Algorytmy sortowania

Sprawozdanie

Przygotowali:
Andrzej Wieczorek
Adrian Sworowski

Spis treści

1. Założenia i cele zadania

Celem zadania było przygotowanie programu zawierającego w sobie cztery różne metody sortowania opierające się o określone algorytmy. W kodzie dostarczonym poprzez platformę Github Classroom należało przygotować generatory instancji liczb – wartości rosnących, malejących, losowych i tzw. rozkład V-kształtny. Ponadto należało zaimplementować algorytmy sortowania – Selection Sort, Insertion Sort, Quick Sort w dwóch wariantach (z wartością ostatnią lub losową jako wartość rozdzielającą) oraz Heap Sort. Wszystkie algorytmy miały mieć zmierzone czasy ich działania dla wybranych wielkości instancji, przedstawiając je za pośrednictwem tabeli wartości oraz wykresu.

2. Metodyka testowa

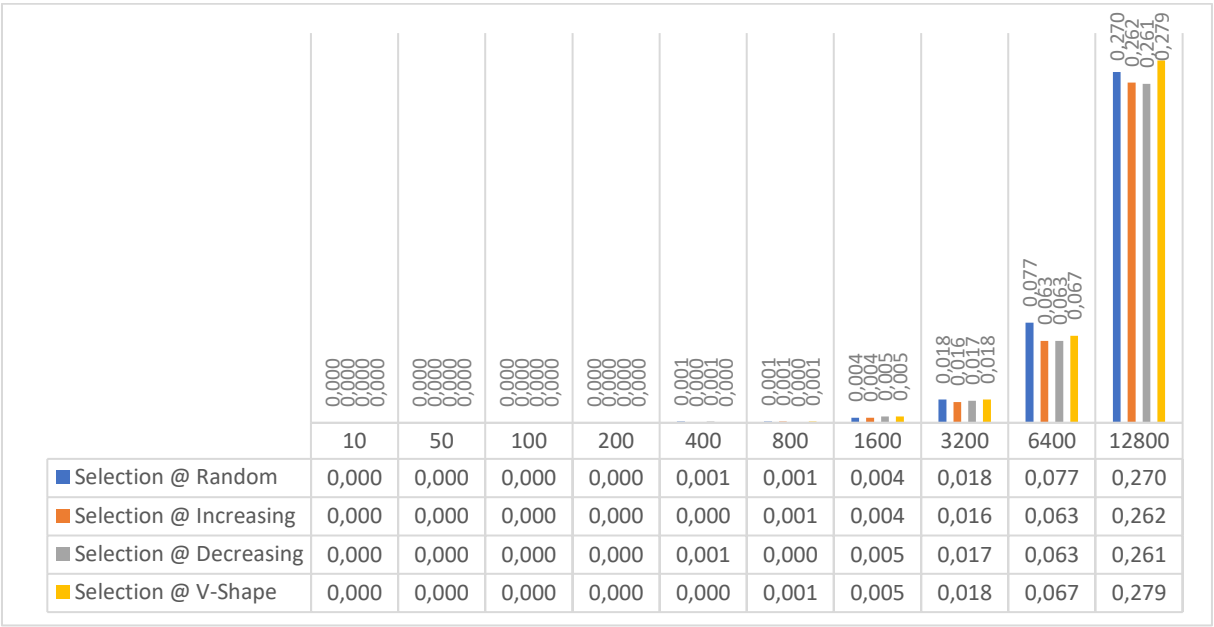
Czasy zostały zmierzone za pośrednictwem dostarczonego kodu. Wartości zostały wpisane do pliku .txt za pośrednictwem odpowiedniej modyfikacji w/w, a następnie wczytane do programu Excel celem dalszej obróbki oraz przygotowania wykresów.

Kod został przygotowany dla dziesięciu wielkości próbek – 10, 50, 100, 200, 400, 800, 1600, 3200, 6400 oraz 12800. Czas wykonania algorytmu dla każdej z tych próbek został przekazany do wykresu i odpowiednich tabel. Zawarte dane są uśrednione dla dziesięciu kolejnych uruchomień programu. W punkcie 7. wykorzystane zostały czasy tylko dla największej próbki.

