Martyna Demczuk
Julia Bandera

Data oddania: 16.03.2020

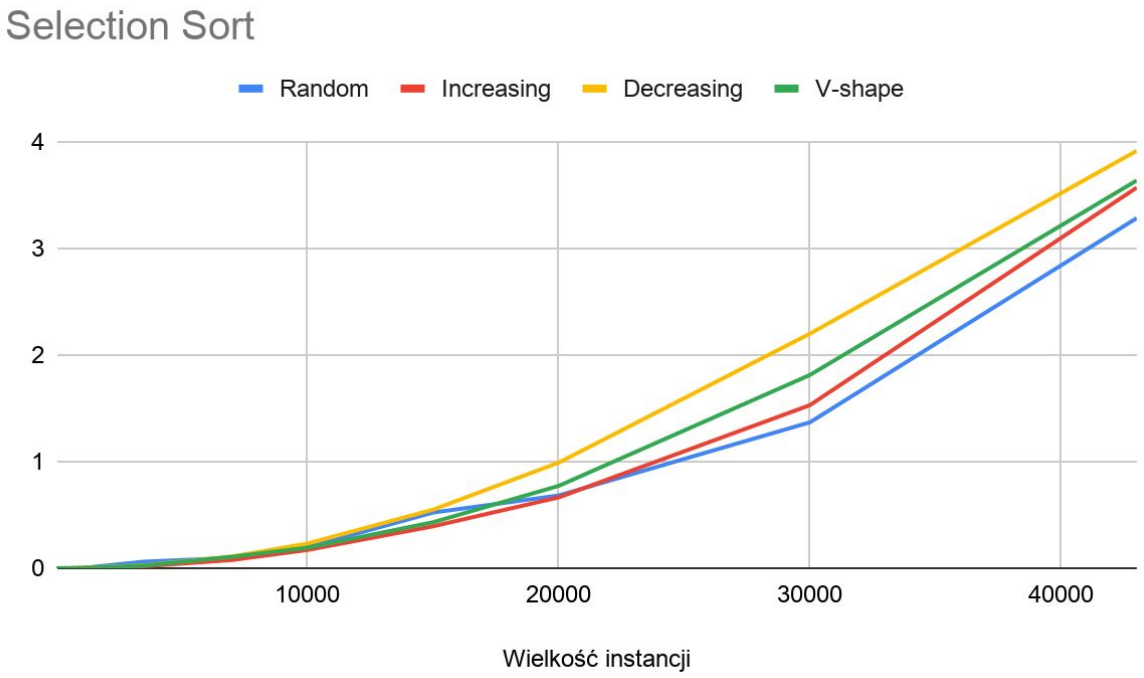
Sprawozdanie

Algorytmy sortujące

Tabele i wykresy znajdujące się w sprawozdaniu przedstawiają czasy (w sekundach), które są potrzebne do wykonania sortowań w zależności od rodzaju wybranego sortowania oraz od ilości wprowadzonych elementów do posortowania i rozkładu danych wejściowych.

