Projektowanie Algorytmów i Metody Sztucznej Inteligencji

Projekt 1

Algorytmy sortujące

Data oddania sprawozdania	03.04.2019
Imię i nazwisko	Kinga Tokarska 241621
Nr indeksu	241621
Termin zajęć	Środa, 11.15-13.00
Prowadzący kurs	Dr inż. Łukasz Jeleń
Kod kursu	E02-47f

1. Wstęp teoretyczny

Sortowanie danych jest jednym z podstawowych, najczęściej rozwiązywanych problemów informatyki. Polega on na uporządkowaniu zbioru danych pewnego kryterium. Jest ono także doskonałym przykładem problemu, który można rozwiązać na wiele sposobów, różniących się od siebie efektywnością.

Miarą służącą do porównywania efektywności algorytmów, a jednocześnie jednym z najważniejszych parametrów charakteryzujących algorytm jest złożoność obliczeniowa, czyli ilość zasobów potrzebnych do rozwiązania problemów obliczeniowych. Złożoność obliczeniowa badana jest w dwóch podstawowych aspektach: czasu (złożoność czasowa) i pamięci (złożoność pamięciowa).

Rozróżniamy kilka rodzajów złożoności: pesymistyczna (określa ilość zasobów potrzebnych do wykonania algorytmu przy założeniu wystąpienia złośliwych lub najgorszych danych), typowa (określa ilość zasobów potrzebnych do wykonania algorytmu przy założeniu wystąpienia typowych lub oczekiwanych danych) oraz optymistyczna (określa ilość zasobów potrzebnych do wykonania algorytmu przy założeniu wystąpienia najlepszych danych).

2. Opis badanych algorytmów

2.1. Sortowanie przez scalanie

Sortowanie przez scalanie jest algorytmem typu "dziel i zwyciężaj". Stanowi przykład algorytmu rekurencyjnego. Ideą działania sortowania przez scalanie jest dzielenie zbioru danych na mniejsze zbiory.

Algorytm sortowania przez scalanie polega na dzieleniu porządkowanego zbioru danych na kolejne połowy, dopóki taki podział jest możliwy, to znaczy do momentu uzyskania tablic jednoelementowych. Proces dzielenia następuje bez sprawdzania warunków. Tablice jednoelementowe uznajemy za posortowane. Kolejnym procesem jest połączenie uzyskanych uporządkowanych zbiorów w jeden, również uporządkowany zbiór. Operacja ta realizowana jest z wykorzystaniem zbioru pomocniczego, w którym tymczasowo przechowywane będą scalane elementy dwóch zbiorów. Zawartość zbioru głównego zostaje skopiowana do struktury pomocniczej, a wskaźniki ustawione na początkach kolejnych zbiorów. Następnie należy porównywać ze sobą pierwsze elementy każdego ze scalanych zbiorów, za każdym razem umieszczając mniejszą wartość w zbiorze tymczasowym i przesuwając wskaźnik o jeden. Czynność powtarza się aż do momentu osiągnięcia końca jednego ze zbiorów. Należy uwzględnić fakt, że wielkość zbiorów może być różna. Jeżeli koniec osiągnął zbiór pierwszy, elementy w zbiorze głównym są posortowane prawidłowo. W przypadku gdy zbiór drugi wcześniej osiągnie koniec, pozostałą zawartość zbioru pierwszego należy umieścić w zbiorze tymczasowym. Ostatni krok stanowi przepisanie zawartości zbioru tymczasowego do zbioru wynikowego.

Tablica wejściowa dzielona jest na pół przy każdym wywołaniu rekurencji, więc wysokość drzewa wywołań funkcji dla tablicy o rozmiarze n wynosi $\log_2 n$. Złożoność operacji scalania tablic jest liniowa, każde scalanie to koszt $O(n)$. Wynikowa złożoność czasowa algorytmu wynosi więc, zarówno w przypadku optymistycznym, typowym, jak i pesymistycznym, $O(n \log_2 n)$.