3. Algorytm Selection Sort

	Random	Increasing	Decreasing	V-Shape
10	0,000	0,000	0,000	0,000
50	0,000	0,000	0,000	0,000
100	0,000	0,000	0,000	0,000
200	0,000	0,000	0,000	0,000
400	0,001	0,000	0,001	0,000
800	0,001	0,001	0,000	0,001
1600	0,004	0,004	0,005	0,005
3200	0,018	0,016	0,017	0,018
6400	0,077	0,063	0,063	0,067
12800	0,270	0,262	0,261	0,279

Tabela 1. Czasy wykonania algorytmu Selection Sort



Wykres 1. Wykres przedstawiający dane zawarte z tabeli wyżej

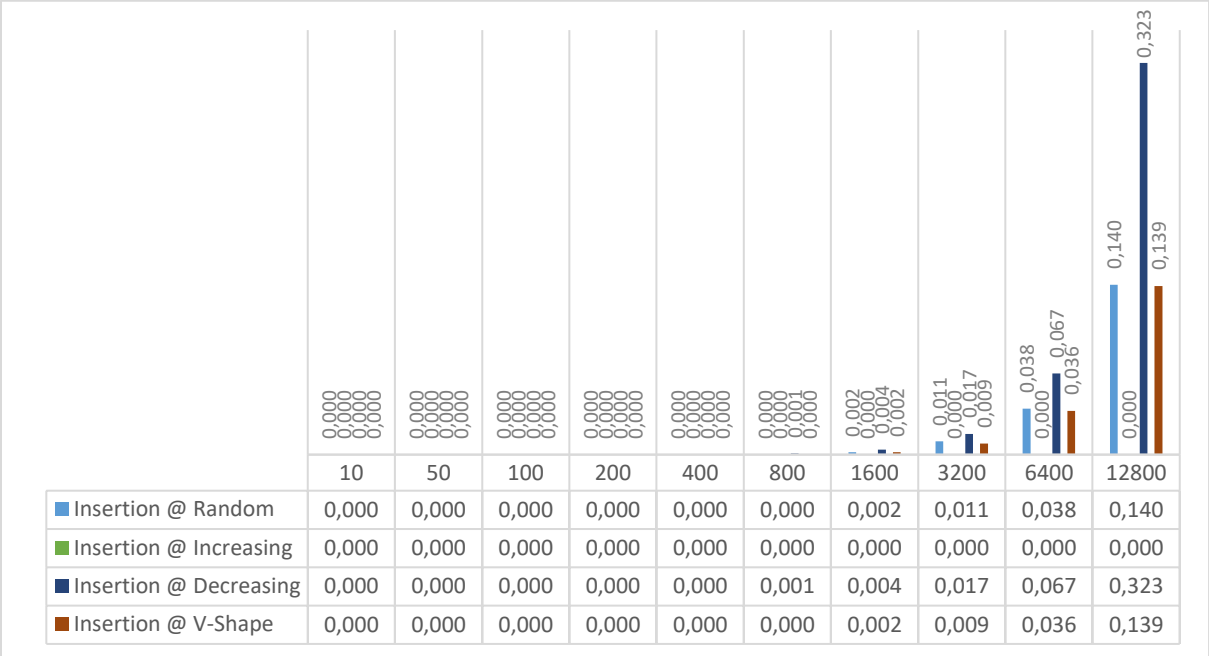
Algorytm Selection Sort jest jednym z bazowych algorytmów sortowania, opierających się na poszukiwaniu najmniejszego elementu w zakresie od 0 do n, a następnie zastępowaniu kolejnych elementów tablicy przez te, które z każdym krokiem sortowania będą najmniejsze. Algorytm ten nie jest stabilny – elementy o tej samej wartości mogą wystąpić w innej kolejności.

Prędkość działania algorytmu różni się tylko marginalnie bez względu na to, jakiego rodzaju instancji były przyjmowane dane i bez względu na wielkość próbki. Wskazuje to więc na to, że bez względu na przypadek (najlepszy, średni, czy też najgorszy), złożoność algorytmu zawsze będzie wynosić $O(n^2)$.

4. Algorytm Insertion Sort

	Random	Increasing	Decreasing	V-Shape
10	0,000	0,000	0,000	0,000
50	0,000	0,000	0,000	0,000
100	0,000	0,000	0,000	0,000
200	0,000	0,000	0,000	0,000
400	0,000	0,000	0,000	0,000
800	0,000	0,000	0,001	0,000
1600	0,002	0,000	0,004	0,002
3200	0,011	0,000	0,017	0,009
6400	0,038	0,000	0,067	0,036
12800	0,140	0,000	0,323	0,139

Tabela 2. Czasy wykonania algorytmu Insertion Sort



Wykres 2. Wykres przedstawiający dane zawarte z tabeli wyżej

Algorytm Insertion Sort również należy do bazowych algorytmów sortowania. Wybiera on dowolny element z tabeli, traktując go jako element posortowany, po czym porównuje kolejne elementy (czy są mniejsze, czy większe od elementów z tabeli „posortowanej") i odpowiednio je układa. Algorytm ten jest algorytmem stabilnym.

