SPRAWOZDANIE

ALGORYTMY SORTOWANIA

ALGORYTMY I STRUKTURY DANYCH

KAMILA GAŁKA

WSTĘP

Sortowaniem nazywamy proces, w którym porządkujemy zbiór względem pewnych cech charakterystycznych jego elementów. W celu wykonania tego zadania stosuje się różne algorytmy, które różnią się m.in. stabilnością oraz złożonością obliczeniową.
Do przeprowadzenia badań zdecydowałam się na wykorzystanie algorytmów Quicksort, Shellsort oraz Combsort.

Quicksort

Algorytm ten jest oparty na zasadzie „divide and conquer" (podział problemu na mniejsze, możliwie najprostsze podproblemy). Quicksort wybiera z sortowanej kolekcji pewien element (*pivot*) i ustawia elementy kolekcji tak, by elementy „mniejsze" od pivota znalazły się w tej kolekcji przed nim, a elementy „większe" – po nim. W ten sposób pivot trafia na swoją docelową pozycję, a na elementach „mniejszych" i „większych" rekursywanie powtarza się całe działanie.
Istotnym punktem tego algorytmu jest odpowiedni wybór pivota. W przypadku, gdy pivot okaże się zawsze być „najmniejszym" lub „największym" elementem danej kolekcji otrzymamy złożoność czasową $O(n^2)$. W przypadku optymistycznym, gdy za każdym razem wybierzemy medianę sortowanej kolekcji możemy się spodziewać złożoności **liniowo logarytmicznej.** W zaimplementowanym przeze mnie algorytmie za pivot wybieram element, który znajduje się w pobliżu środka sortowanej kolekcji. Dzięki temu unika się pesymistycznego przypadku dla ciągu posortowanego. Nie wyklucza to jednak zupełnie szansy na wybór niekorzystnego pivota.
Quicksort jest algorytmem niestabilnym (czyli może nie zachować kolejności „równych" elementów).

Shellsort

Sortowanie Shella jest oparte na porównywaniu ze sobą elementów położonych daleko od siebie i stopniowym zmniejszaniu odstępu między porównywanymi elementami. W ten sposób początkowo otrzymujemy wiele, krótkich ciągów, których elementy można w łatwy sposób porównać i w razie potrzeby zamienić miejscami. Po każdym skróceniu dystansu otrzymujemy coraz mniej, coraz dłuższych ciągów, które należy posortować, a elementy kolekcji znajdują się coraz bliżej swoich docelowych pozycji. Duże znaczenie w przypadku tego algorytmu ma odpowiednie dobranie odstępów. Istnieje wiele propozycji jak powinien wyglądać taki ciąg. Zgodnie z poleceniem, podczas badań sprawdziłam działanie algorytmu dla dwóch różnych ciągów odstępów. Pierwszy z nich, zaproponowany przez Roberta Sedgewicka w najgorszym przypadku wygeneruje złożoność $\boldsymbol{O(n^{4/3})}$, drugi, zaproponowany przez Donalda Knutha - $\boldsymbol{O(n^{3/2})}$**.** Maksymalna długość odstępu nie przekracza ilości elementów w sortowanej kolekcji.
Podobnie jak Quicksort, Shellsort nie jest algorytmem stabilnym.

Combsort

Sortowanie grzebieniowe jest nieco podobne do sortowania Shella. Podobnie jak tam, również tutaj porównuje się elementy odległe od siebie o konkretny dystans. W przypadku Combsort pierwszą taką odległością jest długość sortowanej kolekcji podzielona przez współczynnik 1.3. Po wykonaniu porównań i ewentualnych przestawień elementów odległość ponownie dzieli się przez 1.3 i powtarza cały proces, dopóki odległość nie spadnie do 1 (należy zapewnić, że odległość nie będzie równa 0). Gdy odległość będzie już równa 0 algorytm zaczyna się zachowywać tak jak bubble-sort. Wprowadza się więc zmienną, która kontroluje, czy podczas ostatniego przejścia po kolekcji doszło do jakiegokolwiek zamienienia pozycji elementów, a jeśli nie, przerywa działanie pętli. Złożoność w najlepszym przypadku wynosi *O(nlogn)*, natomiast w praktyce potwierdzono, że jest ona gorsza niż w przypadku quicksorta.
Podobnie jak dwa opisane powyżej algorytmy, również Combsort nie jest algorytmem stabilnym.