2.2. Sortowanie Shella

Sortowanie Shella to algorytm, będący uogólnieniem sortowania przez wstawianie. Opiera się ono na podziale sortowanego zbioru na podzbiory, których elementy są od siebie odległe o pewien ustalony odstęp. Często nazywane jest sortowaniem przez wstawianie z malejącym odstępem.

Pierwszym krokiem jest wybór pewnej przerwy między elementami, które mają zostać posortowane. Każdy z otrzymanych podzbiorów należy posortować algorytmem sortowania przez wstawianie, a następnie zmniejszyć wielkość początkowo wybranej przerwy między elementami

według pewnej reguły. W ten sposób powstają nowe podzbiory, z których każdy należy posortować przez wstawianie, a następnie ponownie zmniejszyć odstęp pomiędzy elementami. Operację sortowania powtarza się, za każdym razem zmniejszając odstęp, aż do momentu kiedy przerwa będzie miała wartość równą jeden. Odstęp taki nie dzieli zbioru wejściowego na podzbiory. Wówczas należy posortować przez wstawianie cały pozostały zbiór. Operacja ta nie jest skomplikowana, gdyż dzięki poprzednim obiegom sortującym zbiór został częściowo uporządkowany. Elementy o małych wartościach zbliżyły się do początku zbioru, z kolei te o większych wartościach do jego końca.

Efektywność algorytmu sortowania Shella w dużym stopniu zależna jest od ciągu przyjętych odstępów. Istnieje wiele sposobów wyboru kolejnych wyrazów z sortowanej tablicy. Każdy z nich charakteryzuje się inną złożonością obliczeniową. Pierwotnie Shell zaproponował pierwszy odstęp równy połowie liczby elementów w sortowanym zbiorze. Kolejne odstępy otrzymywał, dzieląc odstęp przez dwa. Złożoność obliczeniowa optymistyczna wynosi $O(n^{1,14})$, a złożoność obliczeniowa typowa i pesymistyczna $O(n^{1,15})$.

2.3. Quicksort

Sortowanie szybkie również jest przykładem algorytmu rekurencyjnego. Podobnie jak omawiane wcześniej sortowanie przez scalanie wykorzystuje technikę „dziel i zwyciężaj”.

W każdym kroku sortowania szybkiego wybrany zostaje jeden element w sortowanej tablicy, określany jako piwot. Może nim być zarówno pierwszy, środkowy, ostatni, mediana, jak i dowolny element wybrany według innego dostosowanego do zbioru danych schematu. Następnie algorytm porównuje z wybranym elementem wszystkie pozostałe elementy tablicy. Na początku zbioru umieszczone zostają dwa wskaźniki. Pierwszy z nich przebiega przez zbiór, wyszukując wartości mniejsze od piwotu. Po odnalezieniu takich elementów są one wymieniane z elementami na pozycji wskazywanej przez drugi wskaźnik. Po tej operacji drugi wskaźnik jest przesuwany na następną pozycję. Wskaźnik ten zapamiętuje pozycję, na którą trafi następny element oraz na końcu wskazuje miejsce, gdzie znajdzie się piwot. W ten sposób powstają dwie, niekoniecznie tych samych rozmiarów, partycje. Elementy nie większe od piwotu ustawione są po jego lewej stronie (w lewej partycji), natomiast nie mniejsze po prawej (w prawej partycji). Porządek elementów w partycji jest nieustalony. Położenie elementów o wartości równej wartości piwotu nie wpływa na proces sortowania, zatem mogą one występować w obu partycjach. Element wybrany do podziału znajduje się już na swojej prawidłowej pozycji i nie bierze udziału w dalszym sortowaniu. Kroki te są powtarzane aż do uzyskania posortowanej tablicy.

