Algorytmy sortowania

Kacper Leporowski
Adam Malinowski

W tym sprawozdaniu porównane będą wybrane algorytmy sortowania pod względem ich złożoności, czasu wykonania I wpływu różnych danych wejściowych na te kryteria – danych, dla których algorytm będzie wykonywał się dłużej lub krócej w porównaniu do danych losowych.
Rodzaje danych, które zostały sprawdzone, to dane losowe, dane rosnące, dane malejące, dane A-kształtne I dane stałe. Zestawy liczb przeznaczonych do posortowania miały rozmiary od 1000 do 10000 elementów z krokiem 1000, a w przypadkach, w których taka liczba danych nie pozwalała na miarodajny pomiar czasu wykonania wykorzystano listy o wymiarach od 50000 to 500000 elementów, z krokiem 50000.

Wykorzystane algorytmy sortowania to:

- sortowanie przez wybieranie (selection sort)
- sortowanie przez wstawianie (insertion sort)
- sortowanie Shella (Shell sort)
- sortowanie przez kopcowanie (heap sort)
- sortowanie szybkie (quick sort)

Sortowanie przez wybieranie

Sortowanie przez wybieranie to algorytm sortowania w miejscu o złożoności obliczeniowej $O(n^2)$ – w programie występują dwie zagnieżdżone pętle.

