AZO - zadanie projektowe nr 1

Badanie efektywności wybranych algorytmów sortowania ze względu na złożoność obliczeniową

Jakub Grzegocki (264009)

kwiecień, 2024

Spis treści

1 Wprowadzenie

Niniejsze sprawozdanie skupia się na analizie wydajności czterech różnych algorytmów sortowania: sortowania przez wstawianie, szybkiego sortowania (quick sort), sortowania kopcowego (heap sort) oraz sortowania Shella. Celem jest porównanie czasu działania tych algorytmów w zależności od różnych warunków początkowych oraz rozmiarów zbioru danych. Każdy algorytm ma różne cechy złożoności obliczeniowej, które są uzależnione od specyficznych warunków, takich jak początkowe uporządkowanie danych. Sortowanie przez wstawianie jest algorytmem prostym, o średniej i najgorszej złożoności obliczeniowej wynoszącej $O(n^2)$, gdzie n to liczba elementów do posortowania. Algorytm ten jest skuteczny dla małych lub częściowo posortowanych zbiorów danych, lecz jego efektywność maleje przy zwiększaniu się rozmiaru danych. Szybkie sortowanie to algorytm typu dziel i zwyciężaj, który zazwyczaj osiąga złożoność obliczeniową $O(nlogn)$. W najgorszym scenariuszu, gdy dane są już uporządkowane lub zawierają wiele powtarzających się elementów, jego złożoność może wzrosnąć do $O(n^2)$. Efektywność szybkiego sortowania jest zależna od wyboru punktu odniesienia (pivota). Sortowanie kopcowe, wykorzystujące strukturę kopca, zapewnia złożoność $O(nlogn)$ w każdych warunkach. Jest to efekt wykorzystania operacji przywracania własności kopca do efektywnego porządkowania elementów. Sortowanie Shella jest zaawansowaną wersją sortowania przez wstawianie i pozwala na porównania oraz wymiany elementów oddalonych od siebie o określoną wartość, zwaną 'przerwą'. Jego złożoność obliczeniowa jest trudna do jednoznacznego określenia i, zależnie od wybranych przerw, może wynosić od $O(nlog^2n)$ do $O(n^{3/2})$ w średnim przypadku. W ramach badania przeprowadzono testy tych algorytmów na danych o różnej wielkości i różnym stopniu uporządkowania, co umożliwiło głębsze zrozumienie ich zachowania i ograniczeń.

2 Plan eksperymentu

Eksperyment miał na celu ocenę wydajności wybranych algorytmów sortowania przez analizę ich czasu wykonania w zależności od różnorodnych warunków początkowych oraz rozmiaru sortowanych zbiorów danych. Główne założenia eksperymentu przedstawiono poniżej:

- **Badane algorytmy sortowania:** w badaniu uwzględniono sortowanie przez wstawianie, szybkie sortowanie (quick sort), sortowanie kopcowe (heap sort) oraz sortowanie Shella.
- **Rozmiary tablic:** wybrane zostały siedem reprezentatywnych rozmiarów danych, zakres od 10 000 do 20 480 000 elementów, co pozwala na obserwację zmiany wydajności algorytmów w szerokim zakresie rozmiarów danych.
- **Generowanie danych:** dla każdego rozmiaru danych generowano zestawy w pięciu scenariuszach: dane losowe, dane posortowane rosnąco, dane posortowane malejąco, dane częściowo posortowane (33% oraz 66% początkowych elementów posortowanych). To umożliwiło ocenę algorytmów w różnych potencjalnych warunkach rzeczywistego zastosowania.
- **Metodologia pomiaru czasu:** każdy pomiar czasu sortowania był dokonywany wielokrotnie, a uzyskane wyniki były uśredniane w celu zmniejszenia wpływu zmiennych środowiskowych i losowości na wynik końcowy.
- **Narzędzia pomiarowe:** wykorzystano funkcje biblioteki `<chrono>` dostępne w języku C++ do dokładnego mierzenia czasu wykonania algorytmów w milisekundach.

