PAMSI – testowanie algorytmów sortowania

Piotr Wilkosz

21/03/2014

1 Wstęp

Celem ćwiczenia było przetestowanie złożoności obliczeń algorytmów. Do testowania zostały wybrane 3 z puli algorytmów na ocenę bardzo dobrą:

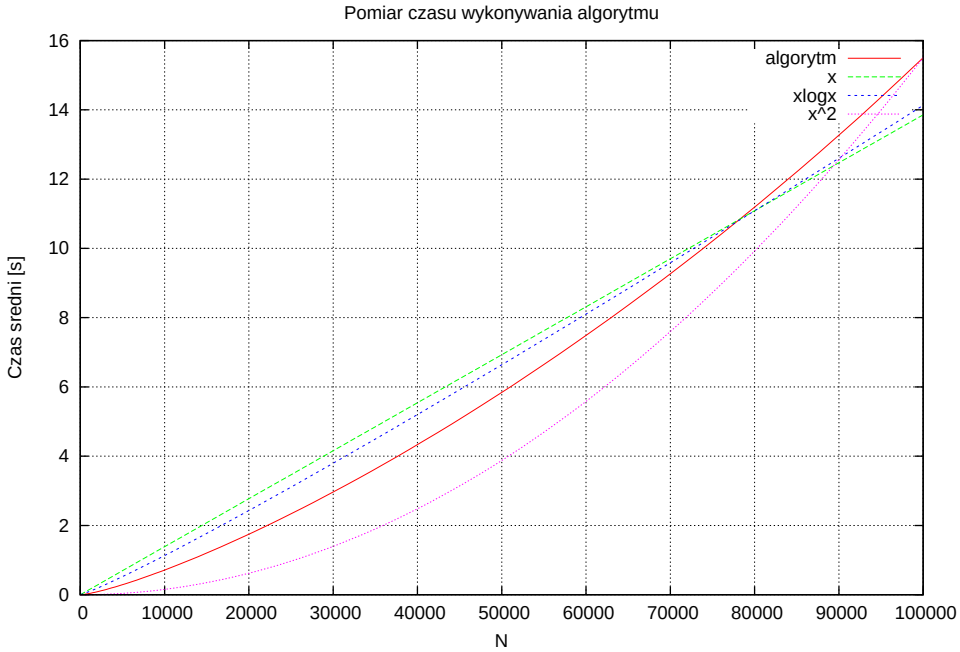
- Quick-sort (sortowanie szybkie)
- Heap-sort (sortowanie przez kopcowanie)
- Merge-sort (sortowanie przez scalanie)

2 Wyniki pomiarów

1. Quick-sort - dane malejące:

Tablica 1: Test nr 1

N	czas	odchylenie
100	2.69796e-05	5.85553e-06
1000	0.00162174	0.000156749
10000	0.159148	0.0182501
50000	3.88031	0.354078
100000	15.5052	1.40588