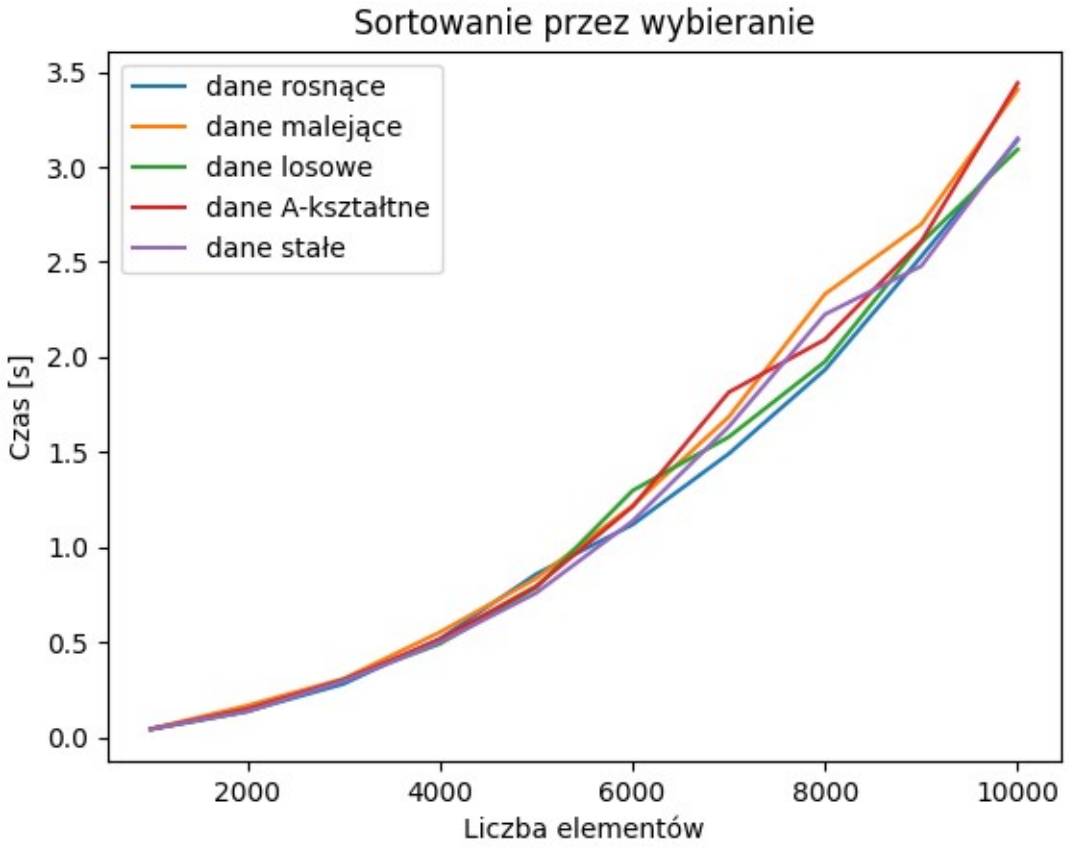

Figura 1: Czas wykonywania - selection sort

Złożoność I czas wykonania algorytmu są niezależne od rodzaju danych wejściowych – dla każdego zbioru algorytm wykonuje się w tym samym czasie.

Sortowanie przez wstawianie

Sortowanie przez wstawianie ma złożoność $O(n^2)$ i sortuje w miejscu.

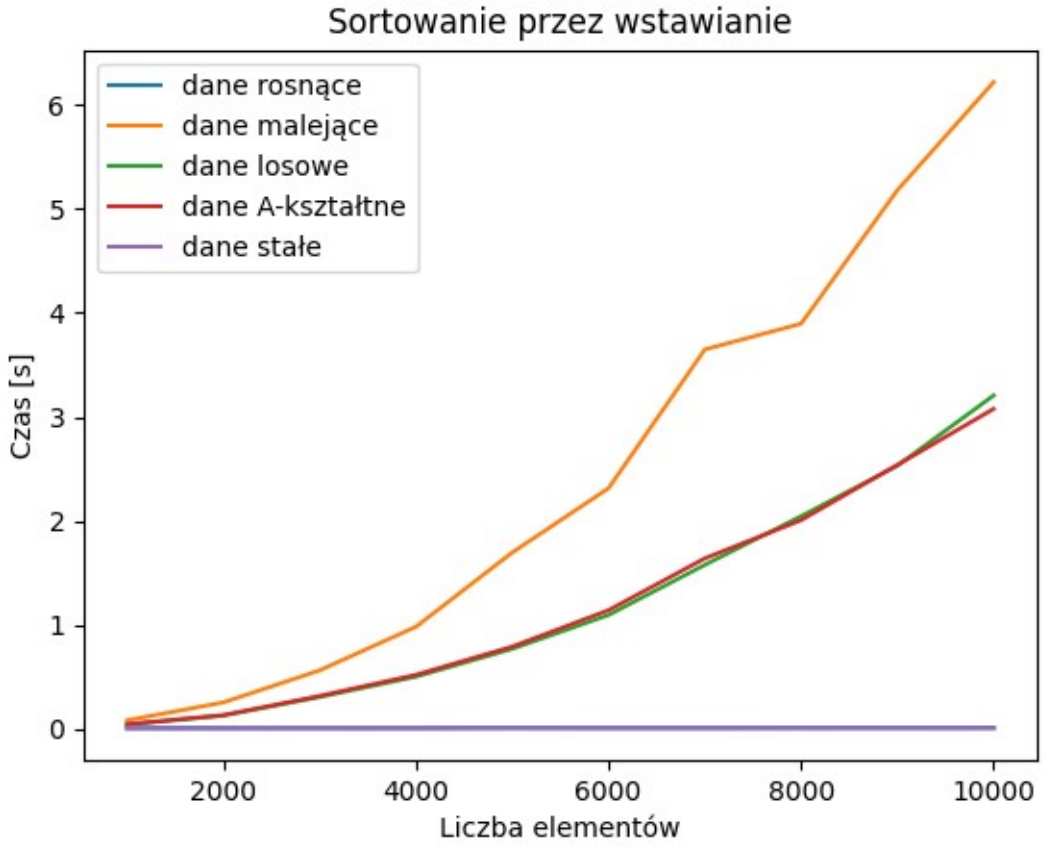

Figura 2: Czas wykonywania insertion sort

Dla danych, które są już wstępnie posortowane (dane stałe, dane rosnące) następuje przypadek optymistyczny, w którym algorytm wykonuje się w czasie liniowym. W pozostałych przypadkach czas jest kwadratowy, a algorytm najgorzej radzi sobie z danymi malejącymi.

