim elementem kopca. Elementy przenoszone na koniec tworzą część posortowaną i w kolejnych krokach nie wchodzą już w skład kopca. Pozostały kopiec (bez elementów już posortowanych) jest naprawiany (przywracanie własności kopca). Procedura jest powtarzana dopóki wszystkie elementy nie zostaną posortowane. Głębokość rekurencji wynosi $O(\log n)$, następnie jest wykonywane n porównań. Złożoność obliczeniowa w przypadku średnim i najgorszym wynosi $O(n \log n)$.

2.2 Quicksort

Na początku wybierany jest tzw. element osiowy. Następnie tablica dzielona jest na dwie podtablice. Pierwsza z nich zawiera elementy mniejsze od elementu osiowego, druga elementy większe lub równe, element osiowy znajdzie się między nimi. Proces dzielenia powtarzany jest rekurencyjnie aż do uzyskania tablic jednoelementowych, które uznawane są za już posortowane. Sortowanie odbywa się podczas podziału na mniejsze podproblemy. Wybór elementu osiowego wpływa na równomierność podziału na podtablice. W każdym kroku jest wykonywane n porównań. Średnia głębokość rekurencji wynosi $O(\log n)$, a w najgorszym przypadku $O(n)$. Złożoność obliczeniowa wynosi $O(n \log n)$ dla przypadku średniego oraz $O(n^2)$ dla przypadku najgorszego.

2.3 Merge sort

Sortowana tablica dzielona jest rekurencyjnie na dwie podtablice aż do uzyskania tablic jednoelementowych. Tablice jednoelementowe uznawane są za posortowane. Następnie podtablice są porównywane, a następnie scalane w odpowiedni sposób. W wyniku tej operacji otrzymujemy tablicę posortowaną. Algorytm w każdym wywołaniu dzieli tablicę na pół, więc maksymalna głębokość rekurencji jest równa $O(\log n)$, następnie wykonywane jest porównanie i scalanie. W przypadku najgorszym zwiększa się tylko liczba porównań elementów podczas scalania. Złożoność obliczeniowa w przypadku średnim i najgorszym wynosi $O(n \log n)$.

3 Przebieg eksperymentu i otrzymane wyniki

3.1 Przebieg badań

1. Zaimplementowanie klasy z konstruktorem pozwalającym na losowe przydzielanie wartości poszczególnym elementom tablicy, przeciążenie potrzebnych operatorów.

2. Zaimplementowanie metody budującej kopiec i algorytmu sortownia przez kopcowanie, algorytmu quicksort, algorytmu sortowania przez scalenie, wstępna (wizualna) analiza poprawności działania trzech algorytmów.

3. Zaimplementowanie metody pozwalającej na sprawdzenie poprawności sortowania, metody odwracającej posortowaną tablicę w celu uzyskania tablicy posortowanej w odwrotnej kolejności, metody pozwalającej na uzyskanie częściowo posortowanej tablicy.

4. Przeprowadzenie eksperymentów dla poszczególnych rozmiarów 100 tablic, poszczególnych algorytmów sortowania i poszczególnych przypadków wcześniejszego ułożenia tablic.

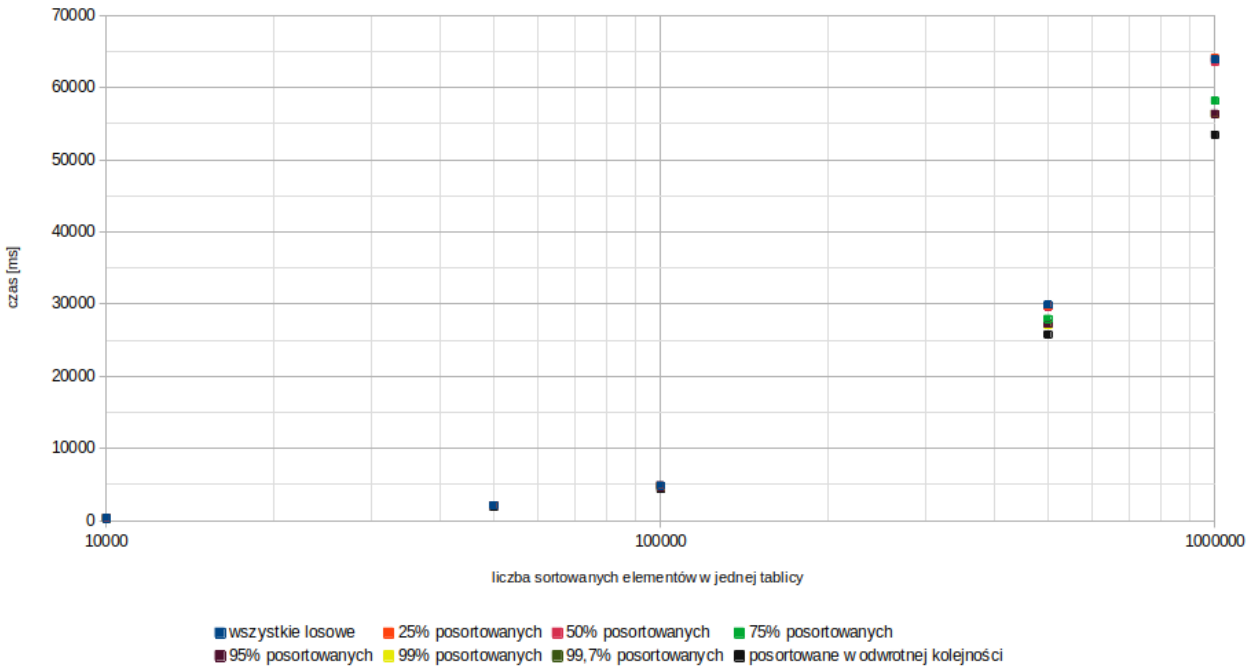
5. Przeanalizowanie złożoności i efektywności poszczególnych algorytmów.

