Shell Sort

Sortowanie Shella (ang. *Shellsort*) – jeden z algorytmów sortowania działających w miejscu i korzystających z porównań elementów. Można go traktować jako uogólnienie sortowania przez wstawianie lub sortowania bąbelkowego, dopuszczające porównania i zamiany elementów położonych daleko od siebie. Na początku sortuje on elementy tablicy położone daleko od siebie, a następnie stopniowo zmniejsza odstępy między sortowanymi elementami. Dzięki temu może je przenieść w docelowe położenie szybciej niż zwykłe sortowanie przez wstawianie.

Opis algorytmu:
Sortowanie Shella to algorytm wieloprzebiegowy. Kolejne przebiegi polegają na sortowaniu przez proste wstawianie elementów oddalonych o ustaloną liczbę miejsc **h** czyli tak zwanym **h**-sortowaniu.
Poniżej zilustrowano sortowanie przykładowej tablicy metodą Shella z odstępami 5, 3, 1.

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}
dane wejściowe:	62	83	18	53	07	17	95	86	47	69	25	28
po 5-sortowaniu:	17	28	18	47	07	25	83	86	53	69	62	95
po 3-sortowaniu:	17	07	18	47	28	25	69	62	53	83	86	95
po 1-sortowaniu:	07	17	18	25	28	47	53	62	69	83	86	95

Pierwszy przebieg, czyli 5-sortowanie, sortuje osobno przez wstawianie zawartość każdego z fragmentów (a1, a6, a11) , (a2, a7, a12) , (a3, a8), (a4, a9), (a5, a10) Na przykład fragment (a1, a6, a11) zmenia (62, 17, 25) na (17, 25, 62). Następny przebieg, czyli 3-sortowanie, sortuje przez wstawianie zawartość fragmentów (a1, a4, a7, a10) , (a2, a5, a8, a11), (a3, a6, a9, a12).

Ostatni przebieg, czyli 1-sortowanie, to zwykłe sortowanie przez wstawianie całej tablicy (a1, . . . , a12).

Jak widać, fragmenty tablicy, na których operuje algorytm Shella, są z początku krótkie, a pod koniec dłuższe, ale prawie uporządkowane. W obu tych przypadkach sortowanie przez proste wstawianie działa wydajnie.

Sortowanie Shella nie jest stabilne, czyli może nie zachowywać wejściowej kolejności elementów o równych kluczach. Wykazuje ono zachowanie naturalne, czyli krótszy czas sortowania dla częściowo uporządkowanych danych wejściowych.

weźmy 6 ciągów, pierwszy 3 ciągi wymyślone wczesniej pozostaly 3 przeze mnie
Wyraz ogólny ciągów:

Sedgewick = $4^k + 3 \cdot 2^{k-1} + 1$ $O(N^{4/3})$ *262913 65921 16577 4193 1073 281 77 23 8 1*

Tokuda = $1{,}8 \cdot 2{,}25^{k-1} - 0{,}8$ $O(N^{3/2})$ *153401 68178 30301 13467 5985 2660 1182 525 233 103 46 20 9 4 1*

Khuth = $(3^k - 1)/2$ $O(N^{4/3})$ *265720 88573 29524 9841 3280 1093 364 121 40 13 4 1*

MySequence-1 = $8^k + 6 \cdot 2^k + 1$ *262529 32961 4193 561 89 21 1*

MySequence-2 = $2 \cdot k + (5^{k+1} / k-1)$ *244155 55816 13031 3133 788 213 65 25 1*

MySequence-3 = $(e^{(k+k)-1} / e^k) \cdot (e^k / k+k+5)$ *408640 63213 9990 1630 283 57 17 8 1*

Rząd złożoności wyrazów MySequence-1 MySequence-2 MySequence-3 znajduje się pomiędzy $O(N^{4/3})$ a $O(N^{3/2})$ przy czym rząd złożoności wyrazu MySequence-1 jest większy od MySequence-2 MySequence-3 zobaczymy to na poniższych tablicach.