Sortowanie Shella

Sortowanie Shella jest algorytmem sortującym w miejscu. Złożoność algorytmu to O(n log n) (w najgorszym wypadku $O(n^2)$). Rozkład danych wejściowych nie wpływa znacząco na działanie, jedynie w przypadku wprowadzenia danych z góry posortowanych złożoność wynosi O(n).

Złożoność wynika z faktu, że jest to usprawniona wersja insertion sort, która najpierw sortuje bardziej oddalone od siebie liczby, a potem przechodzi do mniej oddalonych. Złożoność zależy od „odległości” między liczbami jakie wybierzemy.

W tym przypadku początkowy przyrost to połowa długości zestawu danych który po każdym przejściu jest dzielony przez 2.

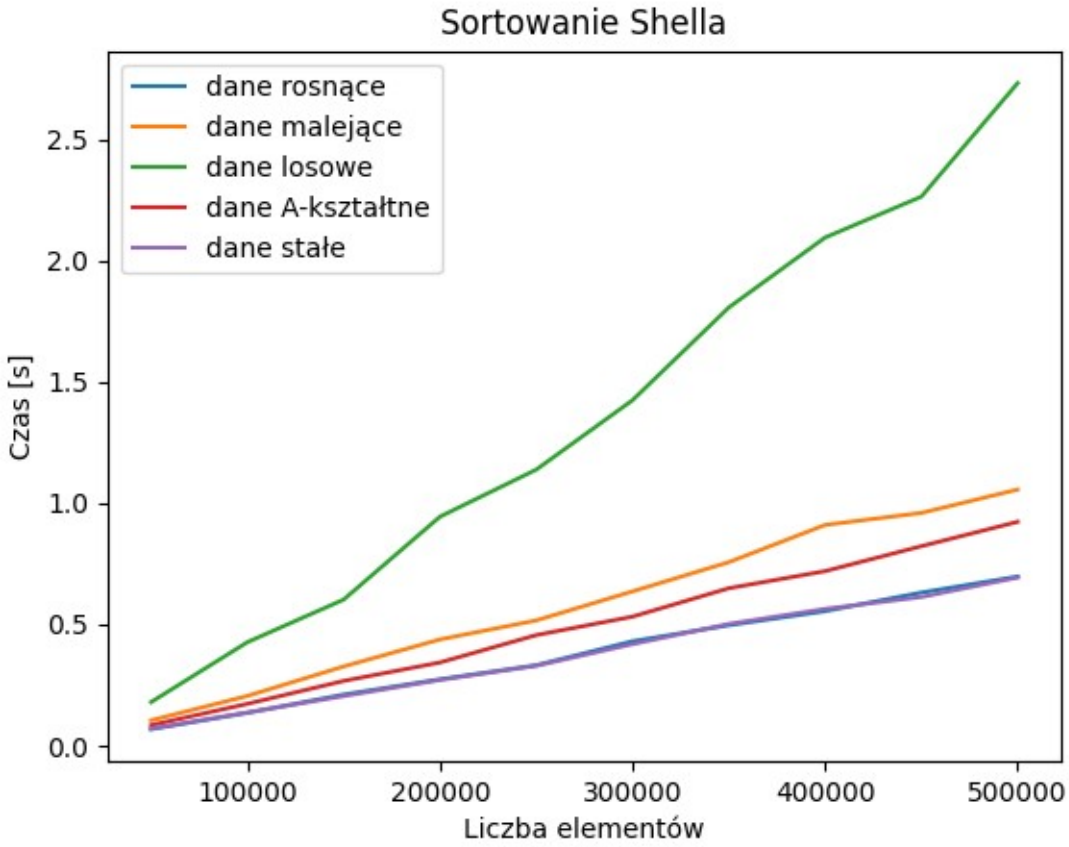

Figura 3: Czas wykonywania Shell sort

Sortowanie przez kopcowanie

