Laboratoria Algorytmów i Struktur Danych

Sprawozdanie nr 1

# Algorytmy Sortujące

Prowadzący: **Dominik Piotr Witczak**

Autorzy: **Marcin Jakubowski(164108), Adam Woźiński(164119)**

POLITECHNIKA POZNAŃSKA

# 1 | Wprowadzenie

Celem projektu było stworzenie programu umożliwiającego użytkownikowi testowanie i porównywanie różnych algorytmów sortujących. Program został napisany w języku Python i zawiera implementacje kilku popularnych algorytmów sortowania, takich jak:

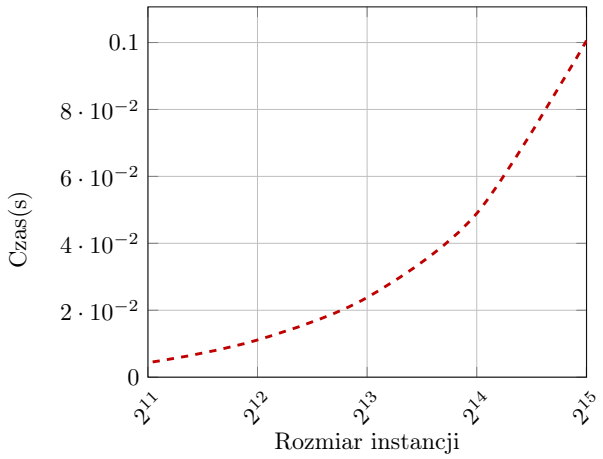
- **Insertion Sort** – sortowanie przez wstawianie,
- **Shell Sort** – sortowanie z przyrostami Sedgewicka,
- **Selection Sort** – sortowanie przez wybieranie,
- **Heap Sort** – sortowanie kopcowe,
- **Quick Sort** – sortowanie szybkie z różnymi strategiami wyboru pivota (lewy pivot i losowy pivot).

Każdy z algorytmów został zaimplementowany jako osobna funkcja, a ich działanie opiera się na kopiowaniu danych wejściowych, aby uniknąć modyfikacji oryginalnych danych. Program pozwala użytkownikowi wybrać jeden z dostępnych algorytmów sortowania i podać dane do posortowania w formie listy liczb.
Dodatkowo, program mierzy czas działania wybranego algorytmu, co pozwala na ocenę jego wydajności w zależności od rozmiaru i charakterystyki danych wejściowych. W menu programu użytkownik może wybrać algorytm sortowania, a następnie wprowadzić dane ręcznie lub za pomocą standardowego wejścia.
Projekt został zaprojektowany w sposób modularny, co ułatwia dodawanie nowych algorytmów sortowania w przyszłości. Dzięki temu użytkownik może w prosty sposób eksperymentować z różnymi metodami sortowania i analizować ich efektywność.