SELECTION SORT

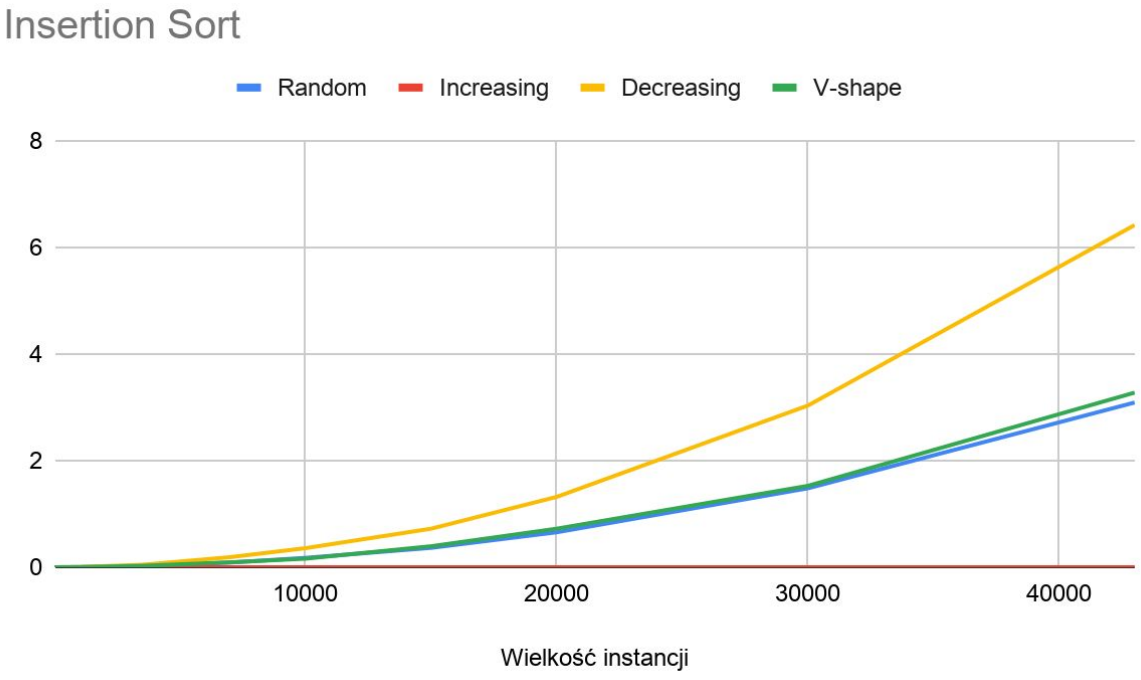
Wielkość instancji	Typ rozkładu danych wejściowych (czas)			
	Random	Increasing	Decreasing	V-shape
10	0	0	0	0
500	0,001	0	0,001	0
1000	0,001	0,002	0,003	0,002
3500	0,061	0,018	0,027	0,024
7000	0,097	0,077	0,11	0,107
10000	0,198	0,171	0,231	0,192
15000	0,522	0,393	0,549	0,431
20000	0,682	0,664	0,99	0,772
30000	1,368	1,529	2,199	1,812
43000	3,282	3,568	3,913	3,635



Dla algorytmu Selection Sort czas wykonywania zadania rośnie dla każdego typu rozkładu danych wejściowych wraz ze wzrostem wielkości instancji. Najlepszym przypadkiem jest rozkład Random, natomiast najgorszym - Decreasing. Rozkłady Increasing i V-shape znajdują się z kolei pomiędzy.

INSERTION SORT

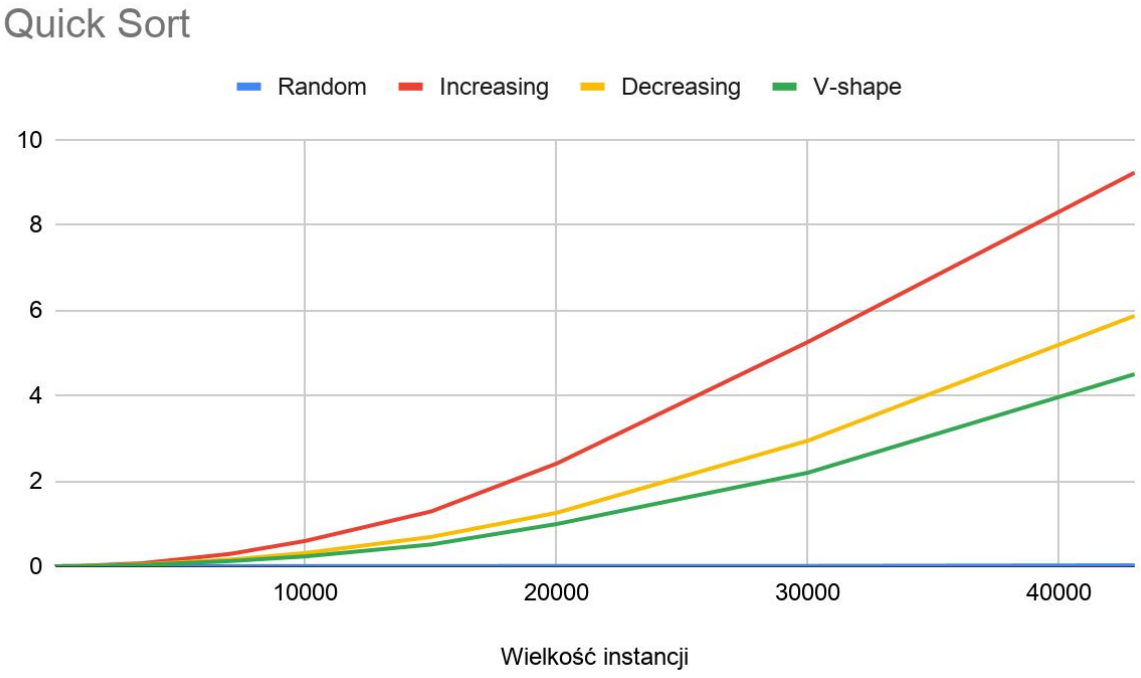
Wielkość instancji	Typ rozkładu danych wejściowych (czas)			
	Random	Increasing	Decreasing	V-shape
10	0	0	0	0
500	0,001	0	0,001	0
1000	0,002	0	0,004	0,002
3500	0,022	0	0,045	0,026
7000	0,087	0	0,188	0,091
10000	0,175	0	0,357	0,16
15000	0,363	0	0,722	0,393
20000	0,655	0	1,319	0,722
30000	1,48	0	3,032	1,526
43000	3,088	0	6,415	3,274