Sortowanie przez kopcowanie jest algorytmem sortującym w miejscu. Złożoność to O(n log n). Rozkład danych nie wpływa znacząco na działanie za wyjątkiem danych stałych – w tym przypadku algorytm wykonuje się znacznie szybciej (O(n)). Złożoność wynika z faktu, że układanie danych w „kopce” ma złożoność O(log n), a późniejsze sortowanie ich złożoność O(n).

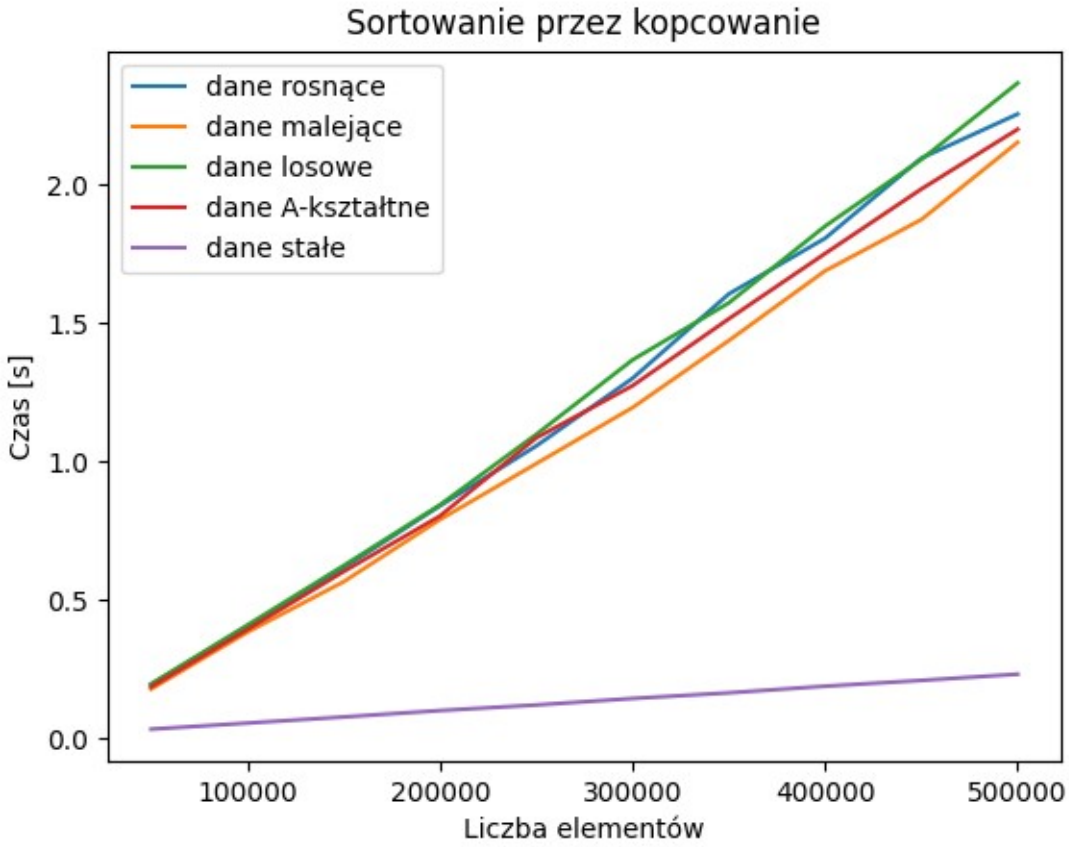

Figura 4: Czas wykonywania heap sort

Sortowanie szybkie

Sortowanie szybkie sortuje w miejscu. Złożoność to O(n log n) (w najgorszym przypadku $O(n^2)$). Rozkład danych nie wpływa znacząco na działanie.
Złożoność wynika z ilości rekurencyjnych wywołań funkcji i tego, który element wybierzemy jako klucz (w najlepszym przypadku byłby to element średni).