- **Zapewnienie powtarzalności eksperymentu:** przed każdym pomiarem stosowano tę samą procedurę generowania danych, a przed sortowaniem dane były przekształcane do tego samego formatu tablicy dynamicznej.
- **Optymalizacje:** wszystkie algorytmy były implementowane w ten sam sposób, bez dodatkowych optymalizacji poza wyborem metody pivot dla szybkiego sortowania, co pozwalało na uczciwe porównanie wydajności algorytmów.

3 Omówienie przebiegu eksperymentów

W ramach eksperymentu porównawczego dla różnych algorytmów sortowania opracowano szczegółową metodologię i wykorzystano odpowiednie narzędzia. Poniżej przedstawiono szczegółowy opis przebiegu eksperymentów.

3.1 Metodologia badań

Do przetwarzania danych eksperymentów wybrano język programowania Python, który umożliwił za pomocą biblioteki Matplotlib realizację wykresów.

3.2 Środowisko testowe

Eksperymenty przeprowadzono na komputerze z systemem operacyjnym Linux Ubuntu 22.04 LTS, który był wyposażony w procesor AMD Ryzen 5000 oraz 32 GB pamięci RAM. Do implementacji algorytmów użyto środowiska programistycznego Clion od JetBrains. Dzięki odpowiedniemu środowisku testowemu oraz narzędziom programistycznym możliwe było przeprowadzenie eksperymentów w sposób precyzyjny i powtarzalny, co pozwoliło na uzyskanie wiarygodnych wyników i wniosków.

4 Wyniki pomiarów

4.1 HeapSort

Liczba elementów tablicy	**20000**	**40000**	**80000**	**160000**	**320000**	**640000**
Czas dla tablicy: wypełnionej losowo	1 ms	2.6 ms	6 ms	14.1 ms	29.9 ms	68.9 ms
posortowanej rosnąco	1.1 ms	2.3 ms	4.2 ms	8.9 ms	18.3 ms	40.5 ms
posortowanej malejąco	0.3 ms	2.1 ms	4.3 ms	9 ms	20.1 ms	48.5 ms
posortowanej w 33%	1.7 ms	4.1 ms	8.7 ms	18.3 ms	38.2 ms	78.3 ms
posortowanej w 66%	1 ms	2.9 ms	5.8 ms	12.4 ms	33.5 ms	63 ms

Tabela 1: Wyniki sortowania dla Sortowania przez kopcowanie

4.2 QuickSort

Liczba elementów tablicy	**80000**	**160000**	**320000**	**640000**	**1280000**	**2560000**	**5120000**
Czas dla tablicy: wypełnionej losowo	3.1 ms	8 ms	17 ms	37.2 ms	80.4 ms	172.1 ms	354.9 ms
posortowanej rosnąco	1.7 ms	2.1 ms	4.2 ms	9 ms	19.6 ms	41.7 ms	85.7 ms
posortowanej malejąco	1.3 ms	2.6 ms	6.2 ms	12.5 ms	25.7 ms	53.4 ms	109.2 ms
posortowanej w 33%	3 ms	6.8 ms	16.3 ms	35.3 ms	74.5 ms	154.5 ms	313.5 ms
posortowanej w 66%	2.2 ms	5.9 ms	13.6 ms	28.3 ms	55.7 ms	126.7 ms	265.5 ms

Tabela 2: Wyniki sortowania szybkiego z pivotem środkowym

4.3 ShellSort

Liczba elementów tablicy	**320000**	**640000**	**1280000**	**2560000**	**5120000**	**10240000**	**20480000**
Czas dla tablicy: wypełnionej losowo	31.8 ms	74 ms	182 ms	421.8 ms	870 ms	2070 ms	4992 ms
posortowanej rosnąco	3 ms	7.8 ms	17 ms	38.8 ms	81.6 ms	174 ms	361.2 ms
posortowanej malejąco	6 ms	12 ms	25.6 ms	56.2 ms	143 ms	260.4 ms	547.6 ms
posortowanej w 33%	33.4 ms	73 ms	172.2 ms	384.8 ms	873.8 ms	1930.4 ms	5921.6 ms
posortowanej w 66%	32.2 ms	73.4 ms	165.7 ms	383.2 ms	837.6 ms	1897.8 ms	4430 ms