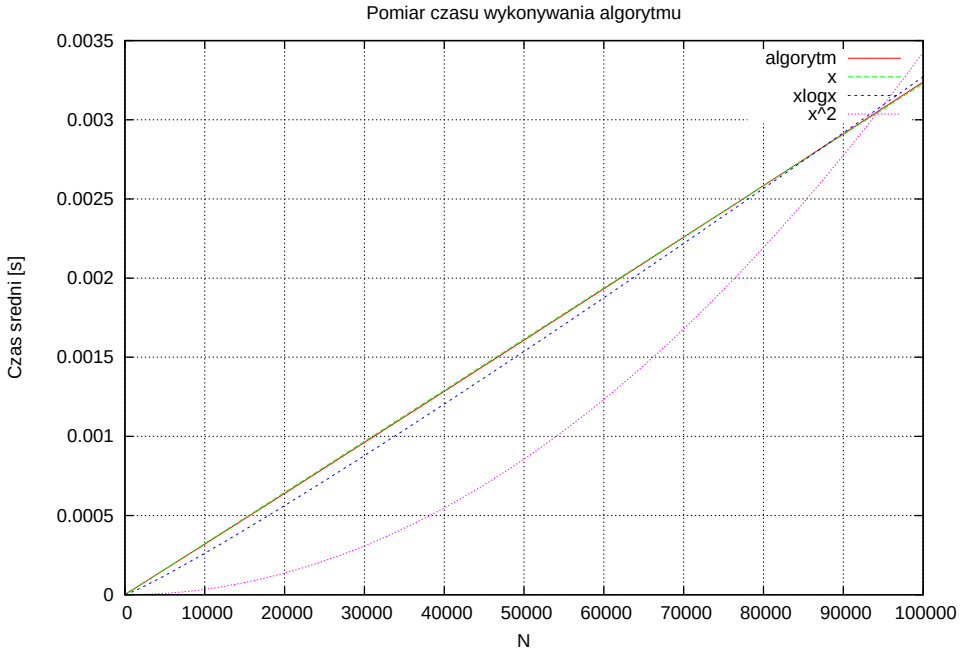


Rysunek 1: Test nr 1

2. Heap-sort - dane malejące:

Tablica 2: Test nr 2

N	czas	odchylenie
10	1.6762e-06	2.49005e-07
100	4.6864e-06	4.41747e-07
1000	3.44455e-05	6.66292e-06
10000	0.000312184	3.15754e-05
50000	0.0015983	0.00015338
100000	0.0032386	0.000345748

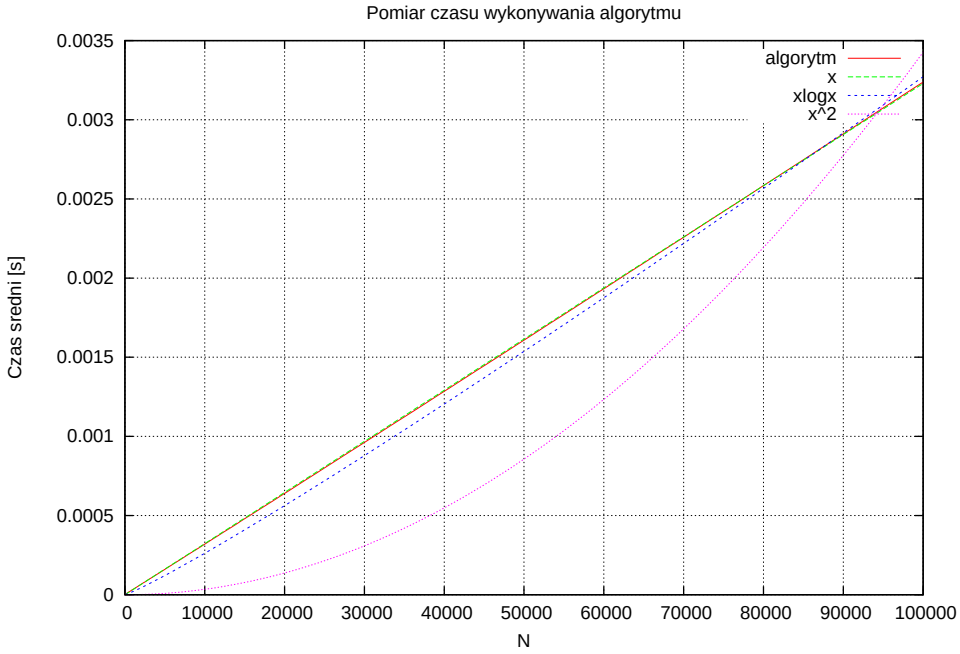


Rysunek 2: Test nr 2

3. Mergesort - dane malejące:

Tablica 3: Test nr 3

N	czas	odchylenie
10	3.0871e-06	3.79339e-07
100	4.37069e-05	4.27675e-06
1000	0.000739304	6.89142e-05
10000	0.00909738	0.000852001
50000	0.0590207	0.00537699
100000	0.128918	0.0118545