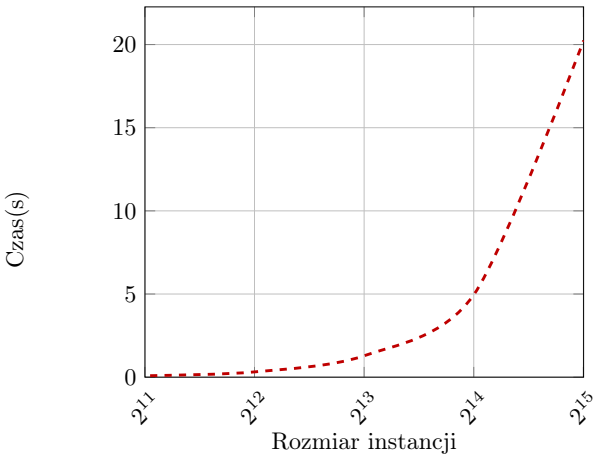
## 2 | Porównywanie czasów wykonywania

## WYKRESY DLA DANYCH TYPU A SHAPED ARRAY

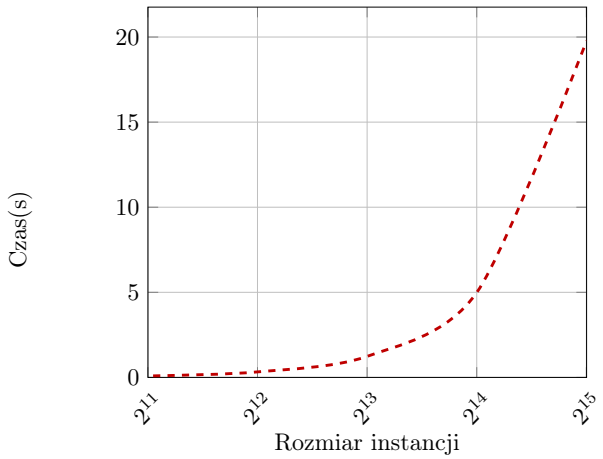
Heap Sort A Shaped Array