Złożoność obliczeniowa zależna jest od wyboru elementu rozdzielającego. Operacja przenoszenia elementów odbywa się w czasie liniowym. O złożoności obliczeniowej algorytmu decyduje więc to, ile nastąpi rekurencyjnych wywołań funkcji. W przypadku optymistycznym tablica będzie dzielona na pół. Wówczas głębokość drzewa wywołań zależy logarytmicznie od liczby danych wejściowych, a złożoność czasowa algorytmu wynosi $O(n \log_2 n)$. Przypadek pesymistyczny występuje, gdy piwot za każdym razem znajduje się na brzegu tablicy. Wówczas przy każdym kolejnym wywołaniu liczba elementów do posortowania jest tylko o 1 mniejsza, a funkcja zostanie wywołana n razy. Pesymistyczna złożoność czasowa wynosi więc $O(n^2)$.

3. Przebieg ćwiczenia

Ćwiczenie polegało na zaimplementowaniu trzech algorytmów sortujących: sortowanie przez scalanie, sortowanie Shella oraz sortowanie quicksort. W celu przeprowadzenia analizy efektywności i porównania ze sobą powyższych algorytmów przeprowadzono pomiary czasu sortowania stu tablic składających się z elementów typu całkowitoliczbowego o rozmiarach: $10\,000$, $50\,000$, $100\,000$, $500\,000$, $1\,000\,000$ o różnych stopniach wstępnego posortowania: 0%, 25%, 50%, 75%, 95%, 99%, $99{,}7\%$ oraz tablicy posortowanej odwrotnie.

Przeprowadzona została także wstępna weryfikacja poprawności sortowania zaimplementowanych algorytmów. Dla tablic o małych rozmiarach dokonano weryfikacji wizualnej. W celu sprawdzenia tablic o większych rozmiarach wykorzystano pomocniczą procedurę sprawdzającą poprawność uporządkowania elementów w tablicy.

Eksperymenty przeprowadzano, korzystając z komputera wyposażonego w procesor Intel Core i5-7200U (2.50 GHz, 2712 MHz, 2 rdzenie) i 8 GB pamięci RAM SO-DIMM DDR4.

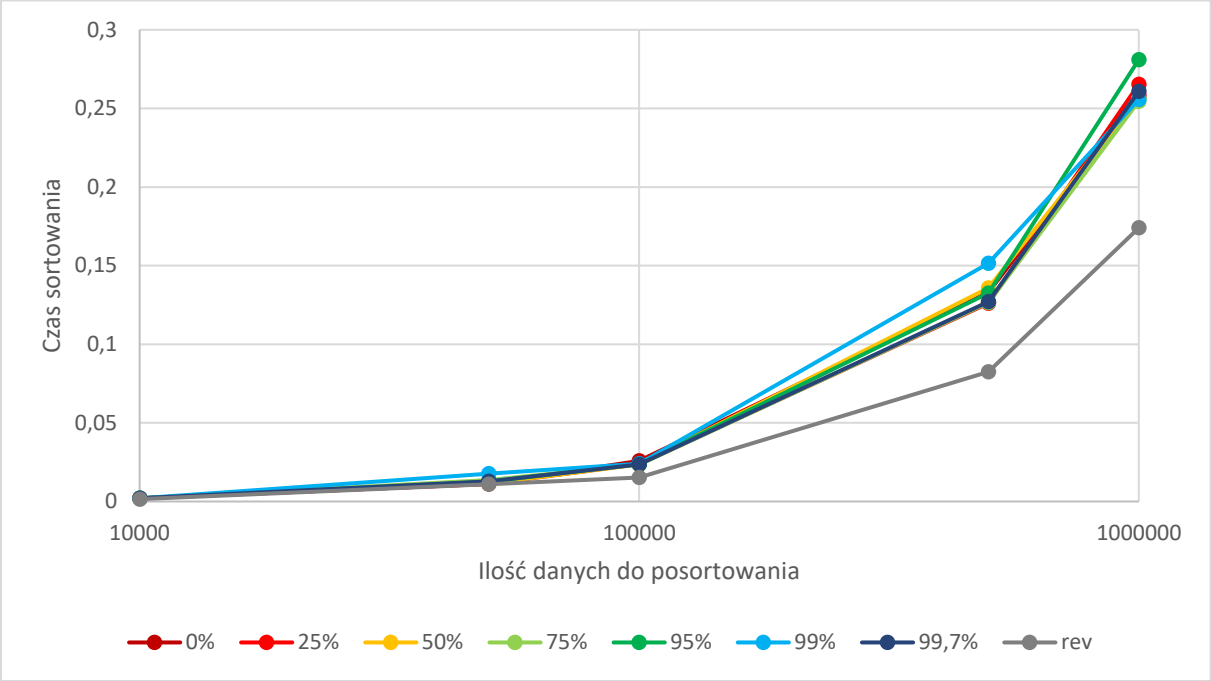
Wyniki przeprowadzanych eksperymentów (minimalne, średnie oraz maksymalne wartości czasów sortowania o różnych rozmiarach i różnych stopniach wstępnego posortowania) przedstawiono w poniższych tabelach oraz na poniższych wykresach.

