Data: 28.03.2021

Sprawozdanie – Projekt 1 – Algorytmy sortujące

Student: Piotr Tomczak – 252867

Kurs: Projektowanie Algorytmów i Metody Sztucznej Inteligencji

Prowadzący: Mgr inż. Marta Emirsajłow

Termin zajęć: 01.03.2021, PN, 9:15

1. Wprowadzenie

Implementujemy 3 wybrane algorytmy sortowania: quicksort, Shella, przez scalanie. Program zawiera także działające funkcje sortujące tablice metodą przez kopcowanie, ale dla tego sposobu nie została przeprowadzona analiza efektywności. Piszemy program w języku C++, który sprawdzi efektywność powyższych algorytmów. Funkcje sortujące napisane zostały na podstawie algorytmów w pseudokodzie ze źródeł podanych w bibliografii.

Testy Efektywności:

Dla 100 tablic (elementy typu całkowitoliczbowego) o nastepujacych rozmiarach: 10 000, 50 000, 100 000, 500 000 i 1 000 000 wykonujemy eksperymenty z sortowaniem w nastepujacych przypadkach:

- wszystkie elementy tablicy losowe,
- 25%, 50%, 75%, 95%, 99%, 99,7% poczatkowych elementów tablicy jest juz posortowanych,
- wszystkie elementy tablicy juz posortowane ale w odwrotnej kolejnosci.

Link do programu na GitLab: https://gitlab.com/252867/project-1-pamsi/

Data: 28.03.2021

2. Opis badanych algorytmów.

2.1 Quicksort

Algorytm sortowania szybkiego (quicksort) bazuje na strategii „dziel i zwyciężaj" dzieląc zbiór na dwie części, tak, by elementy mniejsze lub równe od wcześniej wybranego elementu – piwota znalazły się na lewo od niego, a elementy większe na prawo. Każdą z części zbioru sortujemy rekurencyjnie tym samym sposobem. Połączenie dwóch części daje zbiór uporządkowany. Przy dobrym dobraniu elementu piwota, algorytm ten jest to najszybszy z klasy obliczeniowej O(nlogn) – dla przypadku średniego. W przypadku pesymistycznym klasa złożoności obliczeniowej zostaje zdegradowana do O(n^2).

2.2 Sortowanie przez scalanie

Zbiór danych zostaje dzielony na 2 części rekurencyjnie, aż do uzyskania zbiorów jednoelementowych. Następnie sortujemy przyległe do siebie zbiory, i łączymy je w większe posortowane zbiory, aż złączymy je w jeden, ostatni, posortowany zbiór wyjściowy. Złożoność obliczeniowa średnia i pesymistyczna to O($nlogn$)..

2.3 Sortowanie metodą Shella

Algorytm powinien działać bardzo dobrze, dla zbiorów w dużym stopniu uporządkowanych. Z kolei nieefektywnie dla zbiorów nieuporząkowanych. To sortowanie polega na podziale zbioru na podzbiory, których elementy są bardzo daleko od siebie. Następnie zbiory te sortujemy „przez wstawianie". Zbiór pierwotny dzielimy teraz na kolojne podzbiory, których elementy są juz bliżej od siebie (będzie mniej podzbiorów). Znów sortujemy, ale zbiory są już częściowo uporządkowane, więc sortowanie jest szybsze. Procedurę powtarzamy aż do uzyskania jednego zbioru końcowego, który będzie posortowany. Odstępy między elementami wybieramy metodą D. Knutha. Złożoność obliczeniowa średnia i pesymistyczna to O(nlogn).