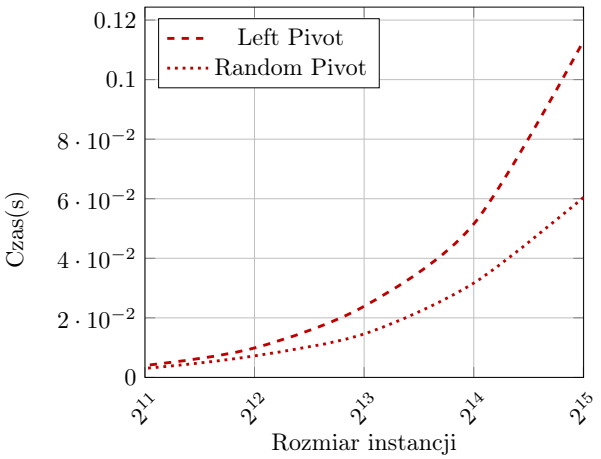

Insertion Sort A Shaped Array



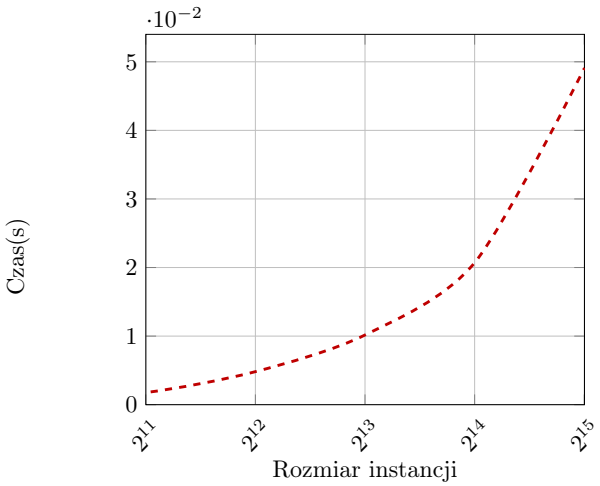

Selection Sort A Shaped Array



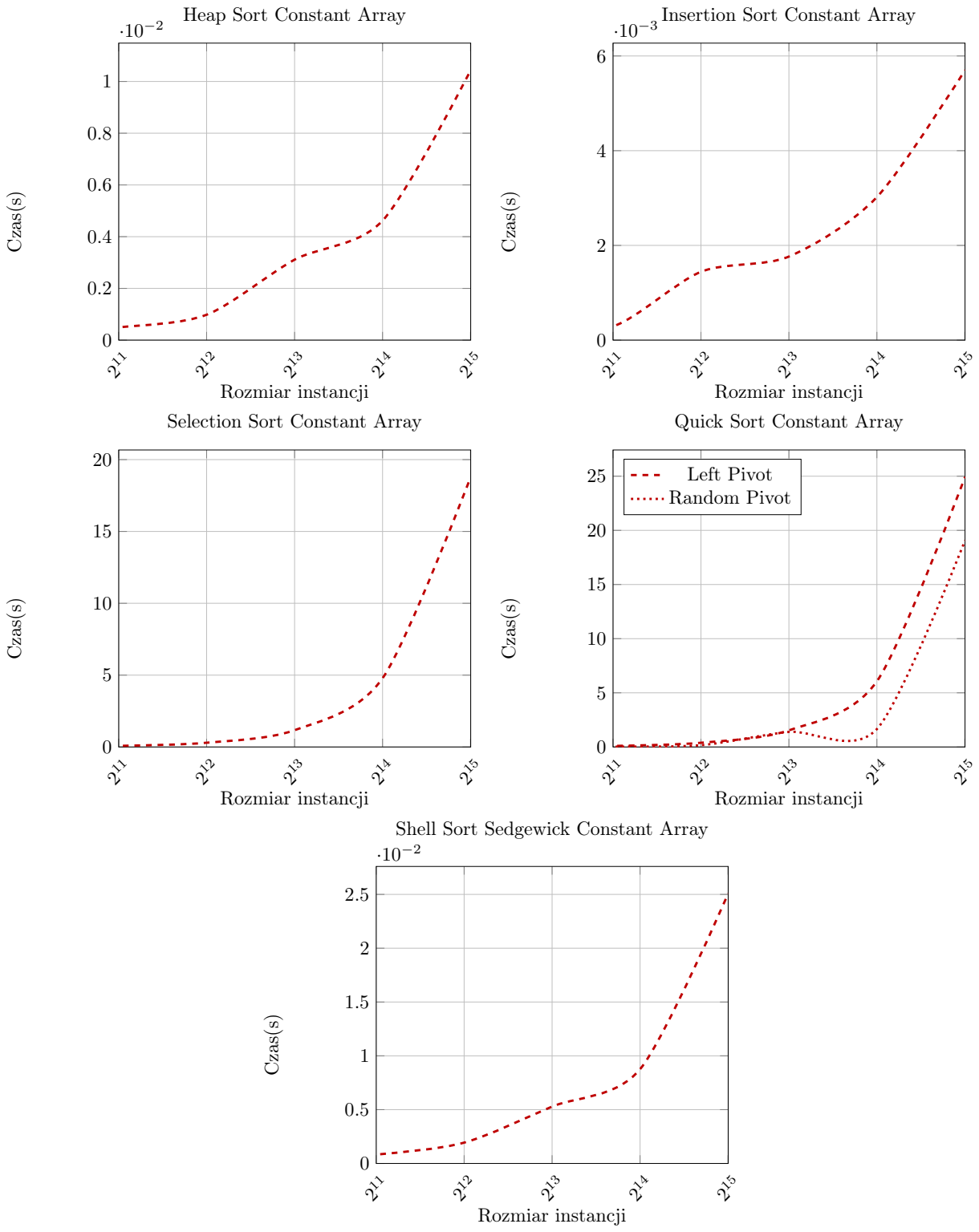