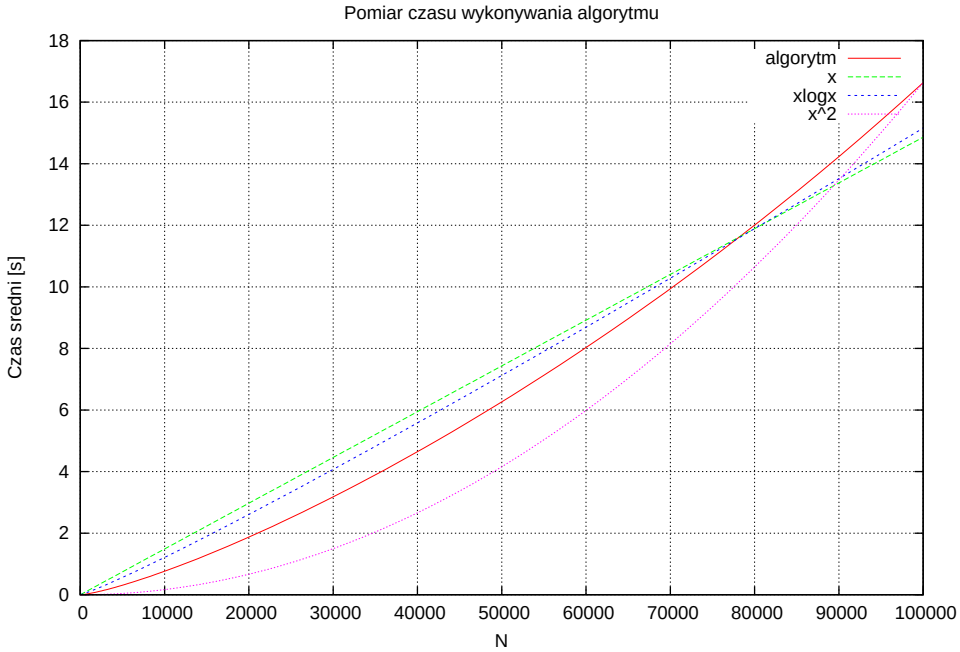


Rysunek 3: Test nr 3

4. Quick-sort - dane rosnące:

Tablica 4: Test nr 4

N	czas	odchylenie
10	2.165e-06	2.73961e-07
100	2.21539e-05	2.6968e-06
1000	0.00161351	0.000150407
10000	0.162241	0.0210872
50000	4.1648	0.404208
100000	16.6305	1.54541

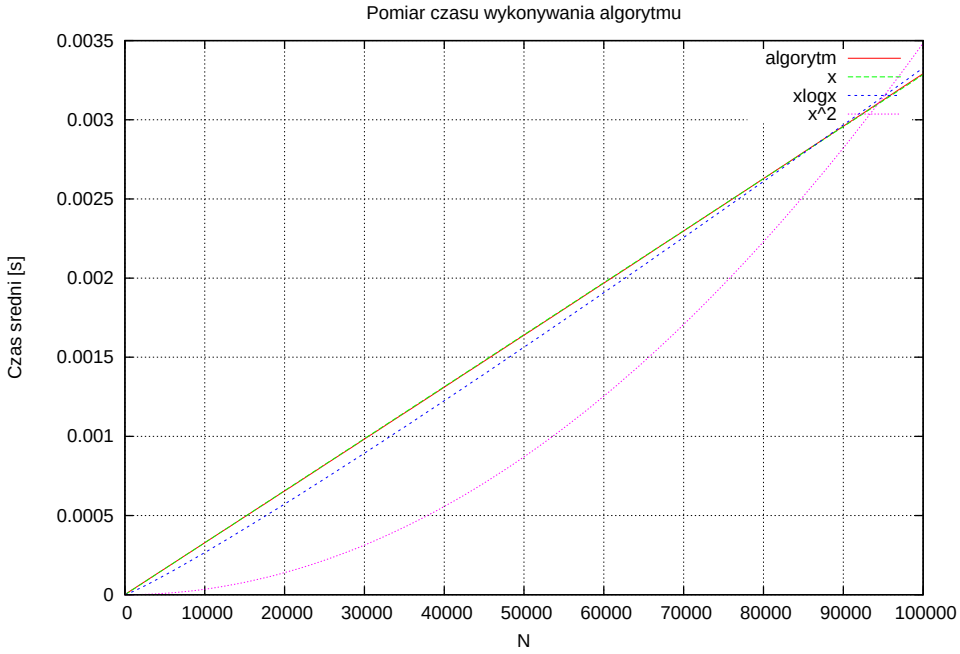


Rysunek 4: Test nr 4

5. Heap-sort - dane rosnące:

Tablica 5: Test nr 5

N	czas	odchylenie
10	3.1777e-06	9.66352e-07
100	1.41498e-05	1.37796e-06
1000	0.000129143	1.37132e-05
10000	0.00125259	0.000136966
50000	0.00610872	0.000647302
100000	0.0148653	0.00293036