Dla algorytmu Insertion Sort czas wykonywania zadania rośnie dla każdego typu rozkładu danych wejściowych wraz ze wzrostem wielkości instancji, za wyjątkiem rozkładu Increasing, jest on zatem najlepszym przypadkiem. Najgorszym jest natomiast Decreasing. Rozkłady Random i V-shape znajdują się pomiędzy.

QUICK SORT

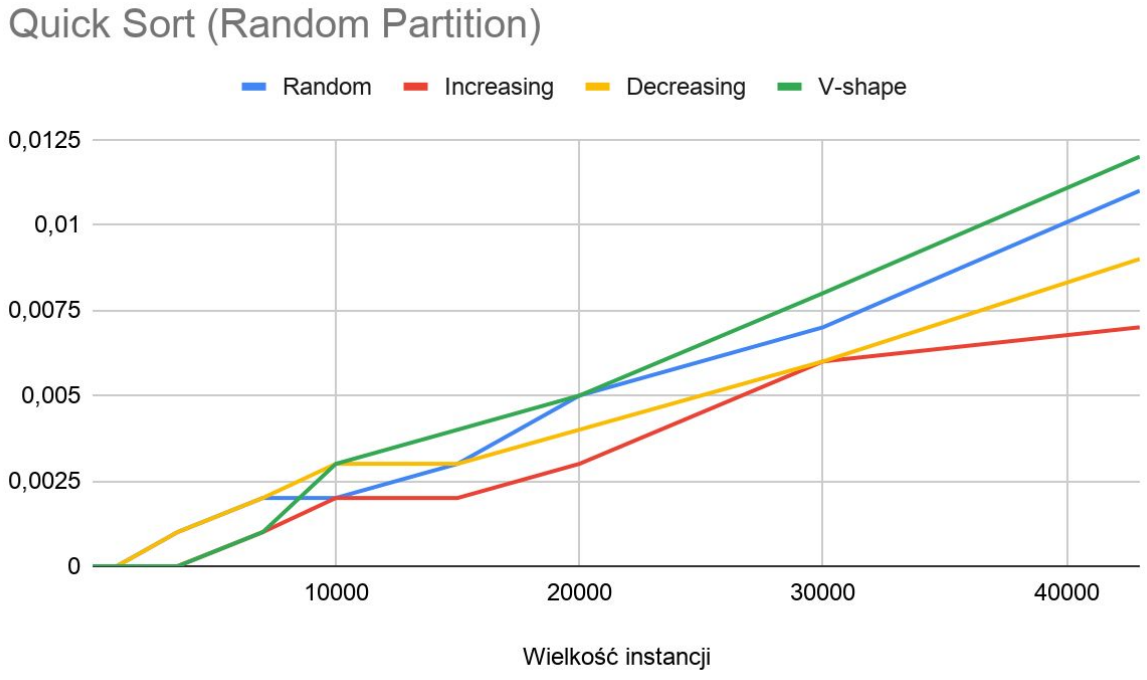
Wielkość instancji	Typ rozkładu danych wejściowych (czas)			
	Random	Increasing	Decreasing	V-shape
10	0	0	0	0
500	0,001	0,001	0	0,001
1000	0	0,006	0,003	0,003
3500	0,001	0,069	0,04	0,03
7000	0,001	0,288	0,154	0,12
10000	0,002	0,593	0,31	0,23
15000	0,003	1,283	0,685	0,508
20000	0,004	2,407	1,252	0,989
30000	0,006	5,256	2,943	2,192
43000	0,021	9,226	5,869	4,501