3. Przebieg eksperymentów i wyniki

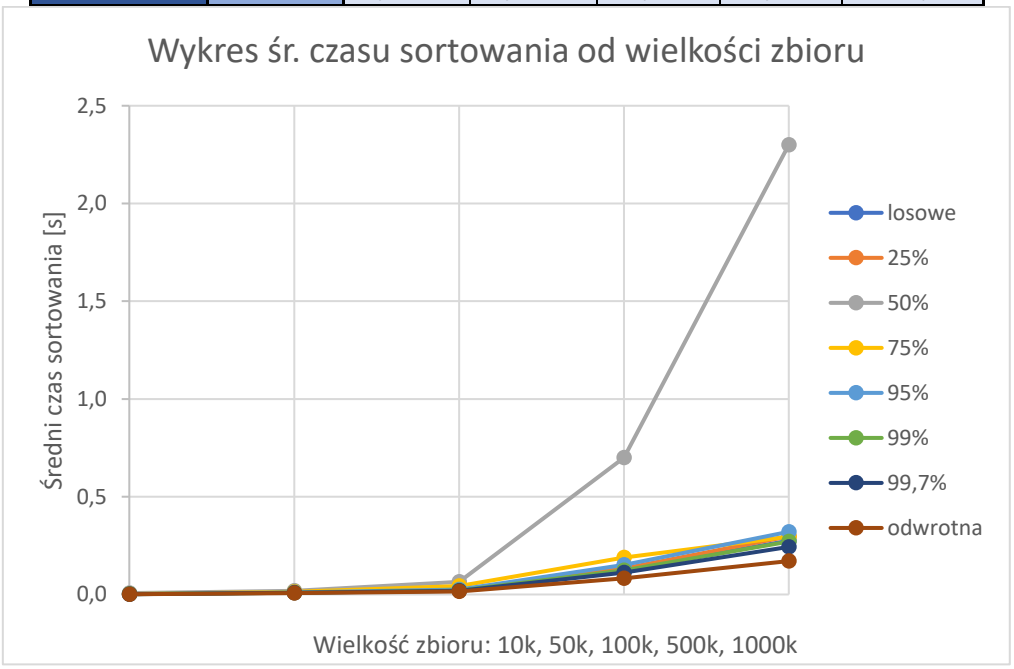
Dla każdego sposobu sortowania, program na początku tworzy tablicę danego rozmiaru o elementach losowych i sortuje (algorytmem quicksort) pewną jej część, zadaną w instrukcji. Następnie sortujemy częściowo posortowaną tablicę wybranym sposobem sortowania. Prawidłowość posortowania tablicy jest za każdym razem sprawdzana odpowiednią funkcją. Mierzymy czas sortowania poprzez zliczanie cykli procesora i przeliczanie ich na sekundy.

Data: 28.03.2021

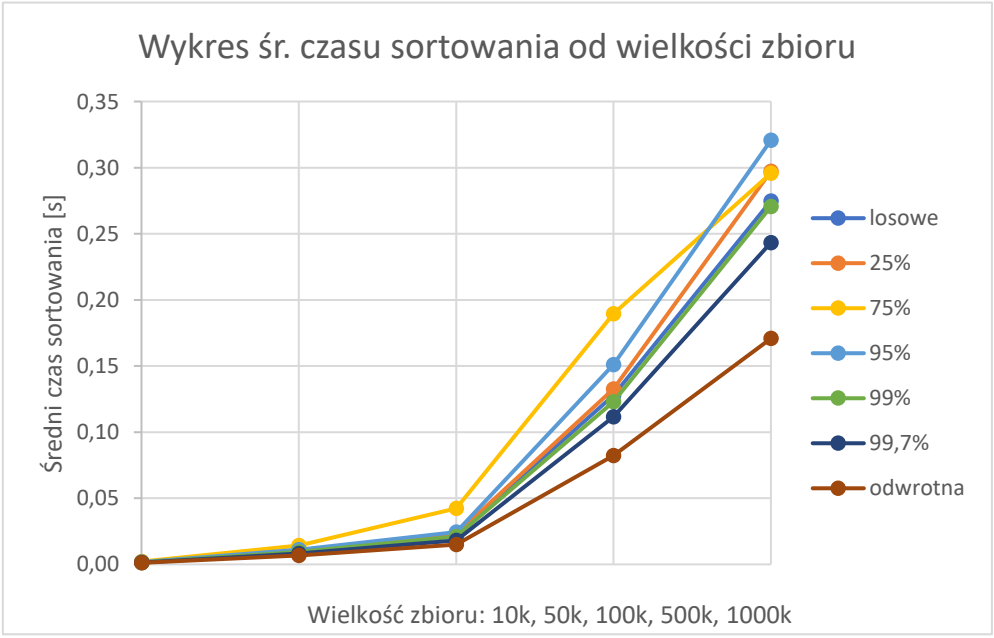
3.1 Quicksort

Tabela 1 Średnie czasy sortowania dla quicksort [s]