4. Wyniki pomiarów

4.1. Sortowanie przez scalanie

		10 000	50 000	100 000	500 000	1 000 000
Wszystkie elementy losowe	min	0,00100	0,01000	0,02200	0,11900	0,25000
	śr	0,00202	0,01119	0,02580	0,13291	0,25808
	max	0,00300	0,01600	0,04000	0,26400	0,33600
Posortowane początkowe 25% elementów	min	0,00100	0,01000	0,02200	0,11900	0,24900
	śr	0,00204	0,01123	0,02372	0,12593	0,26537
	max	0,003000	0,01400	0,03400	0,14800	0,48200
Posortowane początkowe 50% elementów	min	0,001000	0,01000	0,02200	0,11900	0,24800
	śr	0,00206	0,01138	0,02367	0,13571	0,25641
	max	0,004000	0,01900	0,03200	0,45700	0,30500
Posortowane początkowe 75% elementów	min	0,00100	0,01000	0,02200	0,11800	0,24700
	śr	0,00195	0,01372	0,02330	0,12647	0,25458
	max	0,004000	0,11500	0,03500	0,16100	0,29500
Posortowane początkowe 95% elementów	min	0,00100	0,01000	0,02200	0,11900	0,24700
	śr	0,00201	0,01269	0,02375	0,13244	0,28093
	max	0,00300	0,03900	0,03400	0,42900	0,57000
Posortowane początkowe 99% elementów	min	0,00100	0,01000	0,02100	0,11900	0,24900
	śr	0,00201	0,01763	0,02396	0,15147	0,25564
	max	0,00300	0,07400	0,03300	0,31100	0,28700
Posortowane początkowe 99,7% elementów	min	0,00100	0,01000	0,02100	0,11900	0,24800
	śr	0,00207	0,01289	0,02361	0,12700	0,26077
	max	0,00400	0,06000	0,03400	0,15400	0,34600
Posortowane w odwrotnej kolejności	min	0,00100	0,00600	0,01400	0,07600	0,16000
	śr	0,00151	0,01097	0,01527	0,08234	0,17399
	max	0,00300	0,02400	0,02600	0,11500	0,32000