Dla otrzymania jak najbardziej dokladnych wyników (czas sortowania) każdy ciąg będzie wywolany do posortowania 5 raz dla każdej tablicy, i na mocy tych 5-u wywolań dostajemy średniu wartość która i będzie czasem dźialania algorytmu na tablice.

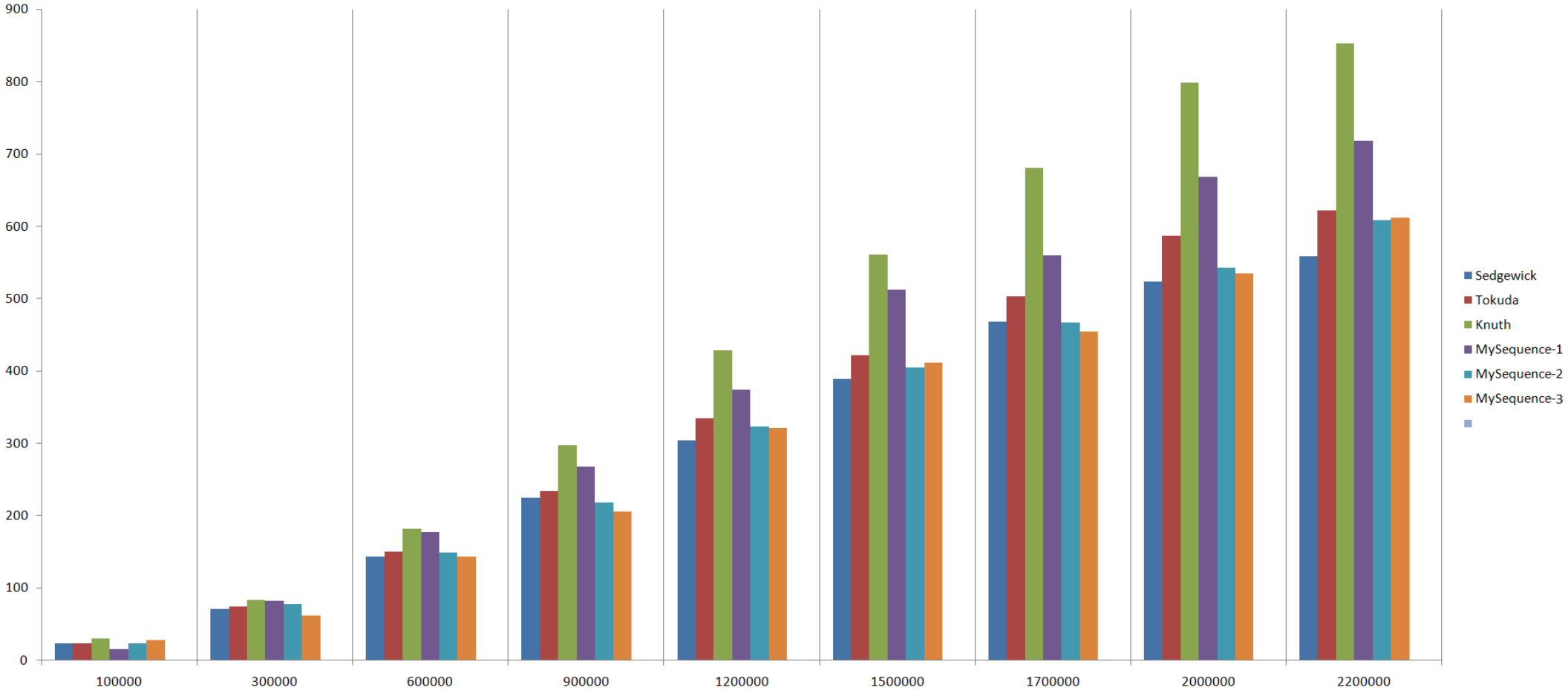
	100.000						300.000						600.000						900.00					
Sedgewick	30	13	16	31	31	24,2	61	63	78	63	93	71,6	156	140	140	140	141	143,4	218	219	203	257	228	225
Tokuda	16	31	32	15	15	24,2	77	78	63	78	78	74,8	141	141	156	161	151	150	234	230	239	234	234	234,2
Knuth	31	31	27	32	32	30,6	78	93	78	78	94	84,2	187	187	172	175	191	182,4	281	334	312	281	281	297,8
MySequence-1	16	16	16	15	15	15,6	78	94	78	78	86	82,8	167	176	172	203	171	177,8	266	250	250	307	266	267,8
MySequence-2	31	16	16	24	31	23,6	78	94	78	78	63	78,2	147	129	140	191	141	149,6	220	218	219	221	213	218,2
MySequence-1	31	31	31	31	16	28	63	62	63	63	63	62,8	147	141	134	172	125	143,8	193	213	213	203	211	206,6