Teoria tego algorytmu jest widoczna w praktyce – instancja rosnąca ma tutaj zdecydowanie największą prędkość wykonania, porównując go nawet z innymi, szybszymi metodami sortowania. W pozostałych przypadkach – jakby nie patrzeć, ważniejszych z punktu widzenia użytkownika – sytuacja jest zupełnie inna, prezentując znacznie wyższe wyniki, zwłaszcza dla instancji malejącej. Stąd też możemy przyjąć, że w przypadku pesymistycznym, dla tabeli posortowanej malejąco, złożoność algorytmu wyniesie $O(n^2)$.

5. Algorytm Quick Sort

Algorytm Quick Sort jest prawdopodobnie najpopularniejszym algorytmem sortowania ze względu na jego prostotę działania i efektywność przy odpowiedniej implementacji. Jest on wykorzystywany w wielu językach programowania jako standardowa metoda sortowania. Polega on na znalezieniu elementu rozdzielającego, według którego zostanie podzielona tablica zawierająca elementy do sortowania – do początkowego fragmentu tablicy przenoszone są wszystkie elementy nie większe, do końcowego wszystkie większe. Potem sortuje się osobno początkowy i końcowy fragment. Sortowanie kończy się, gdy po podziale pozostanie pojedynczy fragment, nie wymagający dalszego sortowania.

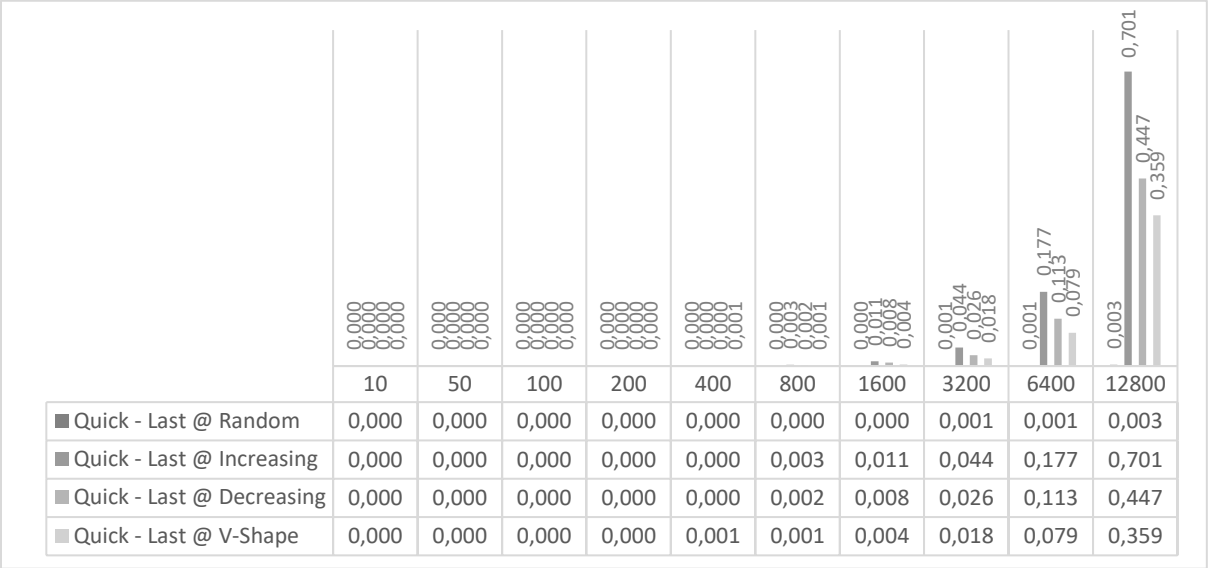
Jest to algorytm stabilny, będący również przykładem sortowania w miejscu.

Ten algorytm miał zostać przygotowany w dwóch wariantach. W pierwszym przypadku wartością rozdzielającą miał być ostatni element z tabeli wartości, w drugim – wartość losowa z tabeli miała być wartością rozdzielającą.