Tabela 3: Wyniki sortowania Shella

4.4 InsertionSort

4.4.1 typ danych integer

Liczba elementów tablicy	**10000**	**20000**	**40000**	**80000**	**160000**	**320000**	**640000**
Czas dla tablicy wypełnionej losowo	8 ms	35.4 ms	142.4 ms	591.4 ms	2420.8 ms	10052.2 ms	41316.6 ms
posortowanej rosnąco	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms
posortowanej malejąco	19 ms	77 ms	317.4 ms	1232 ms	5870.4 ms	20139 ms	98408.8 ms
posortowanej w 33%	8.2 ms	35.2 ms	139.2 ms	569.8 ms	2224.8 ms	8948.8 ms	37259.4 ms
posortowanej w 66%	5 ms	23.2 ms	87.8 ms	358.6 ms	1416.8 ms	5661.2 ms	23042.8 ms

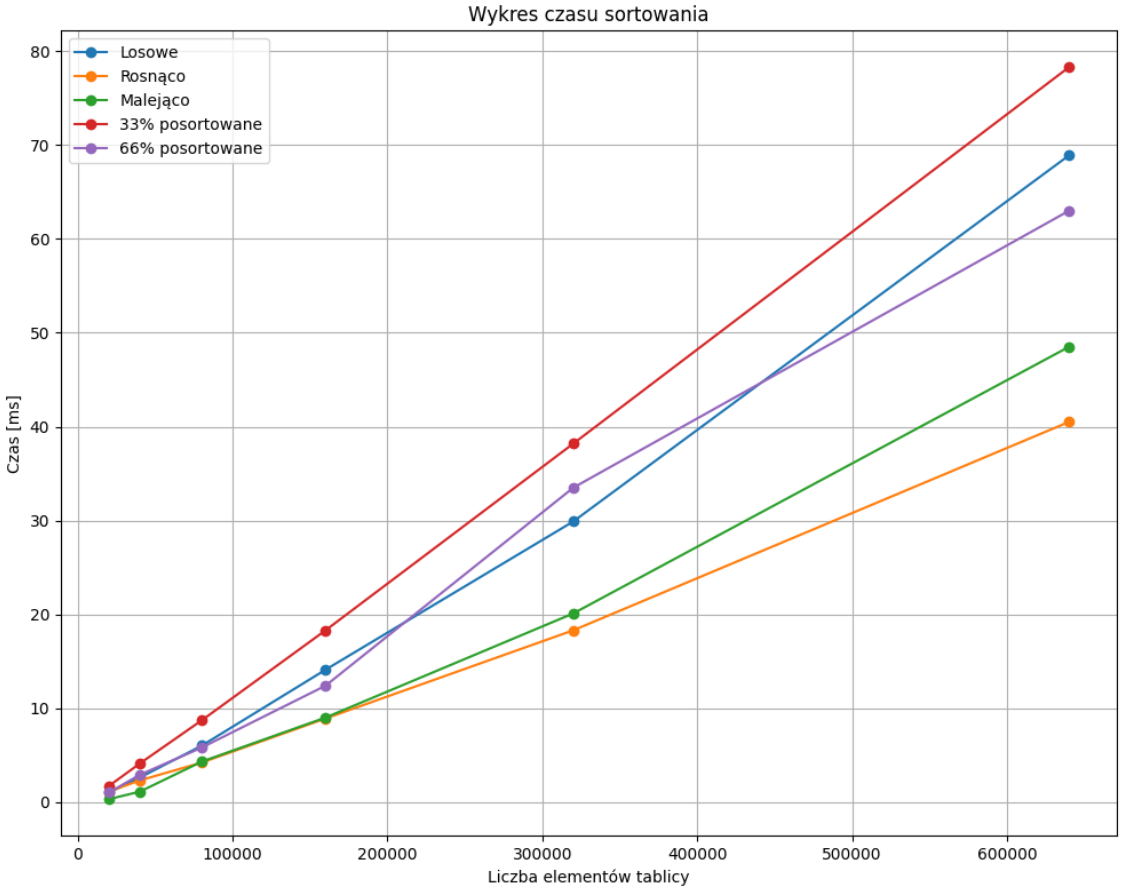
Tabela 4: Wyniki sortowania dla InsertionSort