3.2 Uzyskane wyniki badań

3.2.1 Heap sort

liczba elementów	wszystkie losowe	% początkowych posortowanych elementów						posortowane w odwrotnej kolejności
		25	50	75	95	99	99,7	
	czas $[ms]$							
10000	304	308	311	306	304	304	303	300
50000	2108	2110	2109	2093	2054	2045	2044	1904
100000	4871	4843	4821	4722	4659	4682	4737	4336
500000	29896	29810	29689	27955	27233	27107	27254	25781
1000000	63892	64061	63550	58138	56349	56292	56291	53429

Tabela 1: Porównanie czasów sortowań algorytmy heap sort dla 100 tablic n-elementowych dla poszczególnych przypadków

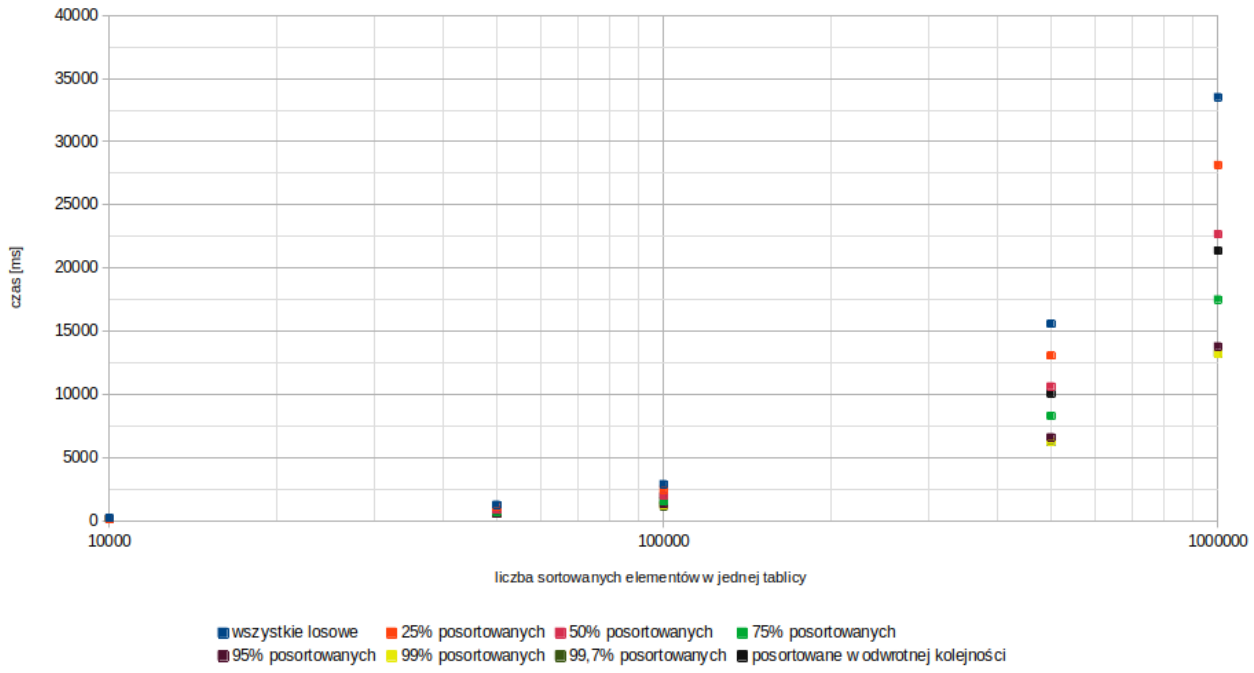


Rysunek 1

3.2.2 Quicksort

liczba elementów	wszystkie losowe	% początkowych posortowanych elementów						posortowane w odwrotnej kolejności
		25	50	75	95	99	99,7	
	czas $[ms]$							
10000	204	135	105	102	98	99	96	112
50000	1254	1058	844	693	516	512	512	823
100000	2887	2453	1951	1529	1297	1213	1152	1898
500000	15582	13053	10619	8287	6589	6271	6219	10047
1000000	33522	28152	22667	17507	13783	13193	13187	21358

Tabela 2: Porównanie czasów sortowań algorytmy quicksort dla 100 tablic n-elementowych dla poszczególnych przypadków