5.1. Wariant z ostatnią wartością jako rozdzielającą

	Random	Increasing	Decreasing	V-Shape
10	0,000	0,000	0,000	0,000
50	0,000	0,000	0,000	0,000
100	0,000	0,000	0,000	0,000
200	0,000	0,000	0,000	0,000
400	0,000	0,000	0,000	0,001
800	0,000	0,003	0,002	0,001
1600	0,000	0,011	0,008	0,004
3200	0,001	0,044	0,026	0,018
6400	0,001	0,177	0,113	0,079
12800	0,003	0,701	0,447	0,359

Tabela 3. Czasy wykonania algorytmu Quick Sort dla ostatniego elementu rozdzielającego



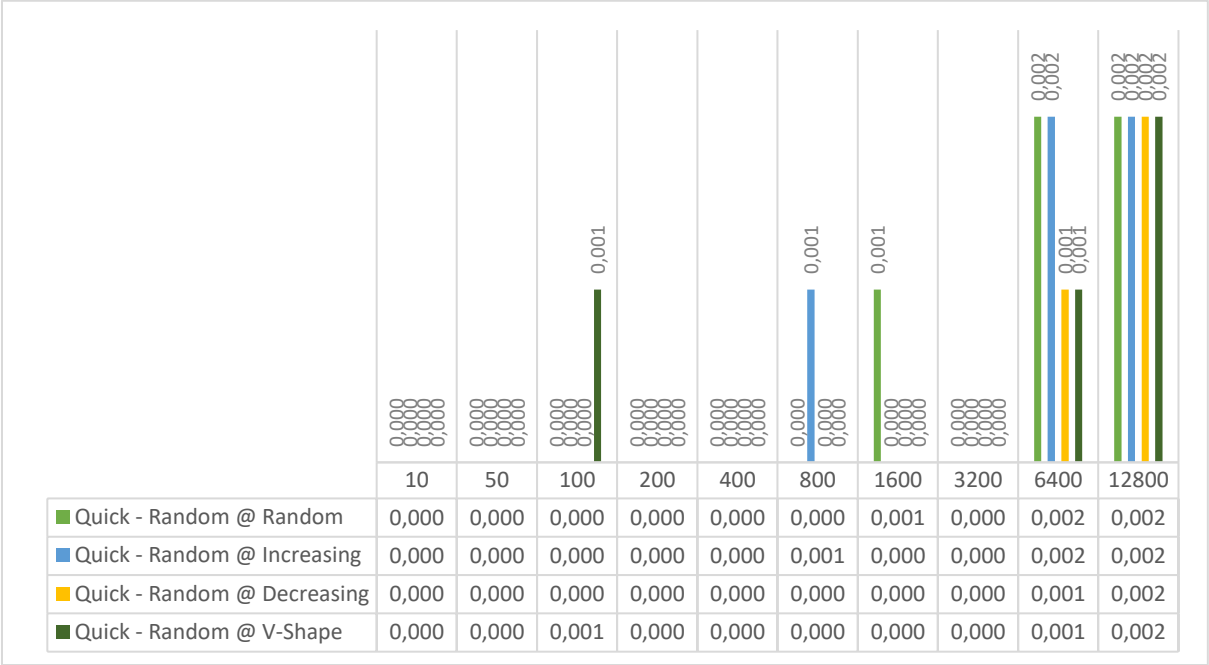
Wykres 3. Wykres przedstawiający dane zawarte z tabeli wyżej

Bardzo istotnym dla tego algorytmu jest odpowiednie dobranie elementu rozdzielającego - Quick Sort, przyjmując, że to ostatni element jest rozdzielający, w każdym wypadku (poza losowym) okazuje się być algorytmem **najwolniejszym**, zwłaszcza w przypadku rosnącym – dla maksymalnej próbki posortowanie zajęło mu ponad 700 milisekund, sugerując tym samym, że złożoność algorytmu w tym przypadku (pesymistyczny - tablica już posortowana, stosując QS z ostatnim elementem jako rozdzielającym) również będzie wysoka – $O(n^2)$.