	1.200.000						1.500.000						1.700.000						2.000.000					
Sedgewick	315	310	300	304	292	304,2	408	384	387	378	390	389,4	462	442	431	572	435	468,4	559	506	515	499	542	524,2
Tokuda	323	357	331	333	332	335,2	422	421	422	423	424	422,4	506	481	486	573	471	503,4	645	570	559	567	597	587,6
Knuth	411	423	428	435	448	429	569	556	570	555	558	561,6	653	661	764	707	625	682	834	809	807	783	763	799,2
MySequence-1	366	366	389	376	377	374,8	483	572	474	503	530	512,4	597	552	540	569	545	560,6	694	715	648	646	645	669,6
MySequence-2	307	316	340	321	333	323,4	408	401	409	403	405	405,2	459	470	471	489	448	467,4	532	523	553	567	543	543,6
MySequence-1	294	304	399	305	307	321,8	415	423	422	398	405	412,6	449	478	447	456	446	455,2	523	543	528	525	560	535,8

Na mocy powyższej tablicy otrzymujemy tablicu o średnem czasu dźialania algorytmu Shell z róznymi ciagami

	100.000	300.000	600.000	900.000	1.200.000	1.500.000	1.700.000	2.000.000	2.200.000
Sedgewick	24,2	71,6	143,4	225	304,2	389,4	468,4	524,2	559,2
Tokuda	24,2	74,8	150	234,2	335,2	422,4	503,4	587,6	622,2
Knuth	30,6	84,2	182,4	297,8	429	561,6	682	799,2	854,4
MySequence-1	15,6	82,8	177,8	267,8	374,8	512,4	560,6	669,6	719,2
MySequence-2	23,6	78,2	149,6	218,2	323,4	405,2	467,4	543,6	608,6
MySequence-1	28	62,8	143,8	206,6	321,8	412,6	455,2	535,8	613

Graficzna interpretacja powyzszej tablicy



Kod programy:

Funkcji do wzoru na ciągi

```
int sedgewick1(int k){return pow(4, k)+ 3 * pow(2, k-1) + 1;}
int tokuda(int k){return ceil(1.8*powf(2.25, k-1) - 0.8);}
int knuth(int k){return (pow(3,k)-1)/2;}
int mySequence1(int k){return pow(8, k)+6*pow(2,k)+1;}
int mySequence2(int k){return (2*k) + (pow(5,k+1)/k-1);}
int mySequence3(int k){return((pow(exp(1),(k+k)-1 )/pow(exp(1), k))*(exp(k)))/k+k+5;}
```

Funkcjia która zwraca ciag

```
int* createArraySequence(int sizeOrigin, int &sizeSequence, int(*sequence)(int k))
{
   int step = 1;
   int k = 1;
   while(k < sizeOrigin)
       k = sequence(step);
       step++;
   step--;
   int *arr = new int[step];
   arr[0] = 1;
   for(int i = 1; i < step ; i++)
        arr[i] = sequence(i);
   sizeSequence = step;
   spreadArray(arr, sizeSequence);
   if(arr[sizeSequence-1] == arr[sizeSequence-2])
        sizeSequence--;
   return arr;
}
```