		Rozmiar tablicy				
		10 tyś.	50 tyś.	100 tyś.	500 tyś.	1000 tyś.
Część posortowana [%]	0	0,00177	0,0102	0,0220	0,128	0,275
	25	0,00179	0,0105	0,0230	0,133	0,297
	50	0,00710	0,0183	0,0643	0,699	2,30
	75	0,00214	0,0143	0,0424	0,190	0,296
	95	0,00176	0,0111	0,0245	0,151	0,321
	99	0,00150	0,00931	0,0209	0,123	0,271
	99,7	0,00133	0,00829	0,0182	0,112	0,243
	odwrotna	0,00119	0,00686	0,0151	0,0822	0,171



Wykres 1 Dla quicksort



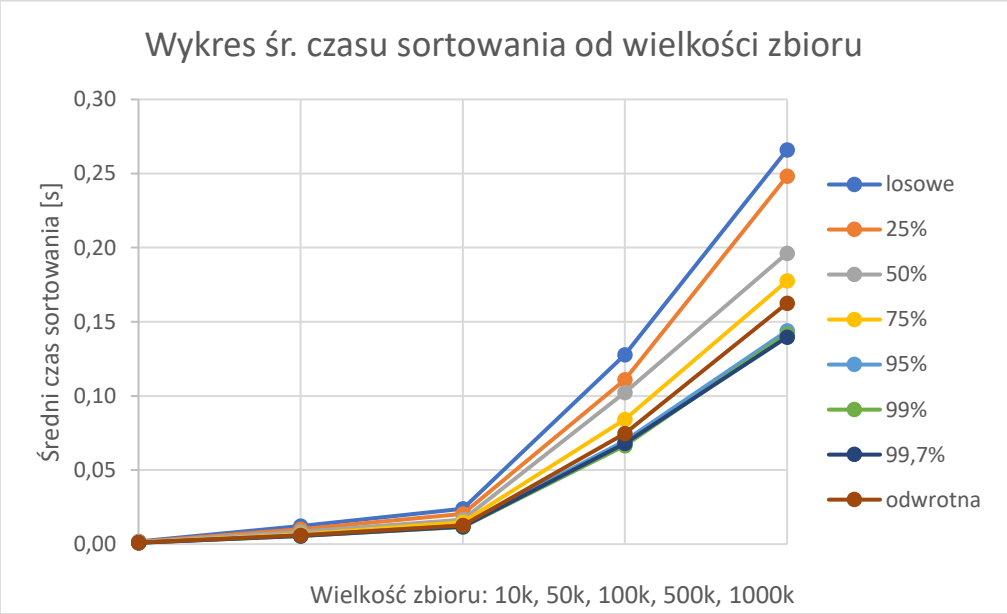
Wykres 2 Dla quicksort, bez pesymistycznego przypadku

Data: 28.03.2021

3.2 Sortowanie przez scalanie

Tabela 2 Średnie czasy sortowania dla algorytmu scalania [s]

		Rozmiar tablicy				
Część posortowana [%]		10 tys.	50 tys.	100 tys.	500 tys.	1000 tys.
	0	0,00183	0,0123	0,0239	0,128	0,266
	25	0,00156	0,0100	0,0204	0,111	0,248
	50	0,00136	0,00837	0,0165	0,102	0,196
	75	0,00113	0,00653	0,0148	0,0841	0,178
	95	0,00100	0,00597	0,0124	0,0696	0,144
	99	0,00093	0,00545	0,0117	0,0665	0,142
	99,7	0,00093	0,00543	0,0117	0,0680	0,140
	odwrotna	0,00106	0,00602	0,0126	0,0747	0,163