Tab.1. Wyniki pomiarów dla sortowania przez scalanie

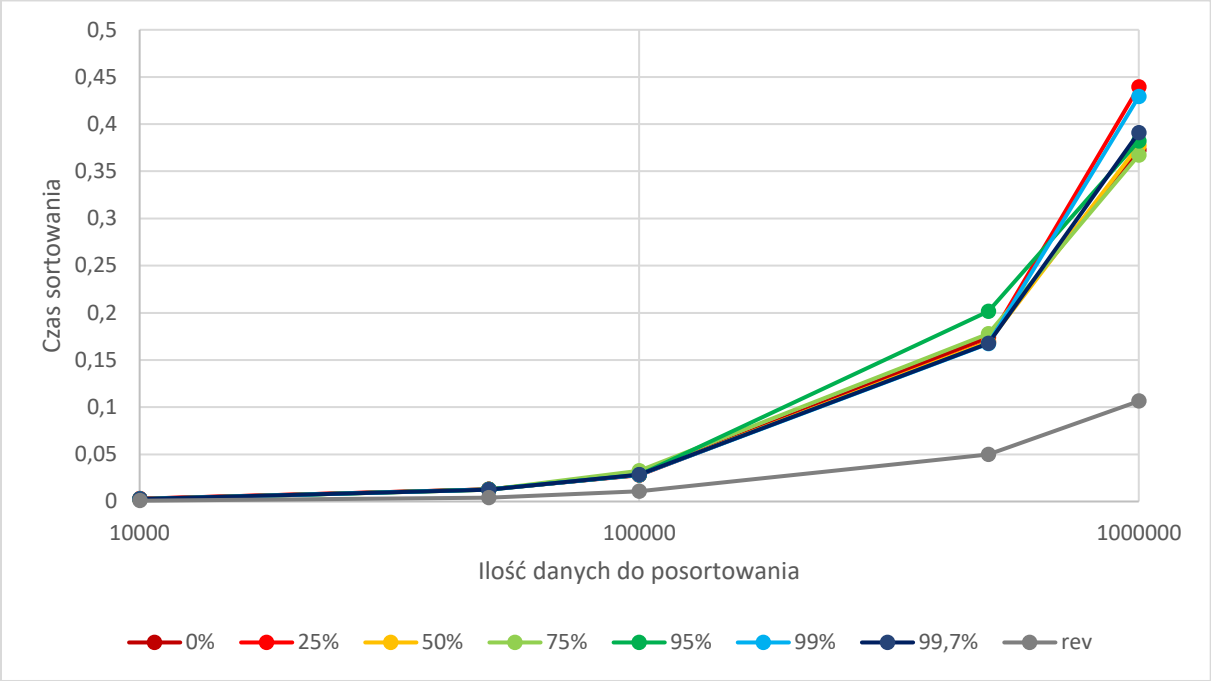


Rys. 1. Wykres zależności czasu sortowania od ilości danych dla sortowania przez scalanie

4.2. Sortowanie Shella

		10 000	50 000	100 000	500 000	1 000 000
Wszystkie elementy losowe	min	0,00100	0,01200	0,02600	0,16200	0,35500
	śr	0,00299	0,01299	0,02800	0,17397	0,37259
	max	0,01200	0,02100	0,03600	0,20900	0,43400
Posortowane początkowe 25% elementów	min	0,00200	0,01200	0,02600	0,15800	0,35500
	śr	0,00294	0,01273	0,02784	0,16959	0,43930
	max	0,00800	0,01700	0,03600	0,20600	1,36000
Posortowane początkowe 50% elementów	min	0,00200	0,01200	0,02600	0,15700	0,34600
	śr	0,00250	0,01270	0,02832	0,16881	0,37732
	max	0,00400	0,01600	0,03900	0,19500	0,70500
Posortowane początkowe 75% elementów	min	0,00200	0,01100	0,02600	0,15700	0,35000
	śr	0,00256	0,01264	0,03244	0,17781	0,36706
	max	0,00500	0,01700	0,17300	0,35800	0,39000
Posortowane początkowe 95% elementów	min	0,00100	0,01100	0,02600	0,15600	0,35000
	śr	0,00226	0,01285	0,02805	0,20147	0,38214
	max	0,00400	0,01700	0,03500	0,44300	0,83400
Posortowane początkowe 99% elementów	min	0,00100	0,01100	0,02600	0,16000	0,35400
	śr	0,00256	0,01263	0,02816	0,16733	0,42914
	max	0,00700	0,01600	0,05100	0,19900	0,94000
Posortowane początkowe 99,7% elementów	min	0,00100	0,01100	0,02600	0,15800	0,35100
	śr	0,00267	0,01240	0,02845	0,16745	0,39099
	max	0,00700	0,01400	0,03600	0,20000	0,72400
Posortowane w odwrotnej kolejności	min	0,00100	0,00300	0,00800	0,04600	0,09800
	śr	0,00085	0,00421	0,01095	0,04981	0,10636
	max	0,00200	0,00600	0,01500	0,06300	0,13500