Quick Sort A Shaped Array



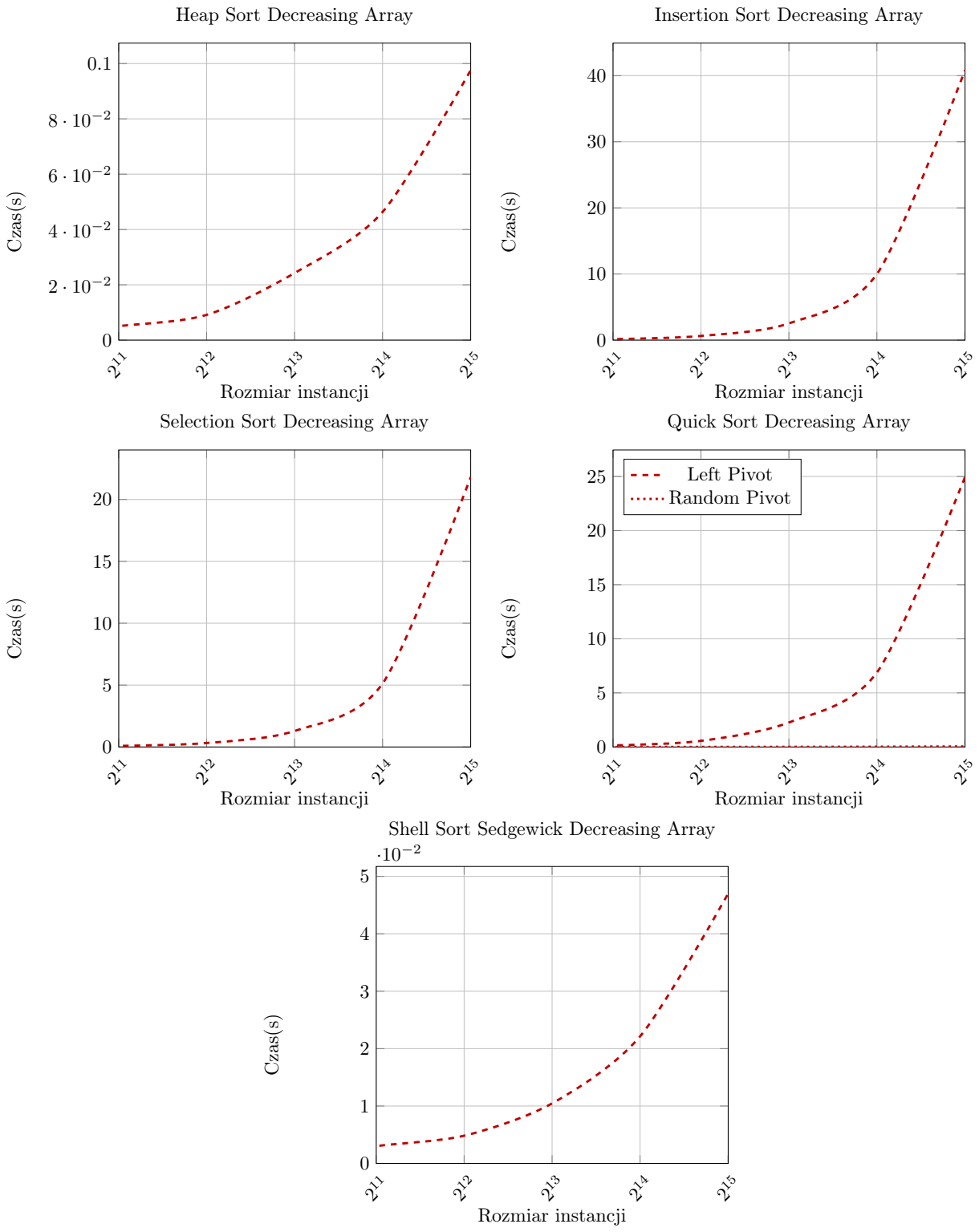

Shell Sort Sedgewick A Shaped Array



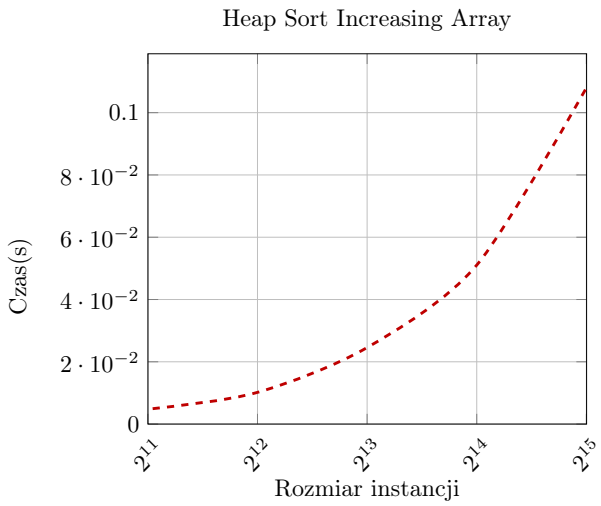
# WYKRESY DLA DANYCH TYPU CONSTANT ARRAY