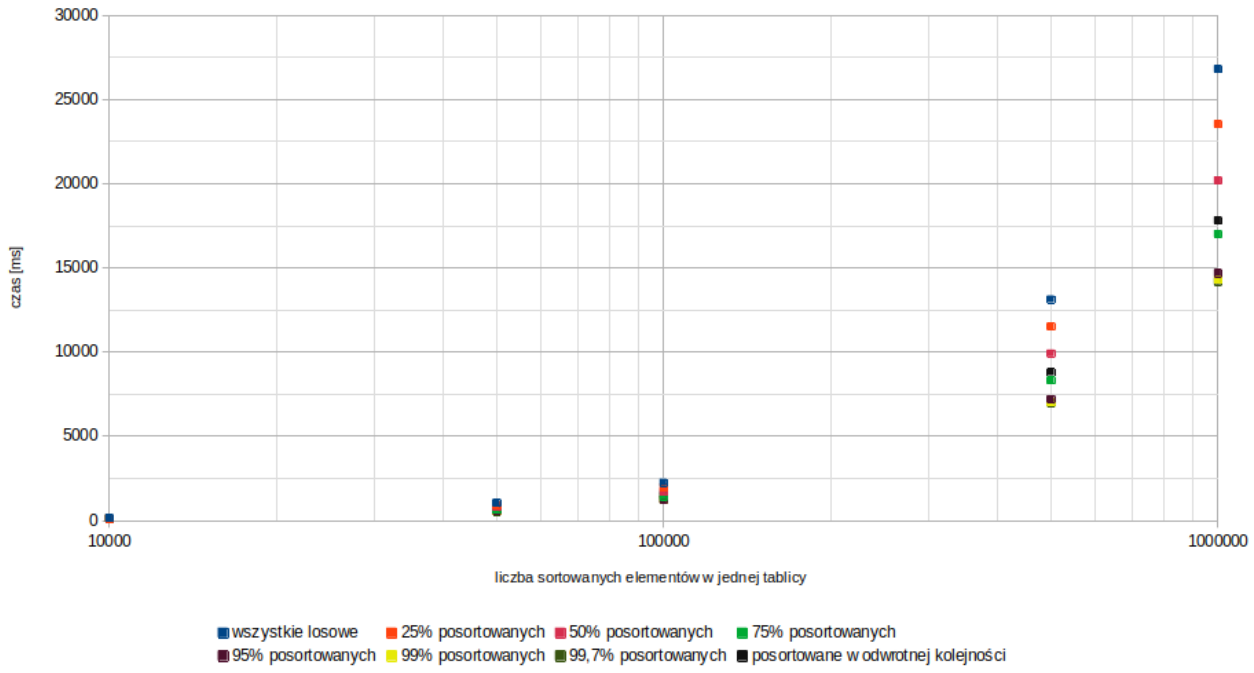


Rysunek 2

3.2.3 Merge sort

liczba elementów	wszystkie losowe	% początkowych posortowanych elementów						posortowane w odwrotnej kolejności
		25	50	75	95	99	99,7	
	czas $[ms]$							
10000	150	118	104	101	100	100	100	103
50000	1049	921	808	637	589	542	539	706
100000	2246	1955	1697	1422	1231	1220	1221	1529
500000	13130	11524	9918	8325	7208	7015	6983	8796
1000000	26795	23530	20192	17006	14688	14280	14181	17806

Tabela 3: Porównanie czasów sortowań algorytmy merge sort dla 100 tablic n-elementowych dla poszczególnych przypadków



Rysunek 3

4 Wnioski i podsumowanie

- W przypadkach posortowanej odwrotnie tablicy i wcześniejszego posortowania mniejszwego od 95% najszybszy okazał się algorytm merge sort, w pozostałych przypadkach najszybszy był algorytm quicksort,
- Algorytm heap sort był najwolniejszy z badanych algorytmów w każdym badanym przypadku,
- Wcześniejsze posortownaie danych nie wpływa znacząco na szybkość działania algorytmu heap sort, na pozostałe dwa algorytmy wpływ jest znaczący,
- Wszystkie z trzech badanych algorytmów w średnim przypadku mają złożoność $O(n \log n)$,
- W najgorszym przypadku algorytmy merge sort i heap sort również mają złożoność $O(n \log n)$,
- Nie udało się pokazać złożoności $O(n^2)$ dla algorytmu quicksort. Aby w badaniach uzyskać taką złożoność należało wybrać element osiowy na:
  - ostatni element tablicy - w przypadku z posortowaną odwrotnie tablicą,
  - pierwszy element tablicy - w przypadku z posortowaną tablicą w 99% i 99.7%.

Literatura

[1] `https://pl.wikipedia.org/wiki/Sortowanie_przez_kopcowanie`

[2] `https://pl.wikipedia.org/wiki/Sortowanie_szybkie`

[3] `https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie`

[4] `https://eduinf.waw.pl/inf/alg/003_sort/0025.php`

[5] `https://en.cppreference.com/w/cpp/numeric/random`

[6] Drozdek A. `C++ Algorytmy i struktury danych`. Helion