4.4.2 typ danych float

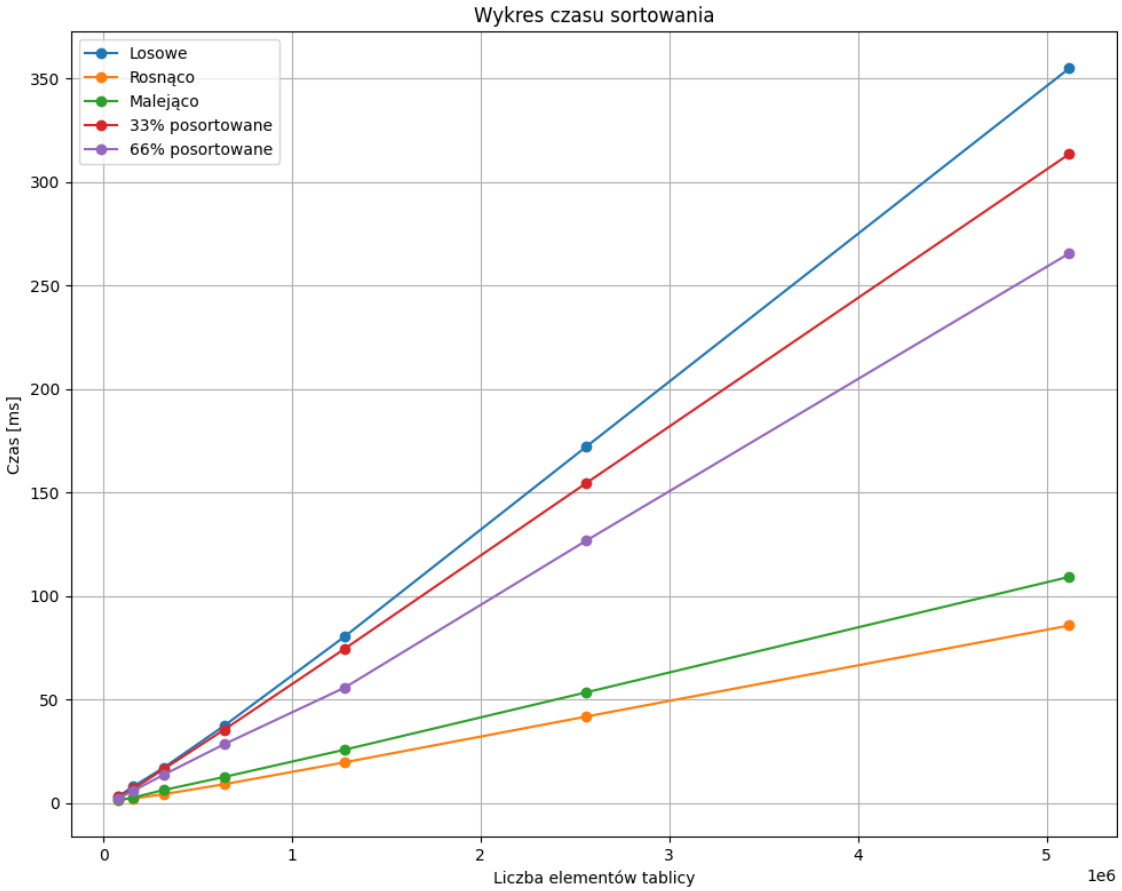
Liczba elementów tablicy	**10000**	**20000**	**40000**	**80000**	**160000**	**320000**	**640000**
Czas dla tablicy: wypełnionej losowo	9 ms	36.6 ms	155 ms	624.9 ms	2599.6 ms	10514.7 ms	42731 ms
posortowanej rosnąco	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms
posortowanej malejąco	18.5 ms	83.6 ms	406.1 ms	1384.7 ms	5172.4 ms	22228.3 ms	91194.4 ms
posortowanej w 33%	9.4 ms	37.6 ms	152 ms	618.6 ms	2439.1 ms	9985.7 ms	41016.7 ms
posortowanej w 66%	7.4 ms	23.4 ms	92.2 ms	420.6 ms	1667.1 ms	6214.1 ms	25444 ms