SCHEMAT I PRZEBIEG BADAŃ

Do przeprowadzenia badań należało wygenerować po sto sekwencji o różnych kombinacjach ilości elementów (100 tys., 500 tys., 1mln, 2mln) oraz typu posortowania (w pełni losowy, posortowany w połowie, posortowany w pełni, posortowany odwrotnie). W tym celu skorzystałam z klasy SequenceGenerator *(w przesłanym programie znajduje się od w katalogu o tej samej nazwie)*.

Dla każdej ilości elementów oraz dla każdego typu posortowania powstały sekwencje zawierające liczby całkowite z przedziału od -100000 do 100000.
Elementy ciągów losowych powstały przy pomocy obiektu klasy Random.
Przy generowaniu sekwencji w pełni posortowanej wykorzystano metodę generującą ciąg losowy, po czym posortowano ten ciąg przy użyciu zaimplementowanego algorytmu Quicksort.
Sekwencje częściowo posortowane powstały poprzez scalenie dwóch sekwencji o długości $\frac{n}{2}$. Pierwsza sekwencja została wygenerowana jako ciąg w pełni posortowany, a jego elementy mieszczą się w przedziale [-100000;0). Druga sekwecja została wyganerowana jako ciąg losowy, którego elementy pochodzą z przedziału [0;100000].
Ciąg posortowany odrotnie powstał poprzez wygenerowanie ciągu w pełni posortowanego i odwrócenia kolejności jego elementów. Wszystkie sekwencje zostały zapisane do plików, z których następnie pobierano je podczas wykonywania badań.

Przed rozpoczęciem badań zatrzymano działanie niepotrzebnych programów. Obciążenie systemu nie było idealne, natomiast utrzymywał się na równym poziomie przez cały czas, gdy przeprowadzano badania.
Czas wykonywania algorytmów sortowania dla każdej kombinacji ilości elementów i typu posortowania był mierzony w nanosekundach, a następnie dzielony przez 1000000. **Prezentowane wyniki są więc wartościami w milisekundach.** Zgodnie z informacją zawartą w poleceniu – pierwszy wynik znacząco różnił się od pozostałych, dlatego przy dalszych obliczeniach został zignorowany.
Program obliczał też średnią czasów oraz odchylenia standardowe. Wszystkie wyniki zostały zapisane do plików.