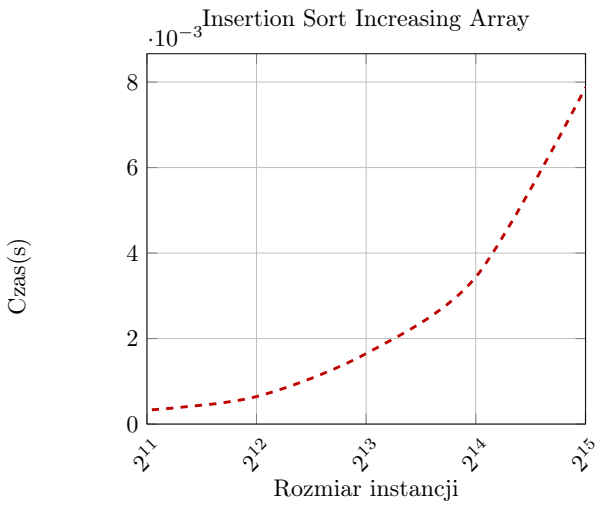


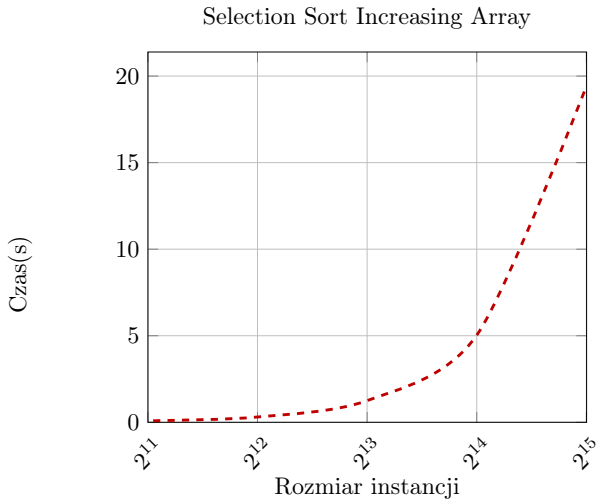
# WYKRESY DLA DANYCH TYPU DECREASING ARRAY

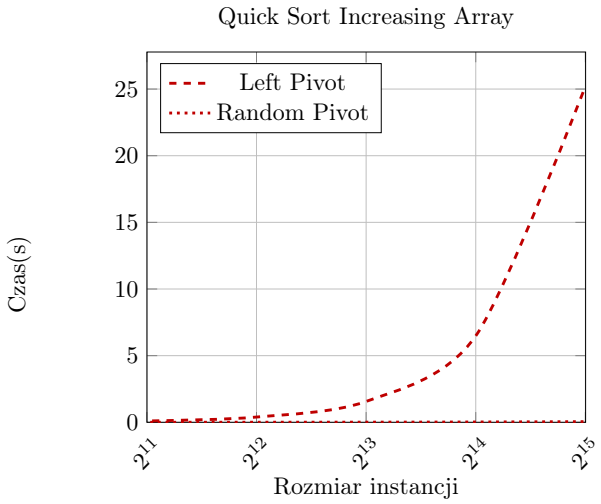


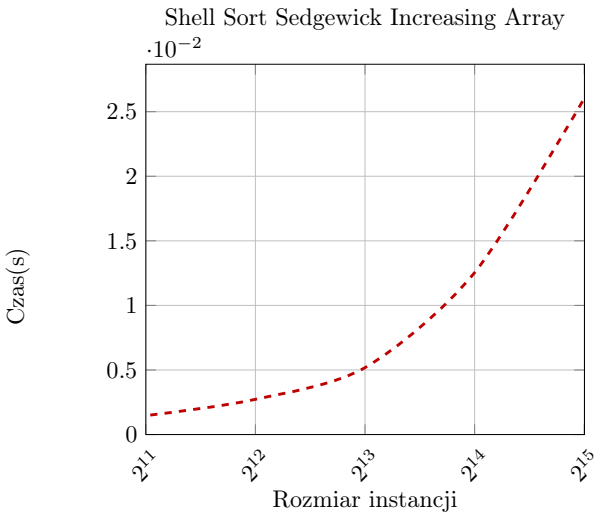
# WYKRESY DLA DANYCH TYPU INCREASING ARRAY