Dla algorytmu Quick Sort czas wykonywania zadania rośnie dla każdego typu rozkładu danych wejściowych wraz ze wzrostem wielkości instancji, jednak dla rozkładu Random rośnie nieznacznie, jest on zatem najlepszym przypadkiem. Najgorszym jest natomiast Increasing. Rozkłady Decreasing i V-shape znajdują się pomiędzy.

QUICK SORT (RANDOM PARTITION)

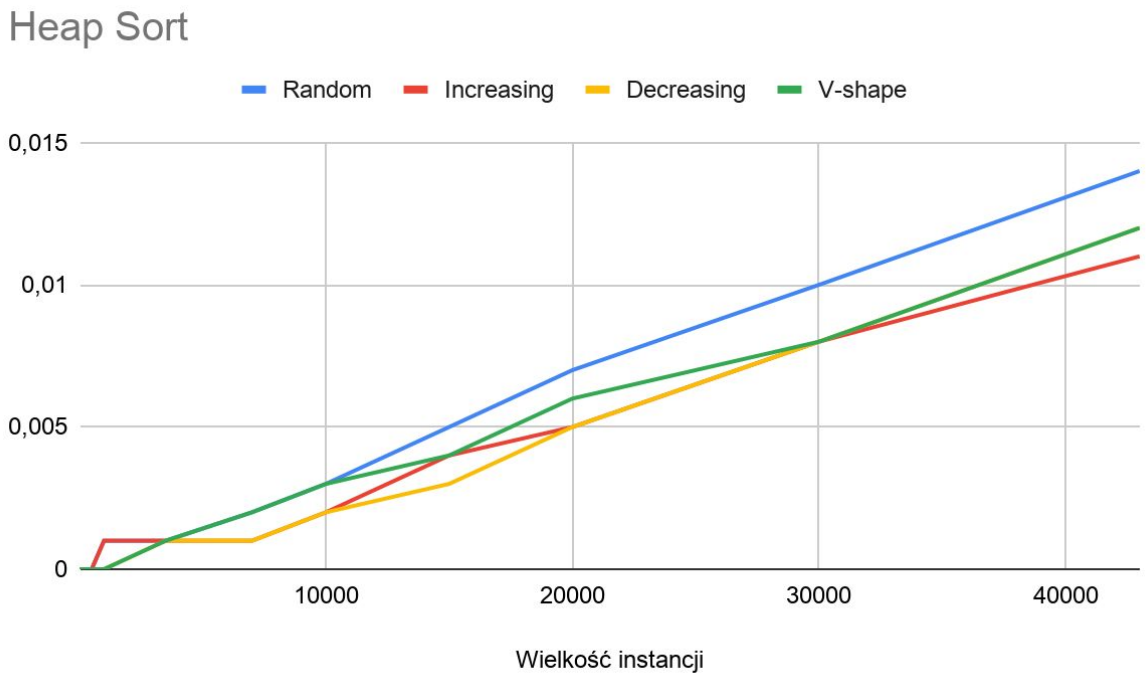
Wielkość instancji	Typ rozkładu danych wejściowych (czas)			
	Random	Increasing	Decreasing	V-shape
10	0	0	0	0
500	0	0	0	0
1000	0	0	0	0
3500	0,001	0	0,001	0
7000	0,002	0,001	0,002	0,001
10000	0,002	0,002	0,003	0,003
15000	0,003	0,002	0,003	0,004
20000	0,005	0,003	0,004	0,005
30000	0,007	0,006	0,006	0,008
43000	0,011	0,007	0,009	0,012



Dla algorytmu Quick Sort z losowym elementem rozdzielającym czas wykonywania zadania rośnie dla każdego typu rozkładu danych wejściowych wraz ze wzrostem wielkości instancji, jednak dla początkowych wynosił zero. Rozkład Increasing jest najlepszym przypadkiem, a najgorszym - V-shape. Rozkłady Random i Decreasing znajdują się pomiędzy.