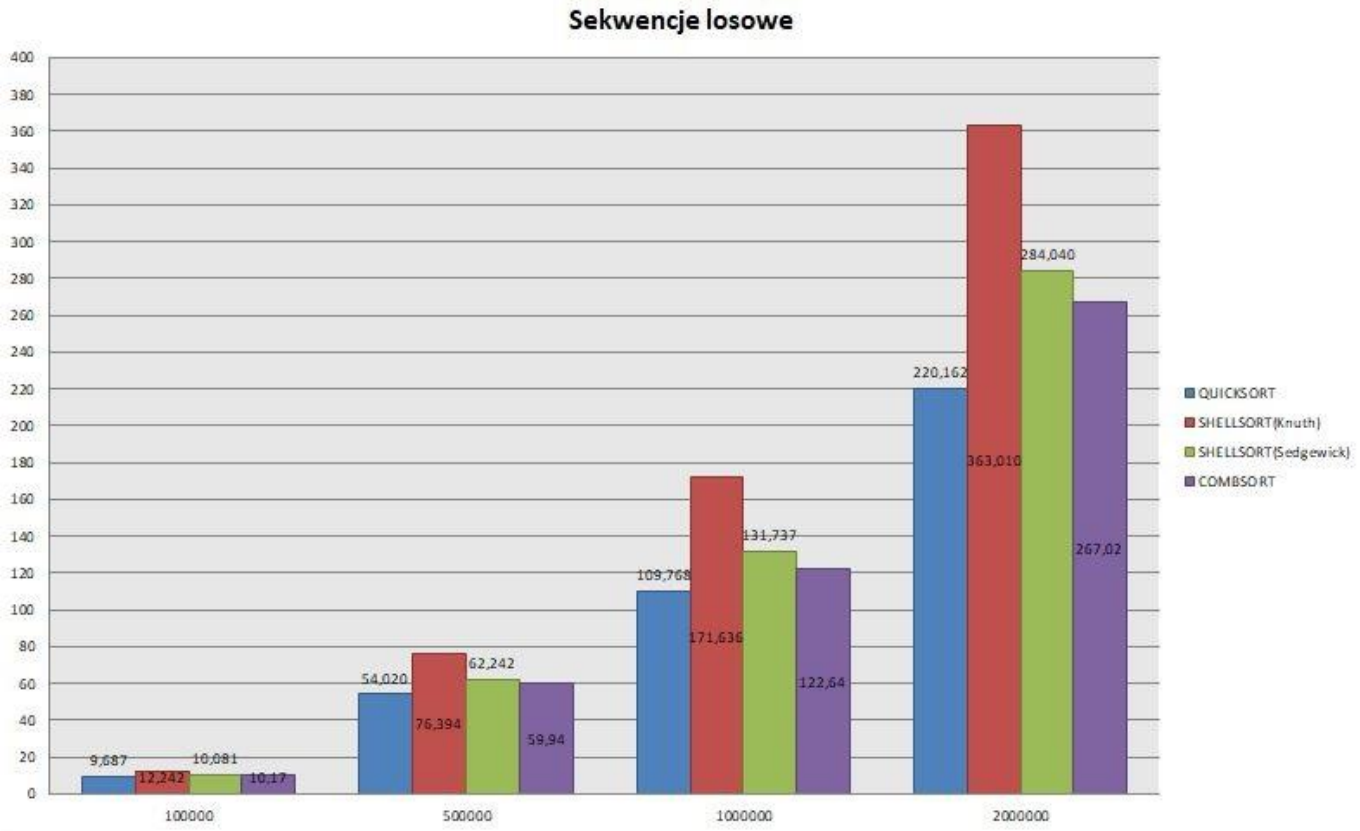
WYNIKI

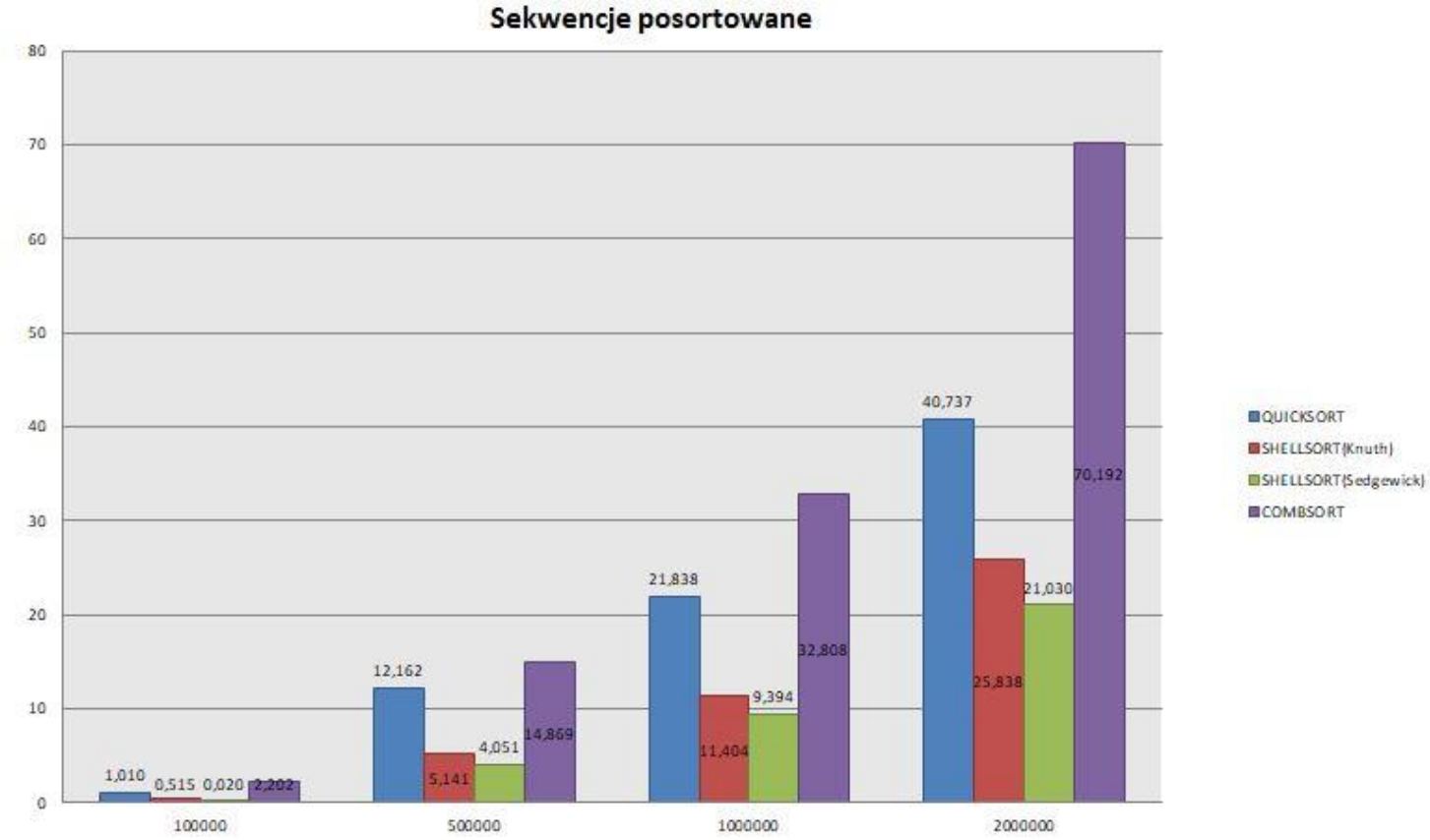
QUICKSORT	100000	odchylenie	500000	odchylenie	1000000	odchylenie	2000000	odchylenie
Losowy	9,687	0,709	54,020	2,070	109,768	3,519	220,162	14,774
Posortowany	1,010	0,101	12,162	2,465	21,838	0,752	40,737	1,516
Posortowany w połowie	5,152	0,861	33,586	4,616	64,929	4,011	123,768	2,035
Posortowany odwrotnie	2,182	2,135	11,384	0,829	24,626	3,228	44,152	3,492