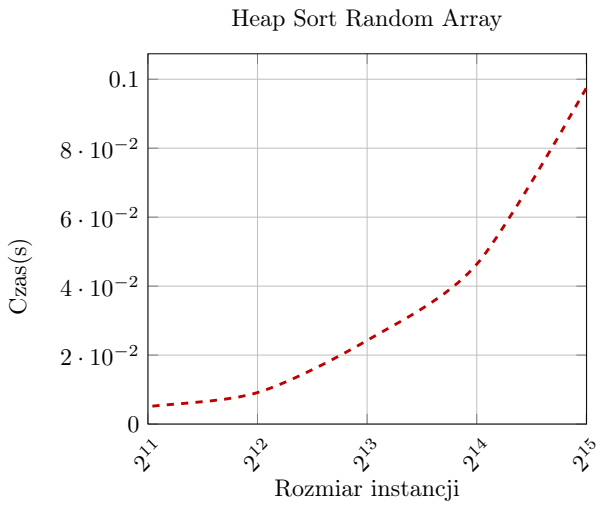



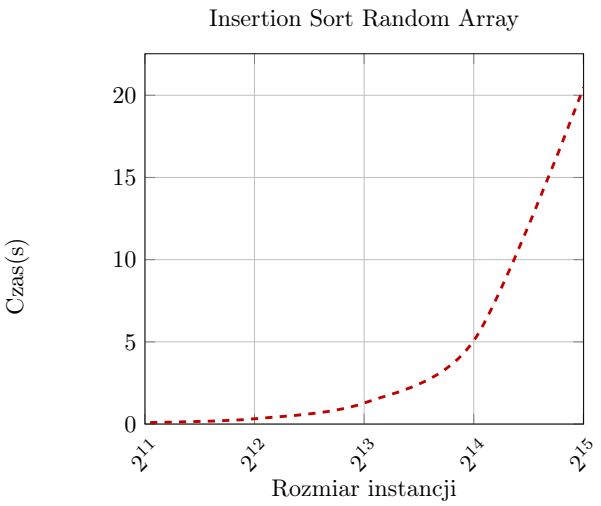



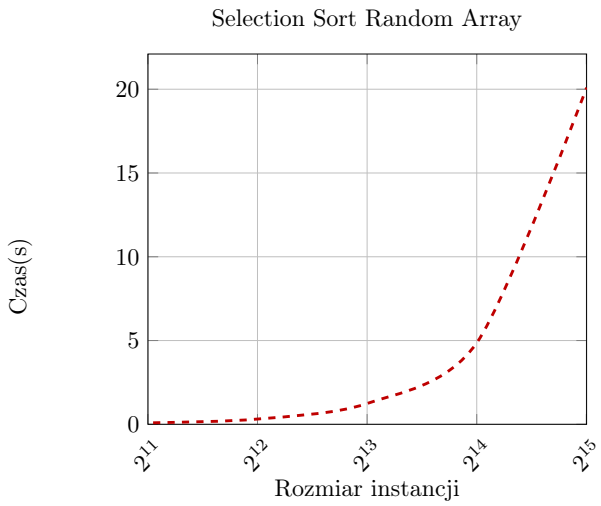



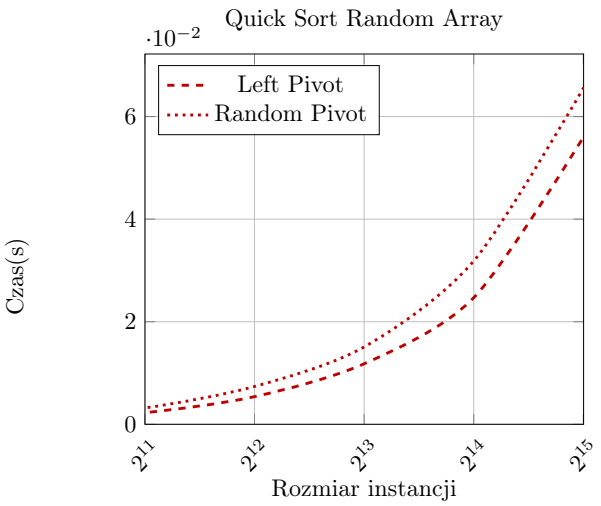



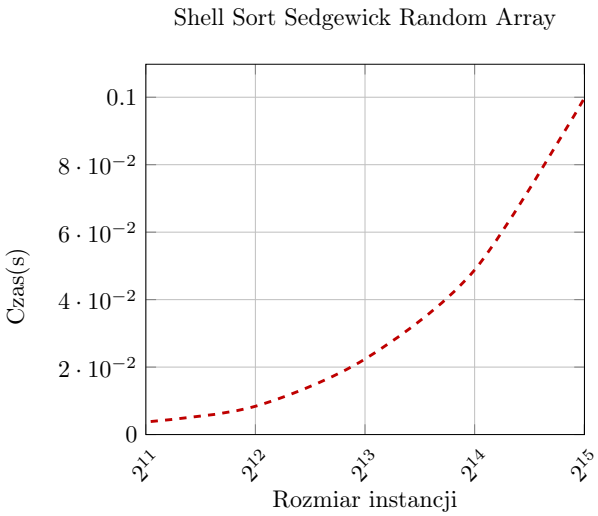
# WYKRESY DLA DANYCH TYPU RANDOM ARRAY



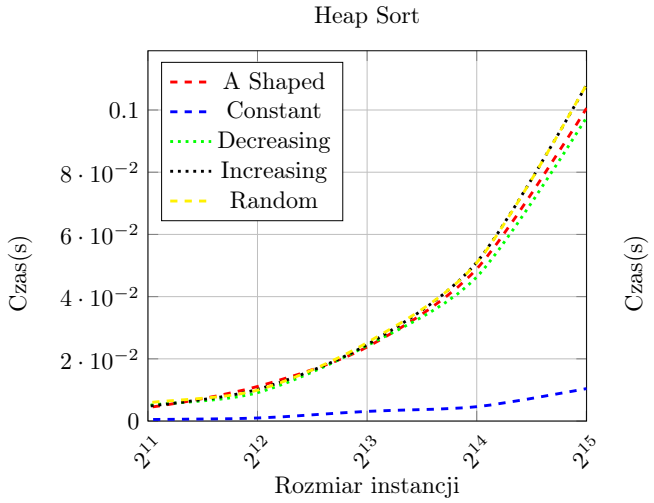



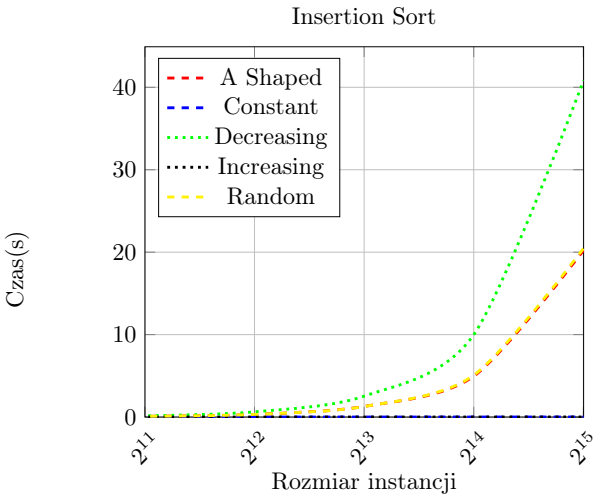



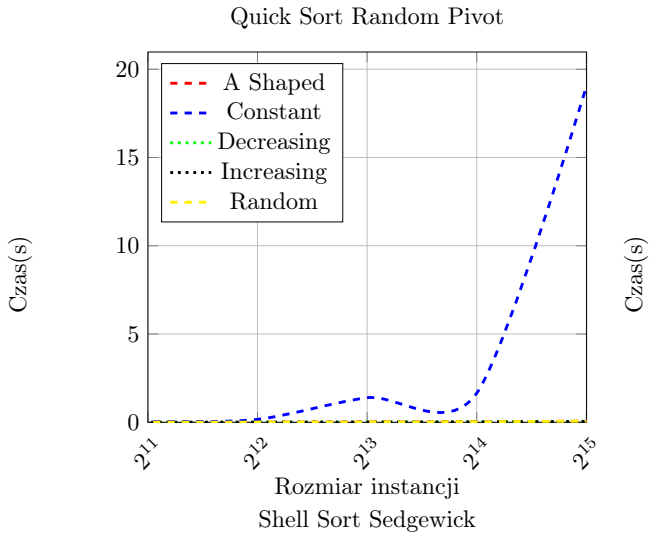



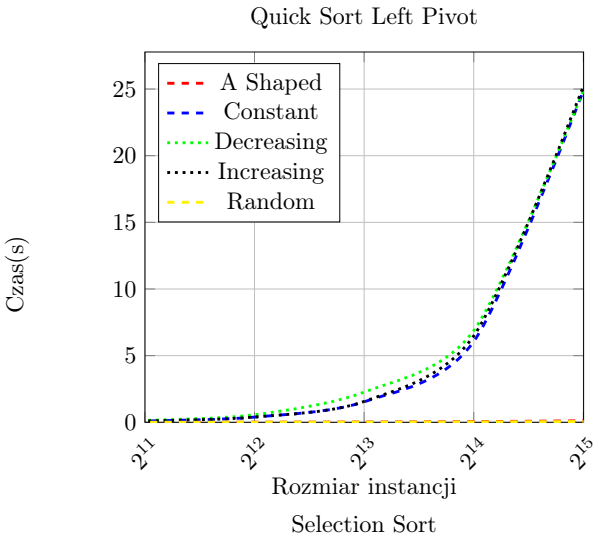



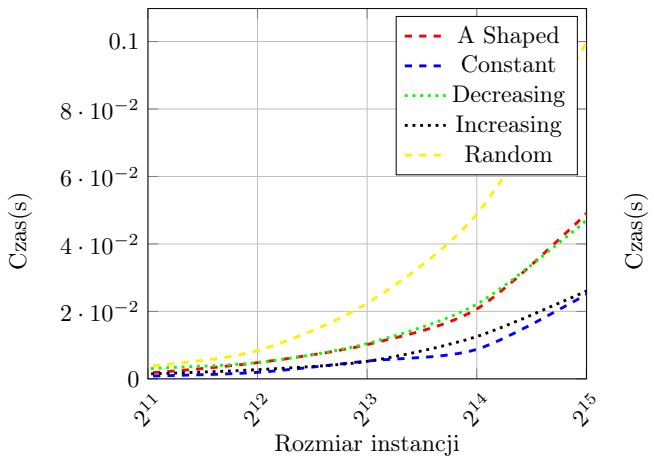
# WYKRESY ZBIORCZE

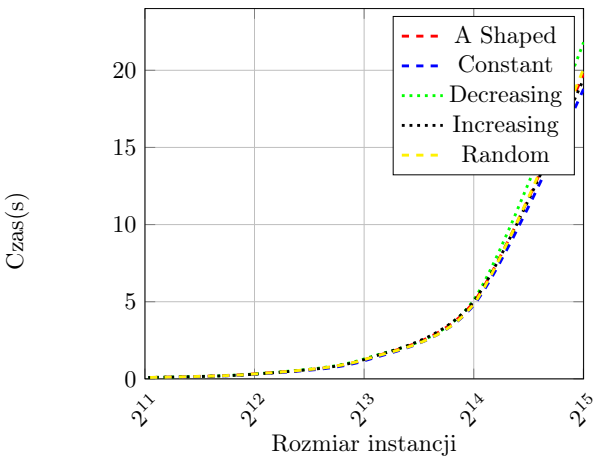