HEAP SORT

Wielkość instancji	Typ rozkładu danych wejściowych (czas)			
	Random	Increasing	Decreasing	V-shape
10	0	0	0	0
500	0	0	0	0
1000	0,001	0,001	0	0
3500	0,001	0,001	0,001	0,001
7000	0,002	0,001	0,001	0,002
10000	0,003	0,002	0,002	0,003
15000	0,005	0,004	0,003	0,004
20000	0,007	0,005	0,005	0,006
30000	0,01	0,008	0,008	0,008
43000	0,014	0,011	0,012	0,012



Dla algorytmu Heap Sort czas wykonywania zadania rośnie dla każdego typu rozkładu danych wejściowych wraz ze wzrostem wielkości instancji, ale dla początkowych wynosił zero.W tym algorytmie ciężko wybrać najlepszy i najgorszy rozkład, ponieważ różnica w czasie wykonywania zadań w każdym rozkładzie jest znikoma. Jednak po bardzo dokładnym przeanalizowaniu wyników można stwierdzić, że najlepszym przypadkiem jest rozkład

Decreasing. Rozkład Random jest z kolei najgorszym przypadkiem. Rozkłady Increasing i V-shape znajdują się pomiędzy.

Poniższa tabela prezentuje złożoność obliczeniową poszczególnych sortowań.

ZŁOŻONOŚĆ OBLICZENIOWA

Algorytm sortujący	Najlepszy przypadek	Przeciętny przypadek	Najgorszy przypadek
Selection Sort	$\Omega(n^2)$	$\theta(n^2)$	$O(n^2)$
Insertion Sort	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$
Quick Sort	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n^2)$
Heap Sort	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n \log(n))$

WNIOSKI

Zaprezentowane algorytmy sortowania różnią się czasem, a co za tym idzie - efektywnością. To, który algorytm sortowania jest najlepszy, zależy na przykład od tego ile elementów sortujemy albo od tego w jaki sposób ułożone były dane wejściowe (losowo, rosnąco, itd.).

Szybkość sortowania dla jednego typu rozkładu danych wejściowych nie oznacza, że dla innego typu rozkładu danych tego samego sortowania, czas będzie równie korzystny. W Insertion Sort przy liczbach ułożonych rosnąco w tablicy czas sortowania wynosił 0 sekund, niezależnie od ilości liczb. Jednak, gdy liczby ułożone były malejąco czas sortowania rósł wraz ze zwiększaniem ilości liczb (przy 43000 liczb sortowanie trwało 6,415 sekund).

Najszybszym algorytmem jest Quick Sort, w którym elementem rozdzielającym jest losowy element tablicy. Jego złożoność obliczeniowa wynosi $O(n \log n)$. Dla każdego typu rozkładu liczb wejściowych algorytm ten był porównywalnie szybki. Dla 43000 liczb czas sortowania wynosił 0,007 - 0,012 s. Jest to najkorzystniejszy algorytm przy sortowaniu do ok. 40000 liczb. Przy próbie sortowania większej ilości liczb, ze względu na rekurencję, program się zawieszał, ponieważ kończyła się pamięć.

Drugim najszybszym algorytmem sortowania jest Heap Sort, którego wydajność jest niewiele mniejsza od Quick Sort. Dodatkowo złożoność obliczeniowa Heap Sort w pesymistycznym

przypadku jest taka sama jak w optymistycznym przypadku. Może być alternatywą dla algorytmu Quick Sort, którego złożoność w pesymistycznym przypadku wynosi $O(n^2)$.

Podsumowując, idealny algorytm sortowania dla każdej ilości i każdego rozłożenia liczb nie istnieje. Optymalność algorytmu zależy od typu rozkładu danych oraz ich ilości.

Przy implementacji algorytmu Quick Sort podczas porównywania naszego kodu z pseudokodem zawartym w materiałach pomocniczych zauważyłyśmy, że funkcja *partition* powinna zwracać (i + 1), zamiast i, jak jest w pseudokodzie.