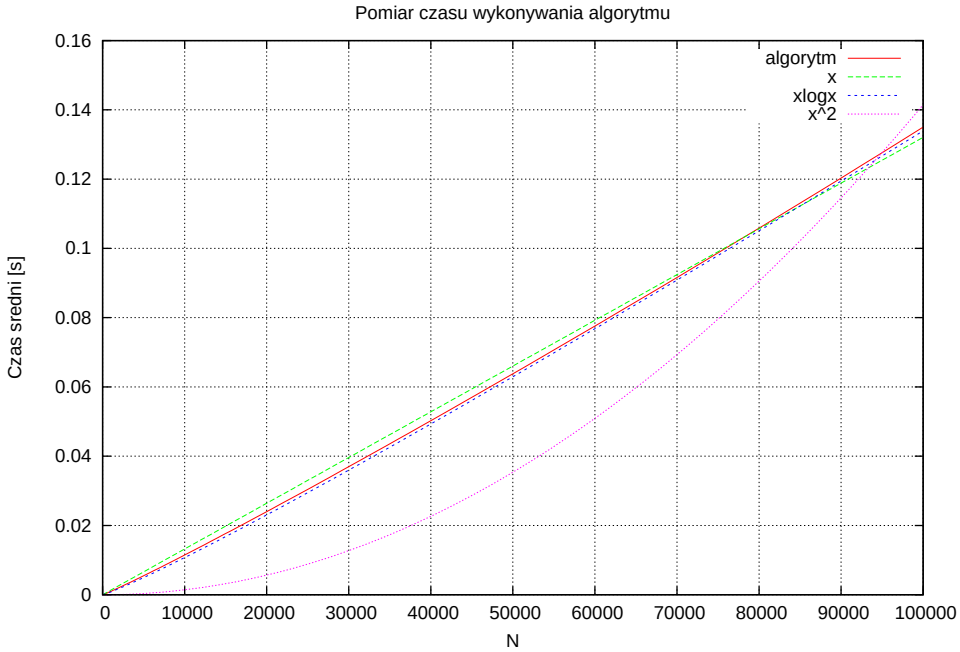


Rysunek 5: Test nr 5

6. Mergesort - dane rosnące:

Tablica 6: Test nr 6

N	czas	odchylenie
10	1.7741e-06	2.50393e-07
100	4.7632e-06	4.60575e-07
1000	3.37824e-05	3.05981e-06
10000	0.000329455	3.13672e-05
50000	0.00162924	0.000157049
100000	0.0032913	0.000301352

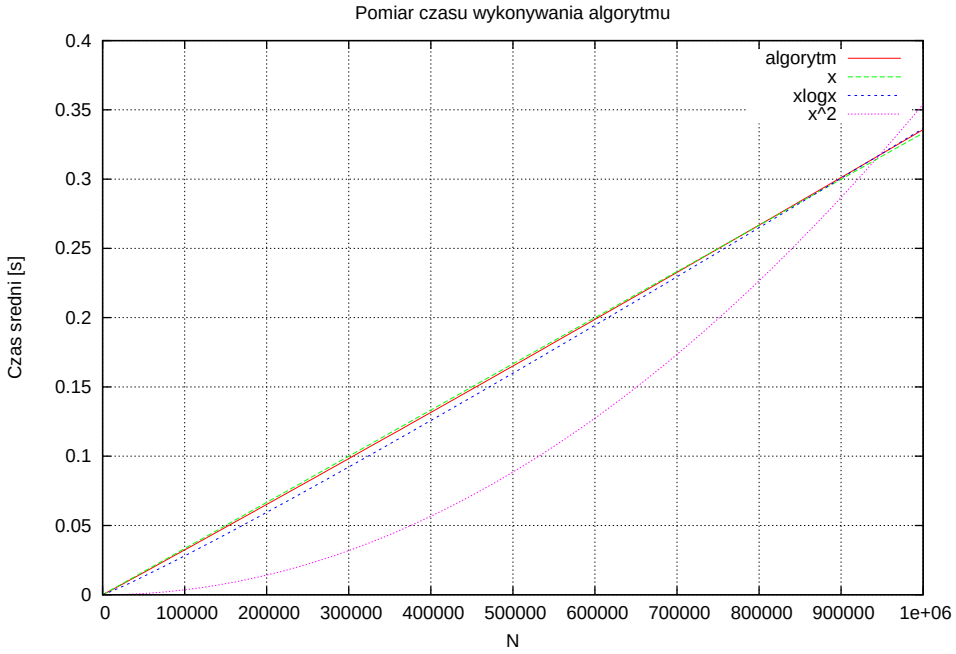


Rysunek 6: Test nr 6

7. Quick-sort - dane przypadkowe:

Tablica 7: Test nr 7

N	czas	odchylenie
10	2.2279e-06	3.13207e-07
100	1.4527e-05	1.44116e-06
1000	0.000188593	1.76021e-05
10000	0.00265908	0.000800968
500000	0.182986	0.0242359
1000000	0.340734	0.0328892