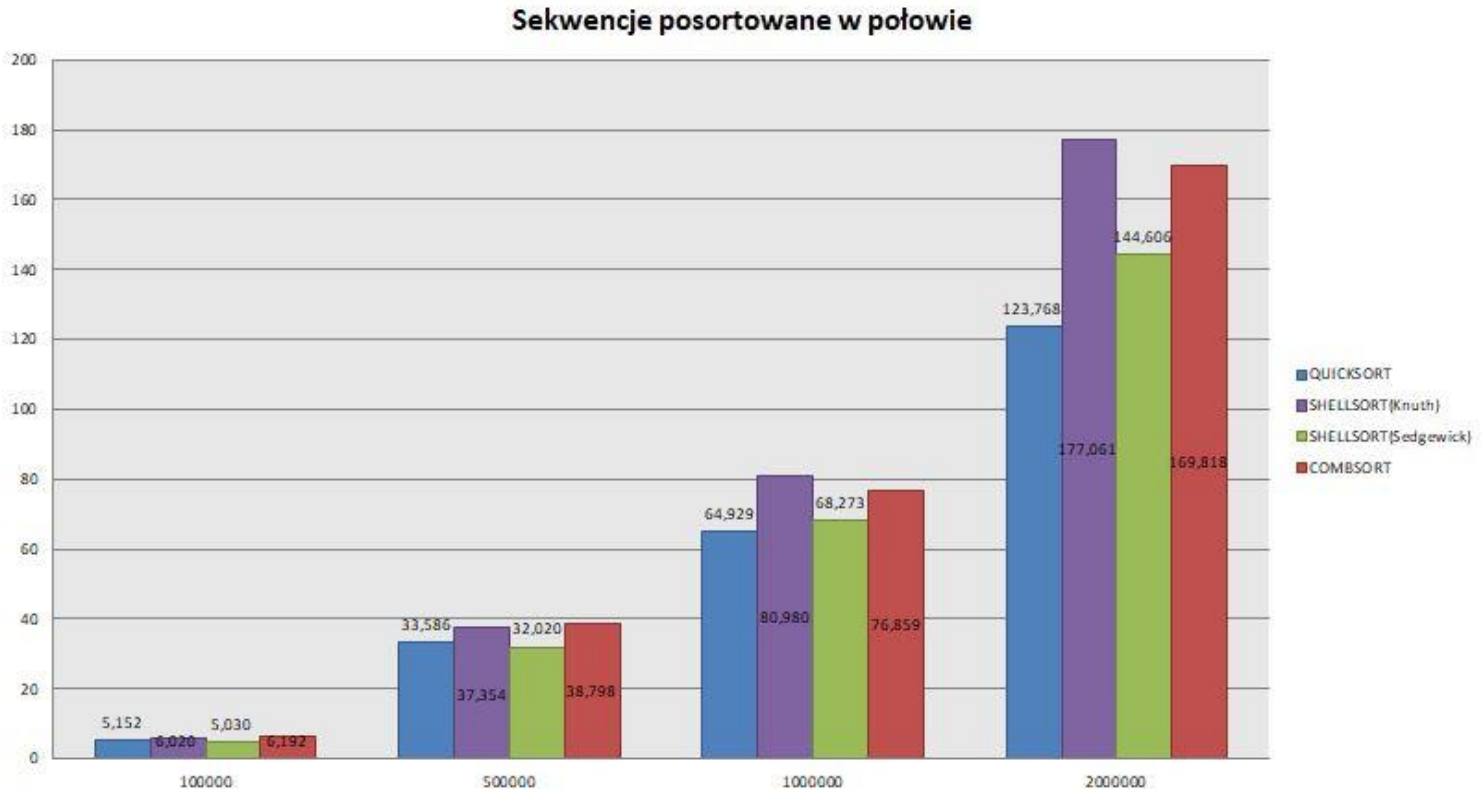
SHELLSORT (Knuth)	100000	odchylenie	500000	odchylenie	1000000	odchylenie	2000000	odchylenie
Losowy	12,242	1,001	76,394	2,044	171,636	9,197	363,010	8,762
Posortowany	0,515	0,502	5,141	0,350	11,404	0,768	25,838	1,695
Posortowany w połowie	6,020	0,589	37,354	1,003	80,980	2,119	177,061	2,924
Posortowany odwrotnie	1,364	0,646	11,394	1,795	22,374	3,151	44,646	1,288