Tabela 5: Wyniki sortowania przez wstawianie dla różnych scenariuszy danych i rozmiarów tablicy (typ float)

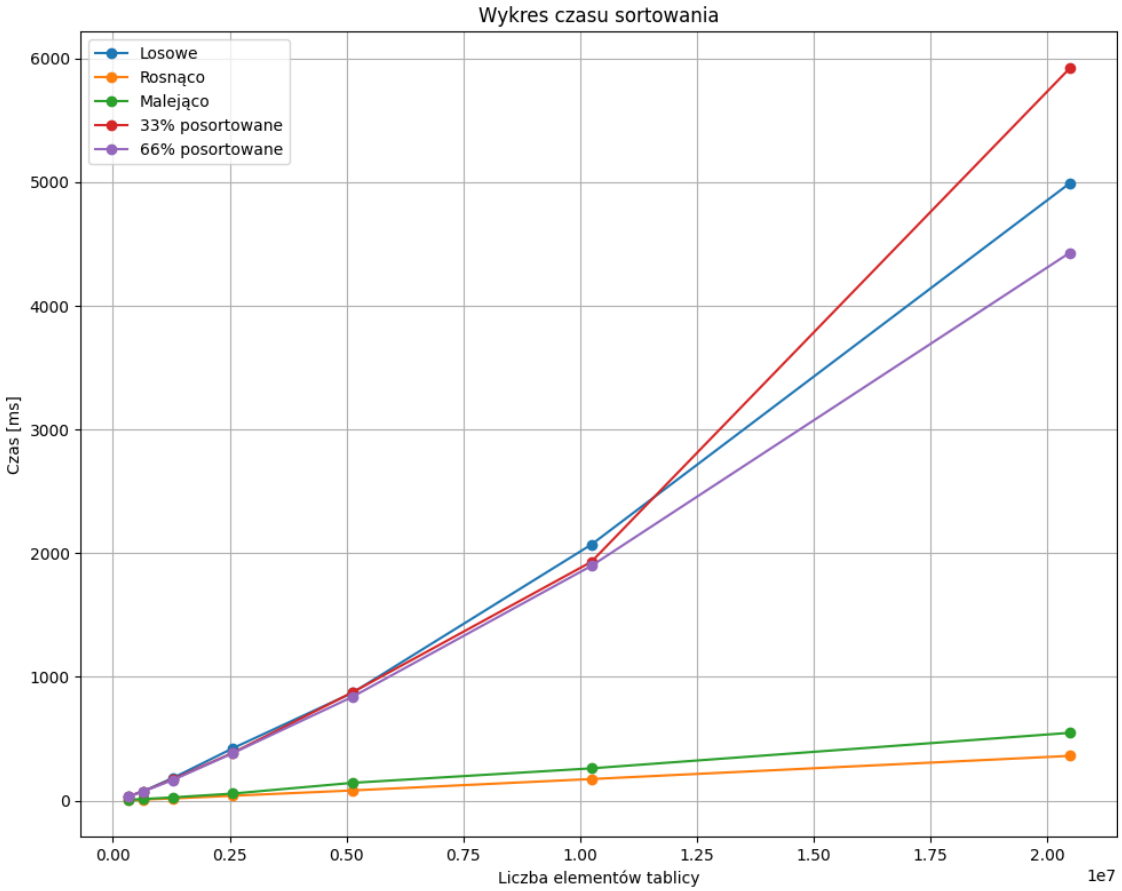
4.5 Wykresy przedstawiające zebrane dane