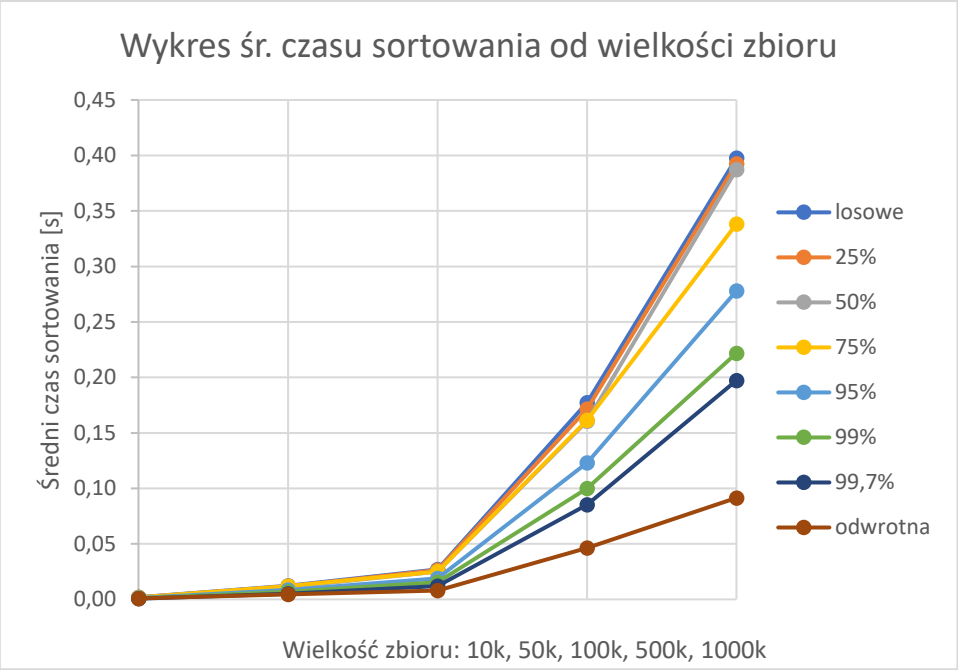
Wykres 3 Dla algorytmu scalania

3.3 Sortowanie metodą Shella

Tabela 3 Średnie czasy sortowania dla algorytmu Shella [s]

		Rozmiar tablicy				
Część posortowana [%]		10 tyś.	50 tyś.	100 tyś.	500 tyś.	1000 tyś.
	0	0,00188	0,0122	0,0271	0,177	0,398
	25	0,00177	0,0117	0,0265	0,172	0,393
	50	0,00173	0,0112	0,0249	0,161	0,387
	75	0,00169	0,0119	0,0250	0,161	0,338
	95	0,00127	0,00857	0,0190	0,123	0,278
	99	0,000950	0,00677	0,0154	0,100	0,222
	99,7	0,000642	0,00500	0,0120	0,085	0,197
	odwrotna	0,000865	0,00454	0,00808	0,0462	0,0914

Data: 28.03.2021



Wykres 4 Dla algorytmu Shella

4. Omówienie i wnioski

Na podstawie wyników, stwierdzamy, że:

1. Algorytm quicksort w przypadku, gdy połowa zbioru została już posortowana działa najwolniej, ze złożonością obliczeniową O(n^2).
2. Algorytm sortowania przez scalanie jest szybszy od algorytmu quicksort dla dużych i małych zbiorów.
3. Algorytm sortowania Shella może być najszybszy, lub najwolniejszy. Im bardzie uporządkowany zbiór, tym szybciej jest on sortowany względem innych algorytmów.
4. Wyniki eksperymentów potwierdziły wszystkie cechy badanych algorytmów sortowania, oprócz tego, że quicksort jest najszybszy w klasie obliczeniowej O(nlogn). Różnica może wynikać z nieoptymalnego napisania funkcji sortującej metodą quicksort.

5. Literatura.

1. Projekt1.pdf
2. http://pl.wikipedia.org/wiki/Sortowanie_przez_kopcowanie
3. http://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie
4. http://pl.wikipedia.org/wiki/Sortowanie_Shella
5. http://pl.wikipedia.org/wiki/Sortowanie_szybkie
6. https://eduinf.waw.pl/inf/alg/003_sort/0012.php
7. https://eduinf.waw.pl/inf/alg/003_sort/0013.php
8. https://eduinf.waw.pl/inf/alg/003_sort/0014.php
9. https://eduinf.waw.pl/inf/alg/003_sort/0015.php
10. https://eduinf.waw.pl/inf/alg/003_sort/0016.php
11. https://eduinf.waw.pl/inf/alg/003_sort/0017.php
12. https://eduinf.waw.pl/inf/alg/003_sort/0018.php