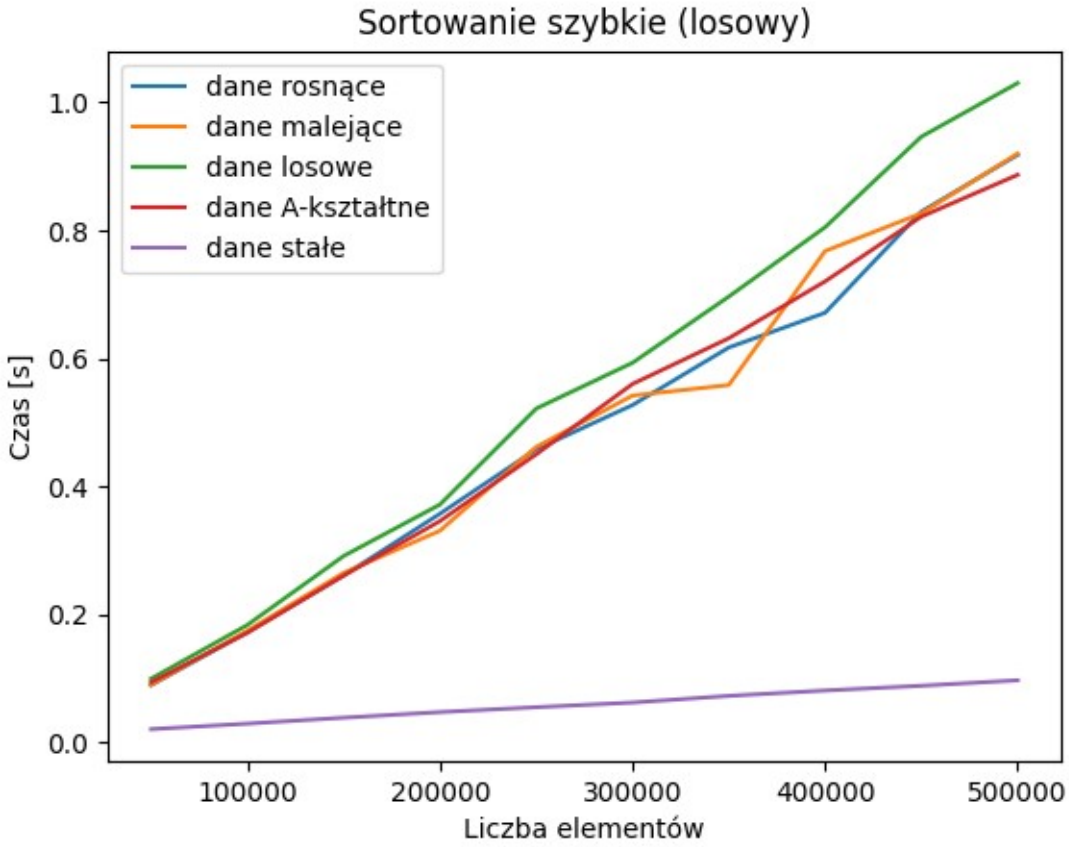

Figura 5: Czas wykonywania quick sort

W przypadku losowego wyboru klucza algorytm wykonuje się w tym samym czasie dla wszystkich zestawów poza danymi stałymi, dla których wykonuje się znacznie szybciej. W przypadku, gdy kluczem jest pierwszy element w liście sytuacja zmienia się – dla danych losowych algorytm nadal wykonuje się w tym samym czasie, jednak dla pozostałych danych program nie wykonuje się – spowodowane jest to tym, że w tych przypadkach kluczem staje się element najmniejszy bądź największy w danej liście – następuje przepełnienie stosu z powodu zbyt wielu wywołań rekurencyjnych.