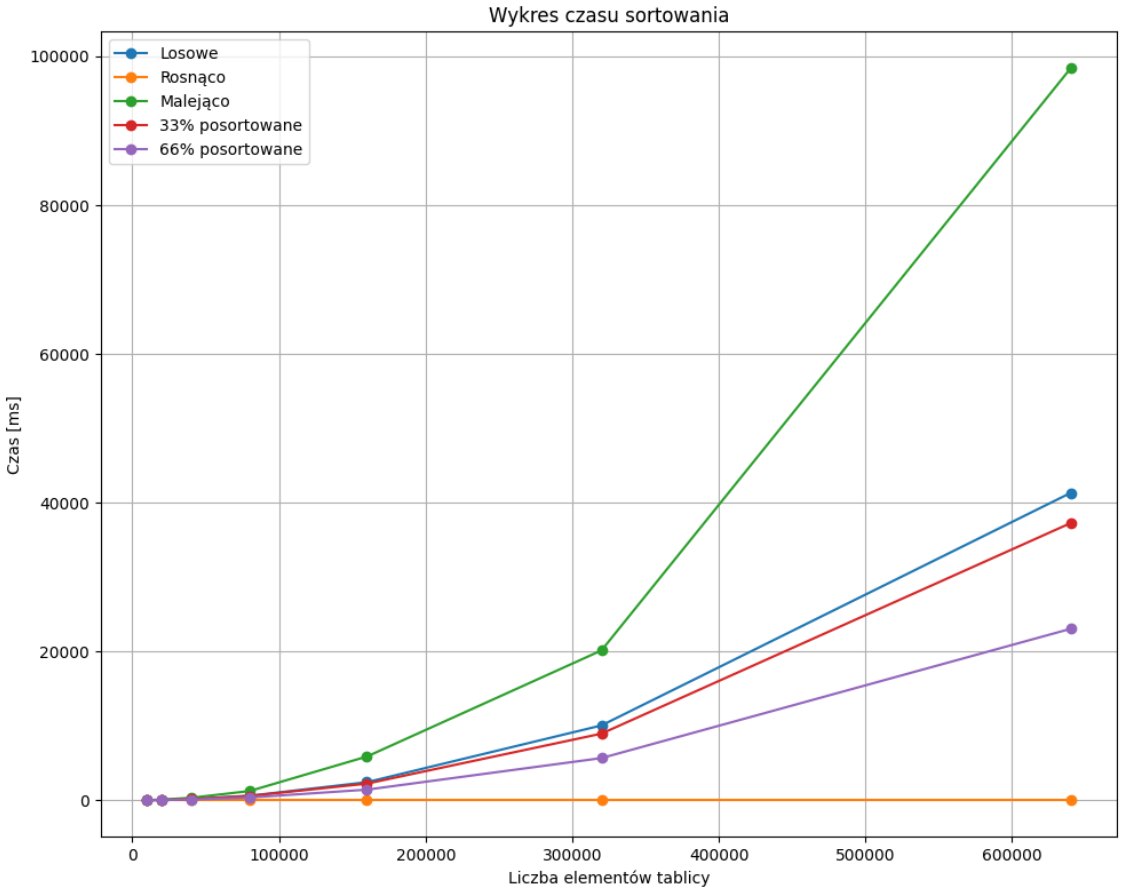
Rysunek 1: Sortowanie przez kopcowanie



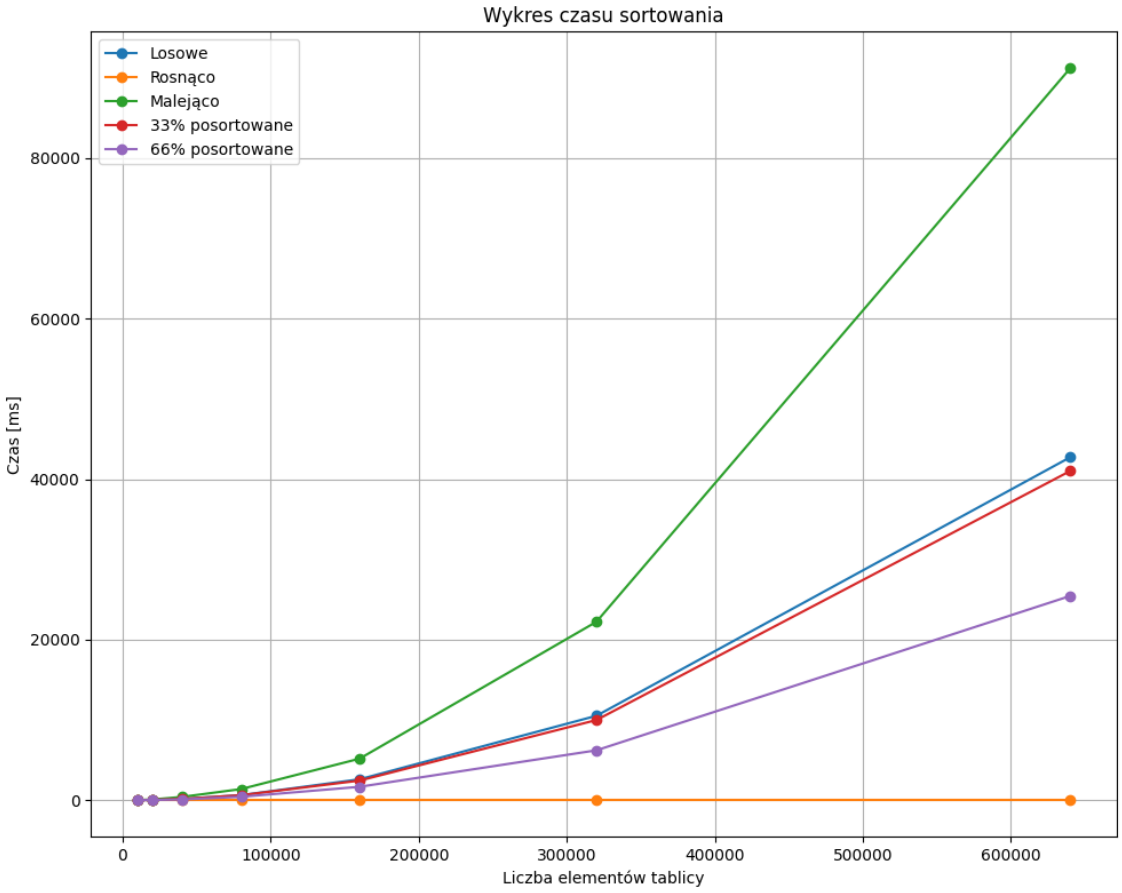
Rysunek 2: Sortowanie szybkie



Rysunek 3: sortowanie shella



Rysunek 4: sortowanie przez wstawianie



Rysunek 5: sortowanie przez wstawianie, dane float

5 Podsumowanie i wnioski

W ramach projektu przeprowadzono badania wydajności różnych algorytmów sortowania dla różnorodnych typów danych oraz w różnych scenariuszach danych. Wyniki eksperymentów dostarczyły cennych informacji o efektywności poszczególnych algorytmów w zależności od rodzaju danych oraz ich charakterystyki.

5.1 Wnioski:

- **Najlepszy algorytm dla danych losowych:** Analiza wyników wskazuje, że sortowanie szybkie okazało się najbardziej efektywnym algorytmem dla danych losowych we wszystkich badanych przypadkach. Zazwyczaj osiągało ono czas sortowania znacznie krótszy niż pozostałe algorytmy.
- **Wpływ rodzaju danych na efektywność sortowania:** Zaobserwowano istotne różnice w czasach sortowania między danymi typu 'int' a 'float'. Dla danych typu 'float', czas sortowania był z reguły dłuższy, co sugeruje mniejszą efektywność niektórych algorytmów sortowania dla liczb zmiennoprzecinkowych.
- **Wpływ charakterystyki danych na efektywność algorytmów:** Przeprowadzone badania dla różnych scenariuszy danych (losowych, rosnących, malejących, częściowo posortowanych) wykazały, że efektywność algorytmów sortowania może znacznie zależeć od charakterystyki danych wejściowych. Na przykład, algorytmy sortowania szybkiego były efektywne dla danych losowych, ale ich wydajność mogła być niższa dla danych posortowanych w określony sposób.

Podsumowując, eksperymenty dostarczyły istotnych informacji na temat efektywności różnych algorytmów sortowania w zależności od rodzaju i charakterystyki danych. Dalsze badania mogłyby się skupić na optymalizacji algorytmów pod konkretny typ danych lub na poszukiwaniu nowych, bardziej efektywnych metod sortowania dla specyficznych przypadków danych.

6 Literatura

https://www.youtube.com/watch?v=SHcPqUe2GZM
https://www.youtube.com/watch?v=JU767SDMDvA
https://www.youtube.com/watch?v=2DmK_H7IdTo
"Wprowadzenie do algorytmów" - Clifford Stein, Ron Rivest i Thomas H. Cormen