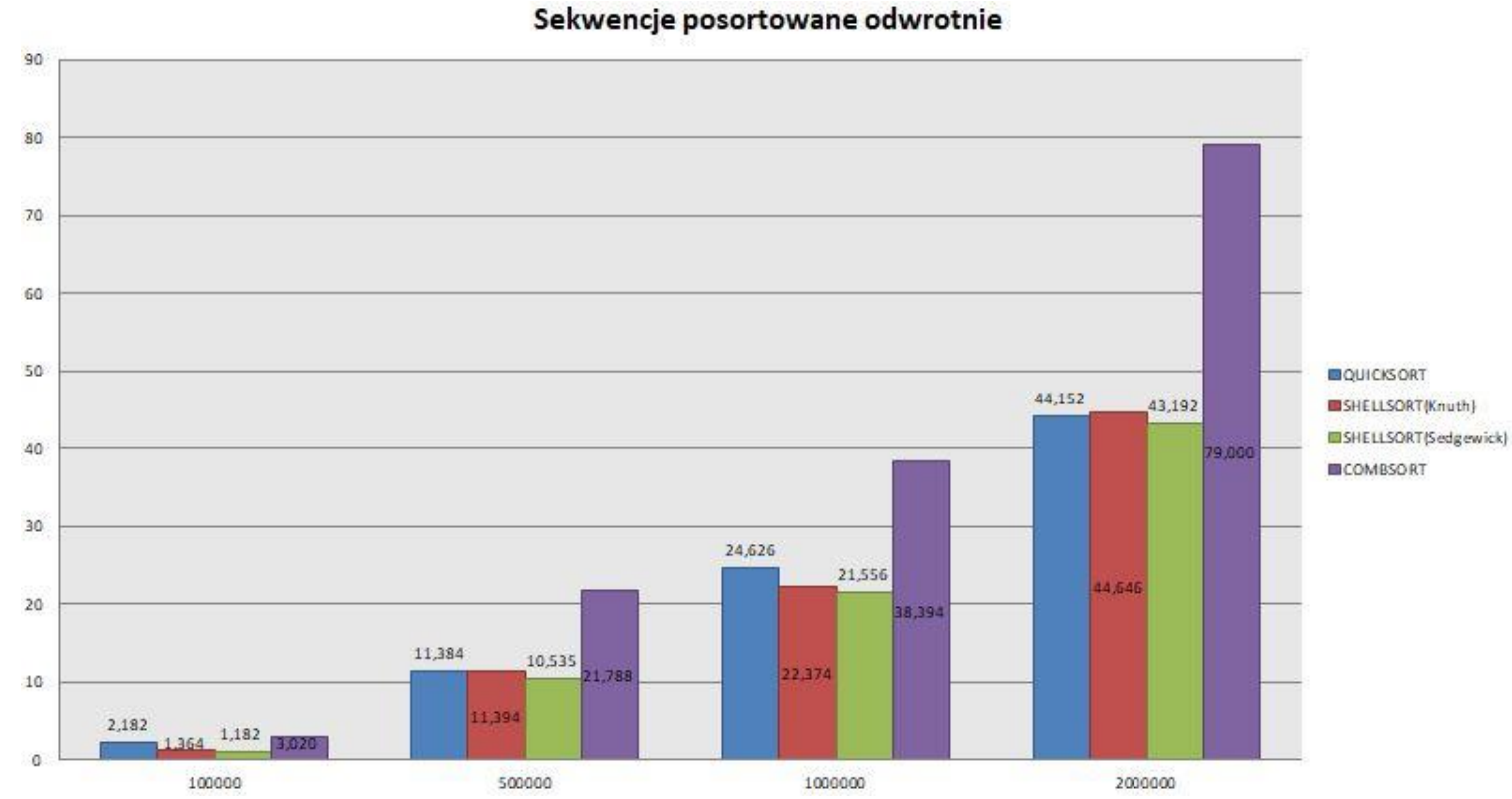
SHELLSORT (Sedgewick)	100000	odchylenie	500000	odchylenie	1000000	odchylenie	2000000	odchylenie
Losowy	10,081	0,369	62,242	1,363	131,737	2,460	284,040	15,296
Posortowany	0,020	0,141	4,051	0,262	9,394	1,331	21,030	1,344
Posortowany w połowie	5,030	0,224	32,020	2,254	68,273	3,425	144,606	5,370
Posortowany odwrotnie	1,182	0,482	10,535	1,240	21,556	0,971	43,192	1,412

COMBSORT	100000	odchylenie	500000	odchylenie	1000000	odchylenie	2000000	odchylenie
Losowy	10,172	0,430	59,939	1,517	122,636	3,621	267,020	8,418
Posortowany	2,202	0,553	14,869	1,569	32,808	1,267	70,192	2,165
Posortowany w połowie	6,192	0,509	38,798	3,232	76,859	4,238	169,818	6,372
Posortowany odwrotnie	3,020	0,141	21,788	8,207	38,394	1,511	79,000	1,505


Sekwencje losowe
400
380
360
340
320
300
280
260
240
220
200
180
160
140
120
100
80
60
40
20
0
9,687
12,242
10,081
10,17
54,020
76,394
62,242
59,94
109,768
171,636
131,737
122,64
220,162
363,010
284,040
267,02
100000
500000
1000000
2000000
QUICKSORT
SHELLSORT(Knuth)
SHELLSORT(Sedgewick)
COMBSORT