Tab.2. Wyniki pomiarów dla sortowania metodą Shella

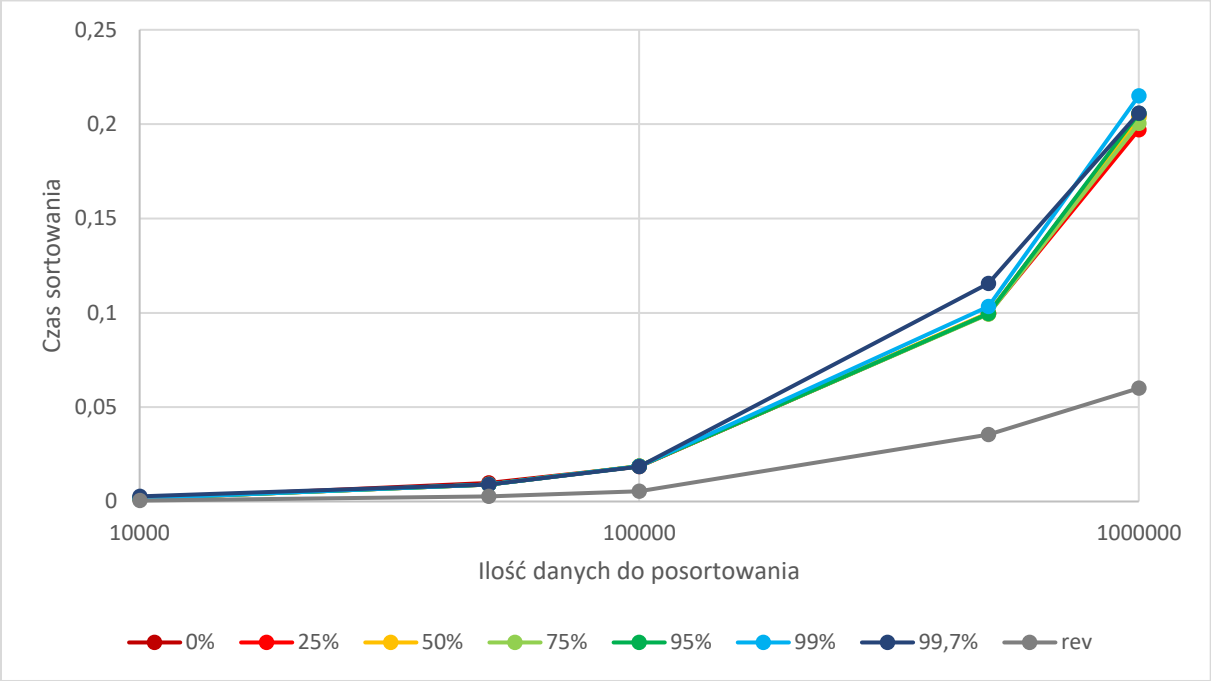


Rys. 2. Wykres zależności czasu sortowania od ilości danych dla sortowania metodą Shella