# 3 | Wnioski

Z analizy działania zaimplementowanych algorytmów sortowania wynika, że czas wykonania każdego z nich jest silnie uzależniony od charakterystyki danych wejściowych oraz rozmiaru listy. Algorytmy o złożoności kwadratowej, takie jak **Insertion Sort** czy **Selection Sort**, działają stosunkowo wolno dla dużych zbiorów danych, co czyni je nieefektywnymi w takich przypadkach. Z kolei algorytmy o złożoności **O(n log n)**, takie jak **Heap Sort** czy **Quick Sort**, wykazują znacznie lepszą wydajność, szczególnie dla dużych list.
Warto również zauważyć, że wybór strategii wyboru pivota w **Quick Sort** ma istotny wpływ na czas działania algorytmu. W przypadku losowego wyboru pivota (**Quick Sort Random**) algorytm działa bardziej równomiernie w porównaniu do wyboru skrajnie lewego pivota (**Quick Sort Left**), który może prowadzić do nieoptymalnych podziałów w przypadku niekorzystnych danych wejściowych.
Podsumowując, wybór odpowiedniego algorytmu sortowania powinien być uzależniony od charakterystyki danych oraz wymagań dotyczących wydajności. Algorytmy o złożoności **O(n log n)** są bardziej uniwersalne i efektywne dla dużych zbiorów danych, podczas gdy algorytmy o złożoności kwadratowej mogą być wystarczające dla małych list lub w sytuacjach, gdzie prostota implementacji jest priorytetem.