Sekwencje posortowane
80
70
60
50
40
30
20
10
0
1,010
0,515
0,020
2,202
12,162
5,141
4,051
14,869
21,838
11,404
9,394
32,808
40,737
25,838
21,030
70,192
100000
500000
1000000
2000000
QUICKSORT
SHELLSORT(Knuth)
SHELLSORT(Sedgewick)
COMBSORT


Sekwencje posortowane w połowie
200
180
160
140
120
100
80
60
40
20
0
5,152
6,020
5,030
6,192
33,586
37,354
32,020
38,798
64,929
80,980
68,273
76,859
123,768
177,061
144,606
169,818
100000
500000
1000000
2000000
QUICKSORT
SHELLSORT(Knuth)
SHELLSORT(Sedgewick)
COMBSORT


Sekwencje posortowane odwrotnie
90
80
70
60
50
40
30
20
10
0
2,182
1,364
1,182
3,020
11,384
11,394
10,535
21,788
24,626
22,374
21,556
38,394
44,152
44,646
43,192
79,000
100000
500000
1000000
2000000
QUICKSORT
SHELLSORT(Knuth)
SHELLSORT(Sedgewick)
COMBSORT

ANALIZA WYNIKÓW

Quicksort

Algorytm ten działa najsprawniej, gdy sortowana jest całkowicie losowa sekwencja. Najgorzej zaś sprawdza się dla ciągów już posortowanych. Jest to zgodne z powszechnymi informacjami o tym algorytmie, a zachowanie takie wynika bezpośrednio z jego sposobu działania.
Warto zauważyć, że quicksort jest też bardzo dobrym rozwiązaniem, gdy sortowana sekwencja składa się z dużej ilości elementów.

Shellsort (Knuth)

Dla odstępów zaproponowanych przez Donalda Knutha algorytm wykazuje najsłabsze działanie podczas sortowania sekwencji całkowicie losowych. W żadnym przypadku nie okazał się on też działać szybciej niż pozostałe algorytmy. Pokrywa się to zatem z teoretyczną wiedzą o tym algorytmie, która została przedstawiona we wstępie. Wskazana tam złożoność obliczeniowa była najgorsza wśród wszystkich badanych algorytmów, co znajduje odzwierciedlenie w wynikach.
Przyglądając się wykresowi dla sekwencji posortowanych odwrotnie można zauważyć, że choć nie generuje on najniższych czasów, to jednak wyniki te nie różnią się one znacznie od tych, które otrzymano dla Quicksorta oraz algorytmu Shella z ciągiem dystansów Sedgewicka.

Shellsort (Sedgewick)

Algorytm ten działa zdecydowanie najszybciej z badanych algorytmów dla ciągów już posortowanych. Dotyczy to także sekwencji posortowanych odwrotnie, jednakże w tym przypadku różnice nie są aż tak znaczące.
Algorytm nie jest najlepszym wyborem, gdy sortujemy ciąg całkowicie losowy, zwłaszcza, gdy sortowana kolekcja składa się z wielu elementów.

Combsort

Algorytm działa najsłabiej dla ciągów już posortowanych. Na tle badanych algorytmów nie wypada też najlepiej w przypadku ciągów losowych. Zdecydowanie najgorzej reaguje on na zwiększenie ilości sortowanych elementów.

WNIOSKI

Uzyskane wyniki raczej pokrywają się z wiedzą teoretyczną.
Odchylenia standardowe, choć występowały to nie wskazywały dużych rozbieżności.
Ewentualne błędy mogą wynikać z niedostatecznej ilości sortowanych ciągów. Mogą one być także spowodowane zakłóceniami związanymi z niewystarczająco wyeliminowanym obciążeniem systemu.

Uzyskane wyniki jasno pokazują, że nie istnieje idealny algorytm sortowania. W zależności od rodzaju sortowanej sekwencji oraz ilości sortowanych elementów sprawność algorytmów zmienia się. Należy zatem zdawać sobie sprawę zarówno z zalet jak i wad każdego algorytmu i rozważnie dobierać je podczas wykorzystywania do różnych celów. Odpowiednie przeanalizowanie warunków w jakich będzie pracował algorytm pozwoli na uzyskanie zdecydowanie lepszej złożoności obliczeniowej.