5.2. Wariant z losową wartością jako rozdzielającą

	Random	Increasing	Decreasing	V-Shape
10	0,000	0,000	0,000	0,000
50	0,000	0,000	0,000	0,000
100	0,000	0,000	0,000	0,001
200	0,000	0,000	0,000	0,000
400	0,000	0,000	0,000	0,000
800	0,000	0,001	0,000	0,000
1600	0,001	0,000	0,000	0,000
3200	0,000	0,000	0,000	0,000
6400	0,002	0,002	0,001	0,001
12800	0,002	0,002	0,002	0,002

Tabela 4. Czasy wykonania algorytmu Quick Sort z losowym elementem rozdzielającym



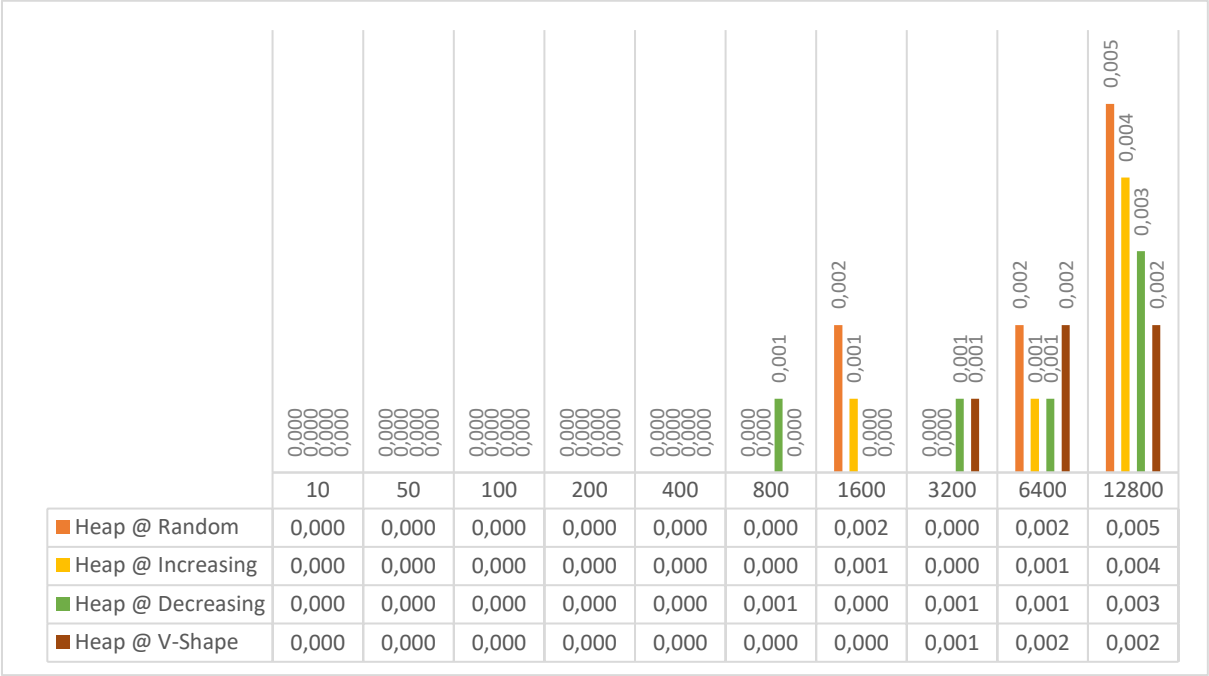
Wykres 4. Wykres przedstawiający dane zawarte z tabeli wyżej

Przyjmując natomiast element losowy jako rozdzielający, Quick Sort okazuje się faktycznie być metodą najszybszą, niezależnie od instancji lub wielkości próbki – średni czas działania algorytmu na poziomie 2ms czyni go, kontrastowo z wariantem z elementem ostatnim jako rozdzielającym, najszybszym względem pozostałych algorytmów.

6. Algorytm Heap Sort

	Random	Increasing	Decreasing	V-Shape
10	0,000	0,000	0,000	0,000
50	0,000	0,000	0,000	0,000
100	0,000	0,000	0,000	0,000
200	0,000	0,000	0,000	0,000
400	0,001	0,000	0,000	0,000
800	0,000	0,000	0,001	0,000
1600	0,002	0,001	0,000	0,000
3200	0,000	0,000	0,001	0,001
6400	0,002	0,001	0,001	0,002
12800	0,005	0,004	0,003	0,002