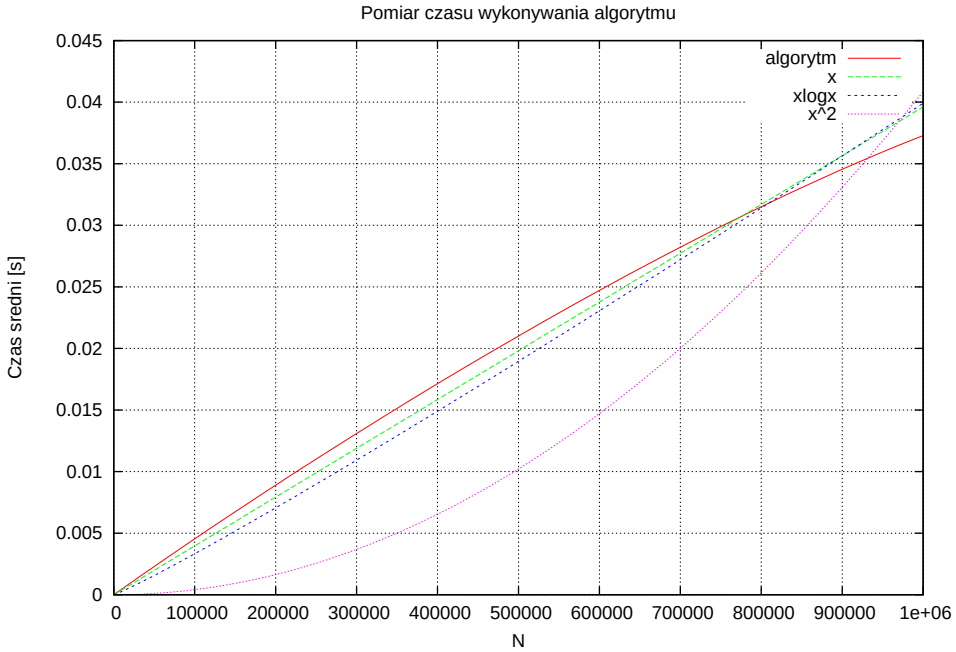


Rysunek 7: Test nr 7

8. Heap-sort - dane przypadkowe:

Tablica 8: Test nr 8

N	czas	odchylenie
10	1.8508e-06	2.27902e-07
100	4.8819e-06	4.74368e-07
1000	3.432e-05	3.1194e-06
10000	0.000334107	3.23179e-05
500000	0.0244684	0.00483988
1000000	0.0372819	0.00902984

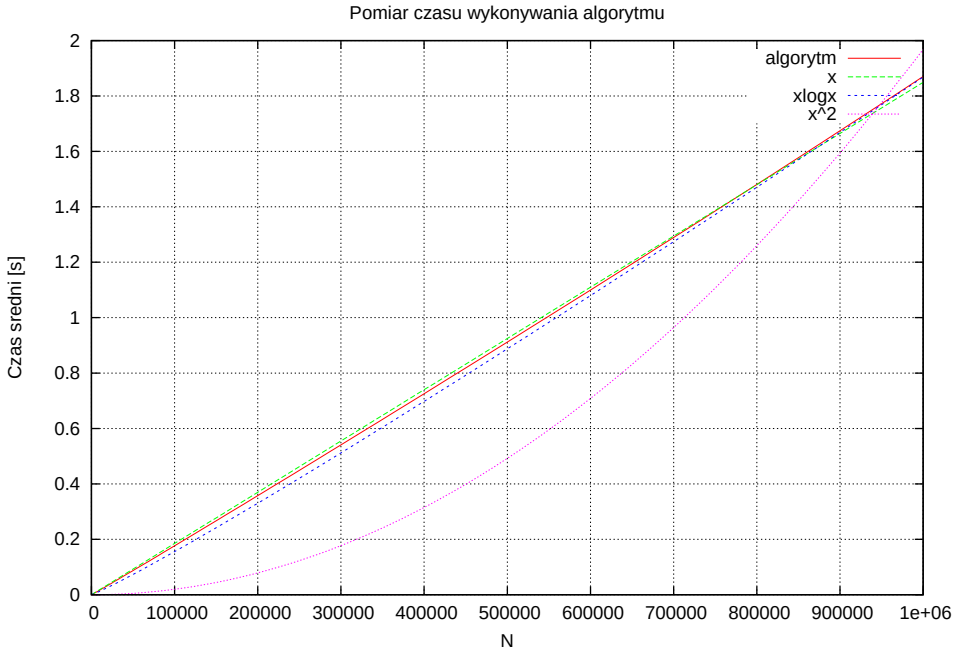


Rysunek 8: Test nr 8

9. Mergesort - dane przypadkowe:

Tablica 9: Test nr 9

N	czas	odchylenie
10	2.9822e-06	3.79038e-07
100	4.42445e-05	4.52298e-06
1000	0.000785393	8.62454e-05
10000	0.00955412	0.000909519
500000	0.883084	0.0849108
1000000	1.87052	0.166319