4.3. Sortowanie Quicksort

		10 000	50 000	100 000	500 000	1 000 000
Wszystkie elementy losowe	min	0,00100	0,00800	0,01800	0,09500	0,19300
	śr	0,00159	0,00982	0,01854	0,09952	0,20474
	max	0,00200	0,04300	0,02200	0,18800	0,29600
Posortowane początkowe 25% elementów	min	0,00100	0,00800	0,01800	0,09400	0,19200
	śr	0,00161	0,00903	0,01860	0,09951	0,19704
	max	0,00300	0,01000	0,02200	0,16600	0,24000
Posortowane początkowe 50% elementów	min	0,00100	0,00800	0,01800	0,09400	0,19100
	śr	0,00156	0,00899	0,01877	0,09999	0,20337
	max	0,00200	0,01000	0,02100	0,22400	0,32000
Posortowane początkowe 75% elementów	min	0,00100	0,00800	0,01800	0,09400	0,19200
	śr	0,00158	0,00897	0,01861	0,09924	0,20032
	max	0,00200	0,01000	0,02200	0,19900	0,30000
Posortowane początkowe 95% elementów	min	0,00100	0,00800	0,01800	0,09400	0,19300
	śr	0,00164	0,00896	0,01860	0,09958	0,20561
	max	0,00300	0,01000	0,02300	0,16300	0,33300
Posortowane początkowe 99% elementów	min	0,00100	0,00800	0,01800	0,09400	0,19000
	śr	0,00163	0,00892	0,01869	0,10325	0,21499
	max	0,00300	0,01000	0,02200	0,17600	0,32200
Posortowane początkowe 99,7% elementów	min	0,00100	0,00800	0,01800	0,09400	0,19000
	śr	0,00272	0,00896	0,01849	0,11549	0,20586
	max	0,03900	0,01000	0,02200	0,23100	0,44700
Posortowane w odwrotnej kolejności	min	0,00100	0,00200	0,00500	0,02800	0,05800
	śr	0,00050	0,00270	0,00543	0,03551	0,06004
	max	0,00100	0,00300	0,00700	0,07100	0,06300

Tab.3. Wyniki pomiarów dla sortowania metodą quicksort



Rys. 3. Wykres zależności czasu sortowania od ilości danych dla sortowania metodą quicksort

5. Wnioski

Dla sortowania przez scalanie wszystkie otrzymane czasy (Tab. 1) są proporcjonalne do iloczynu $nlog_2n$, co oznacza, że złożoność obliczeniowa algorytmu wynosi $O(nlog_2n)$. Najdłużej trwa sortowanie zbioru elementów losowych. Należy jednak zwrócić uwagę na fakt, że różnice w otrzymanych czasach są niewielkie, algorytm nie jest specjalnie czuły na rozkład danych wejściowych (Rys. 1). Przypadek optymistyczny i pesymistyczny nie występują.

Dla sortowania metodą Shella (Tab. 2) czas sortowania zbioru posortowanego odwrotnie jest znacznie krótszy od czasu sortowania zbioru losowego. Przypadkiem pesymistycznym jest więc tablica z elementami losowymi. Czym większy jest stopień posortowania tablicy początkowej, tym czas wykonania algorytmu jest krótszy, ale nawet przy posortowanych 99,7% tablicy, czas sortowania nie jest krótszy niż przy sortowaniu tablicy uporządkowanej odwrotnie (Rys. 2).

Dla sortowania szybkiego wszystkie otrzymane czasy (Tab. 3) są proporcjonalne do iloczynu $nlog(n)$, co oznacza, że złożoność obliczeniowa algorytmu wynosi $O(nlog_2n)$. Czasy sortowania dla poszczególnych przypadków tablic (tablic losowych, częściowo uporządkowanych i posortowanych odwrotnie) są tego samego rzędu, nie udaje się wyodrębnić przypadku optymistycznego i pesymistycznego (Rys. 3).

Porównanie czasów działania badanych algorytmów pozwala stwierdzić, że najszybszą metodą sortowania jest algorytm sortowania szybkiego. Zarówno przy wykorzystaniu tablic losowych, jak i częściowo uporządkowanych, okazuje się on znacznie szybszy od sortowania przez scalanie i sortowania metodą Shella. Ze względu na swoją rekurencyjność najlepiej sprawdza się w przypadku sortowania większych ilości danych.

6. Literatura

https://www.tutorialspoint.com/data_structures_algorithms/shell_sort_algorithm.htm
https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie
https://pl.wikipedia.org/wiki/Sortowanie_szybkie
https://pl.wikipedia.org/wiki/Sortowanie_Shella