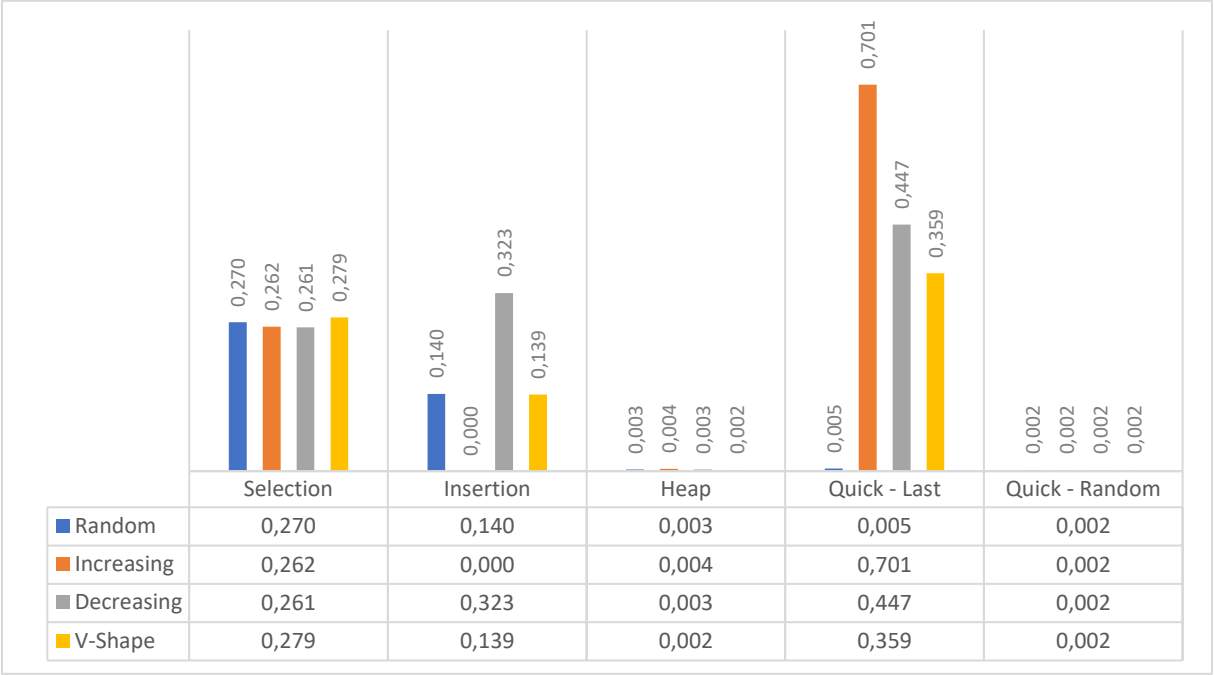
Tabela 5. Czasy wykonania algorytmu Heap Sort



Wykres 5. Wykres przedstawiający dane zawarte z tabeli wyżej

Algorytm Heap Sort również należy do kategorii szybszych algorytmów sortowania. Wykorzystuje on strukturę kopca w celu posortowania danych, stopniowo układając dane w tablicy jako kolejne elementy kopca, a następnie je sortując. Mimo tego, że jest nieco wolniejszy od Quick Sort, ma względem niego bardzo znaczną zaletę – złożoność. Nawet w najgorszym wypadku (wartości wygenerowane losowo) Heap Sort ma złożoność na poziomie **O(n log n)**, znacznie niższą niż najgorszy przypadek algorytmu od niego teoretycznie szybszego.

7. Porównanie czasu wykonania algorytmów dla największej próbki



	Selection	Insertion	Heap	Quick - Last	Quick - Random
Random	0,270	0,140	0,003	0,005	0,002
Increasing	0,262	0,000	0,004	0,701	0,002
Decreasing	0,261	0,323	0,003	0,447	0,002
V-Shape	0,279	0,139	0,002	0,359	0,002

Wykres 6. Porównanie czasów działania pięciu algorytmów dla każdej instancji przy próbce 12800

Na powyższym wykresie widać, w jakim czasie zostały posortowane tablice danych przy użyciu różnych algorytmów. Najszybszym algorytmem, jak widać, został algorytm Quick Sort, uzyskując około dwóch milisekund czasu wykonania dla największej próbki przy losowym elemencie rozdzielającym bez zależności od instancji tabeli. W przypadku innej implementacji sytuacja była zupełnie inna – Quick Sort okazał się być najwolniejszym algorytmem (poza instancją losową), doprowadzając do tego, że jako ogół nie można go traktować jako algorytmu najlepszego. W tej roli lepiej spełnił się Heap Sort, mając niewiele gorsze czasy od Quick Sort w losowym przypadku, nie mając jednak elementu niepewności – nawet jeśli wybierzemy losowy element, jest możliwość, że dane będą posortowane, a element rozdzielający może być ostatnim. Pozostałe algorytmy (Selection oraz Insertion Sort) mają tą przewagę, że są prostsze w implementacji w samym kodzie programu i w zrozumieniu działania od pozostałych, kosztem jednak prędkości wykonania i złożoności w najgorszym wypadku.