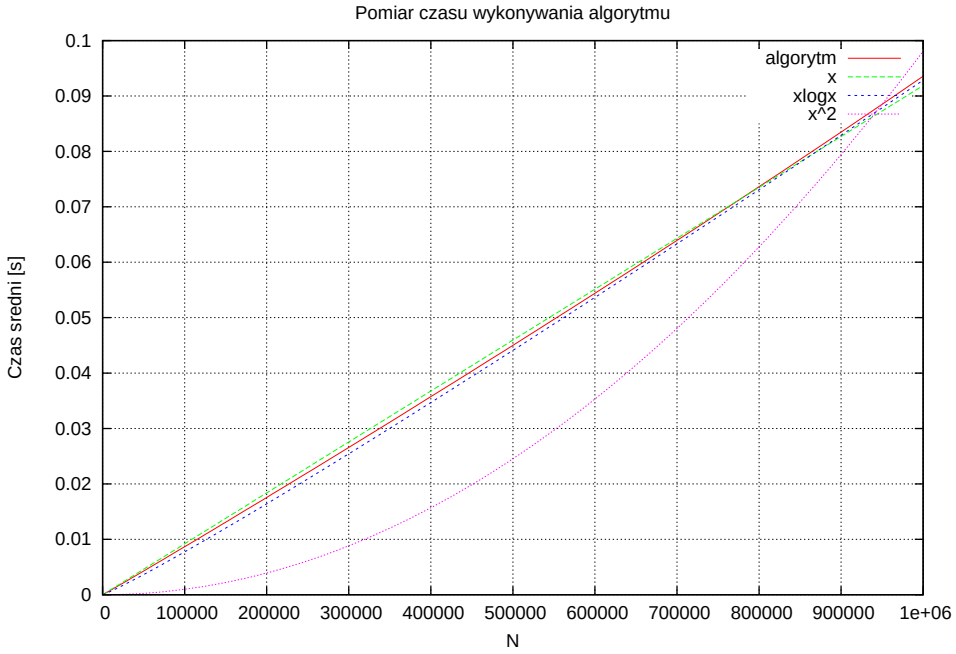


Rysunek 9: Test nr 9

10. Quick-sort - dane posortowane z usprawnieniem algorytmu (wybieranie środkowego elementu):

Tablica 10: Test nr 7

N	czas	odchylenie
10	1.8649e-06	2.82212e-07
100	6.0485e-06	5.95969e-07
1000	5.18922e-05	5.21905e-06
10000	0.000637951	5.80617e-05
500000	0.0425858	0.0045352
1000000	0.0935673	0.0140937



Rysunek 10: Test nr 7

3 Wnioski

- Sortowanie szybkie, spośród powyższych, okazało się najłatwiejsze w implementacji.
- Złożoność obliczeniową powyższych algorytmów szacuje się na $O(nlogn)$
- Algorytm sortowania szybkiego okazał się najszybszy, co nie oznacza, że jest on niezawodny. Istnieje taka możliwość, gdzie złożoność obliczeniowa algorytmu sortowania szybkiego wynosi $O(n^2)$.Dotyczy ona np. tablicy posortowanej. Gdy takie sytuacje miałyby miejsce dość często, warto wówczas usprawnić algorytm sortowania szybkiego. Jednym z rozwiazań, użytym w badaniach, był zabieg polegający na wybieraniu środkowego indeksu tablicy, jako elementu służącego do partycjonowania całości. Nie jest to rozwiązanie idealne, aczkolwiek znacząco usprawnia algorytm i chroni przed danymi posortowanymi. Inną metodą może być też poszukiwanie mediany; znajdowanie mediany całego zbioru cechuje się dużą złożonością, dlatego też wybiera się losowe elementy tablicy i to z ich mediany ustala się element rozdzielający, co zapobiega trafieniu na największy element.
- Algorytmy sortowania stogowego i sortowania przez scalanie mają gwarantowaną złozoność niezależnie od danych wejściowych, więc są dobrą alternatywą dla algorytmu quicksort.