Struktury Danych	
Kierunek *Informatyczne Systemy Automatyki*	**Termin** *środa TN 15:15 – 16:55*
Imię, nazwisko, numer albumu *Jakub Wojtala 272542, Mateusz Marko 273168*	**Data** *5.05.2024r.*
Link do projektu https://github.com/kubikal7/Struktury-danych-2	

Sprawozdanie – miniprojekt nr 2

Struktury danych są istotnym elementem informatyki, umożliwiają efektywne zarządzanie i manipulację zbiorami danych. W ramach tego projektu koncentrujemy się na implementacji i analizie dwóch różnych struktur danych wykorzystywanych do budowy kolejki priorytetowej typu MAX: tablicy (Array) i kopca (Heap). Celem projektu jest nie tylko implementacja tych struktur, ale także szczegółowe zbadanie ich złożoności czasowej podczas wykonywania kluczowych operacji, takich jak wstawianie elementów, ekstrakcja elementu o najwyższym priorytecie oraz modyfikacja priorytetu istniejących elementów. Nasze badania pozwolą zrozumieć, w jakich scenariuszach każda z tych struktur oferuje lepszą wydajność, co jest istotne dla optymalizacji algorytmów i aplikacji wykorzystujących kolejki priorytetowe.

Spis treści

1. Wstęp

• Wstęp teoretyczny

➢ **Kolejka priorytetowa (Priority Queue)** - abstrakcyjny typ danych, który służy do przechowywania zbioru elementów w taki sposób, że każdy element posiada przypisany priorytet. W kolejce priorytetowej typu MAX elementy są uporządkowane tak, że element z najwyższym priorytetem jest zawsze dostępny jako pierwszy. Kolejka priorytetowa jest często używana w systemach operacyjnych do zarządzania procesami, symulacjach, algorytmach planowania i wszędzie tam, gdzie wymagane jest efektywne zarządzanie zadaniami według ich priorytetu.

• Użyte struktury

➢ **Tablica (Array)** – Jest to podstawowa struktura, która przechowuje elementy w ciągłym bloku pamięci. W kontekście kolejki priorytetowej, tablica może być używana do implementacji poprzez utrzymanie niestrukturalizowanego zbioru elementów, gdzie każda operacja wymaga przeszukania całej tablicy w celu znalezienia elementu o najwyższym priorytecie lub wyszukiwania odpowiedniego miejsca przy wstawianiu.

Operacje i złożoność

- ***insert***$(\boldsymbol{e},\boldsymbol{p})$ - Wstawianie elementu $\boldsymbol{e}$ z priorytetem $\boldsymbol{p}$. Złożoność $O(1)$ wstawianie na koniec tablicy lub przyjmując taktykę bez sortowania przy wstawianiu. Złożoność $O(n)$ dla taktyki wstawiania od razu w odpowiednie miejsce.
- ***extract-max()*** - Usunięcie i zwrócenie elementu o najwyższym priorytecie. Złożoność $O(n)$, gdyż wymaga przeszukania całej tablicy w celu znalezienia maksimum lub zwrócenia pierwszego elementu i przesunięcia pozostałych.
- ***find-max() / peek()*** - Złożoność $O(n)$, gdy wymaga przeszukania całej tablicy w celu znalezienia maksimum. Złożoność $O(1)$, gdy elementy są już posortowane.
- ***modify-key***$(\boldsymbol{e},\boldsymbol{p})$ - Zmiana priorytetu elementu. Złożoność $O(n)$ wymaga przeszukania tablicy w celu lokalizacji elementu i zmiany jego priorytetu.
- ***return-size()*** - Zwrócenie liczby elementów w kolejce. Złożoność $O(1)$.

- **Kopiec (Heap)** - Specjalna struktura drzewiasta, która umożliwia efektywną implementację kolejek priorytetowych. Kopiec typu max utrzymuje właściwość, gdzie wartość w każdym węźle jest większa niż wartości w jego dzieciach. Ta właściwość sprawia, że kopiec jest idealny do operacji związanych z priorytetami.

  Operacje i złożoność

  - ***insert***(e, p) - Wstawianie elementu e z priorytetem p. Złożoność pesymistyczna $O(\log n)$ element jest wstawiany na koniec kopca, a następnie "przepychany" w górę (bubble up) do odpowiedniej pozycji.
  - ***extract-max()*** - Usunięcie i zwrócenie elementu o najwyższym priorytecie. Złożoność pesymistyczna $O(\log n)$ element z korzenia jest usuwany, a kopiec jest reorganizowany, aby zachować strukturę kopca.
  - ***find-max() / peek()*** - Zwrócenie elementu o najwyższym priorytecie bez usuwania. Złożoność $O(1)$ element jest zawsze na wierzchu kopca.
  - ***modify-key***(e, p) - Zmiana priorytetu elementu. Złożoność $O(n)$ po zmianie priorytetu, element może być przesunięty w górę lub w dół w kopcu w zależności od zmiany.
  - ***return-size()*** - Zwrócenie liczby elementów w kopcu. Złożoność $O(1)$.

• Założenia projektowe

- Projekt zakłada implementację kolejki priorytetowej typu MAX przy użyciu dwóch struktur danych: kopca oraz tablicy. Głównym celem jest porównanie tych struktur pod kątem złożoności czasowej ich podstawowych operacji oraz analiza efektywności zależnie od rozmiaru danych i różnorodności priorytetów elementów.

- Eksperymenty przeprowadzone zostały dla 30 losowych populacji a nastepnie została z nich wyciągnięta średnia. Przyjęliśmy także zasadę FIFIO, która dla takich samych priorytetów zwróci nam ten, który został pierwszy dodany. Same badania z kolei wykonane zostały na stacjonarnym komputerze wyposażonym w procesor Intel Core i-7-13700H, dysk SSD i 16 GB pamięci RAM . Oprogramowanie użyte do implementacji i testów to środowisko programistyczne Visual Studio, z wykorzystaniem języka C++ bez dostępu do gotowych bibliotek takich jak STL, co wymagało samodzielnej implementacji wszystkich struktur danych oraz skorzystania z utworzonych wcześniej struktur w ramach poprzedniego projektu.

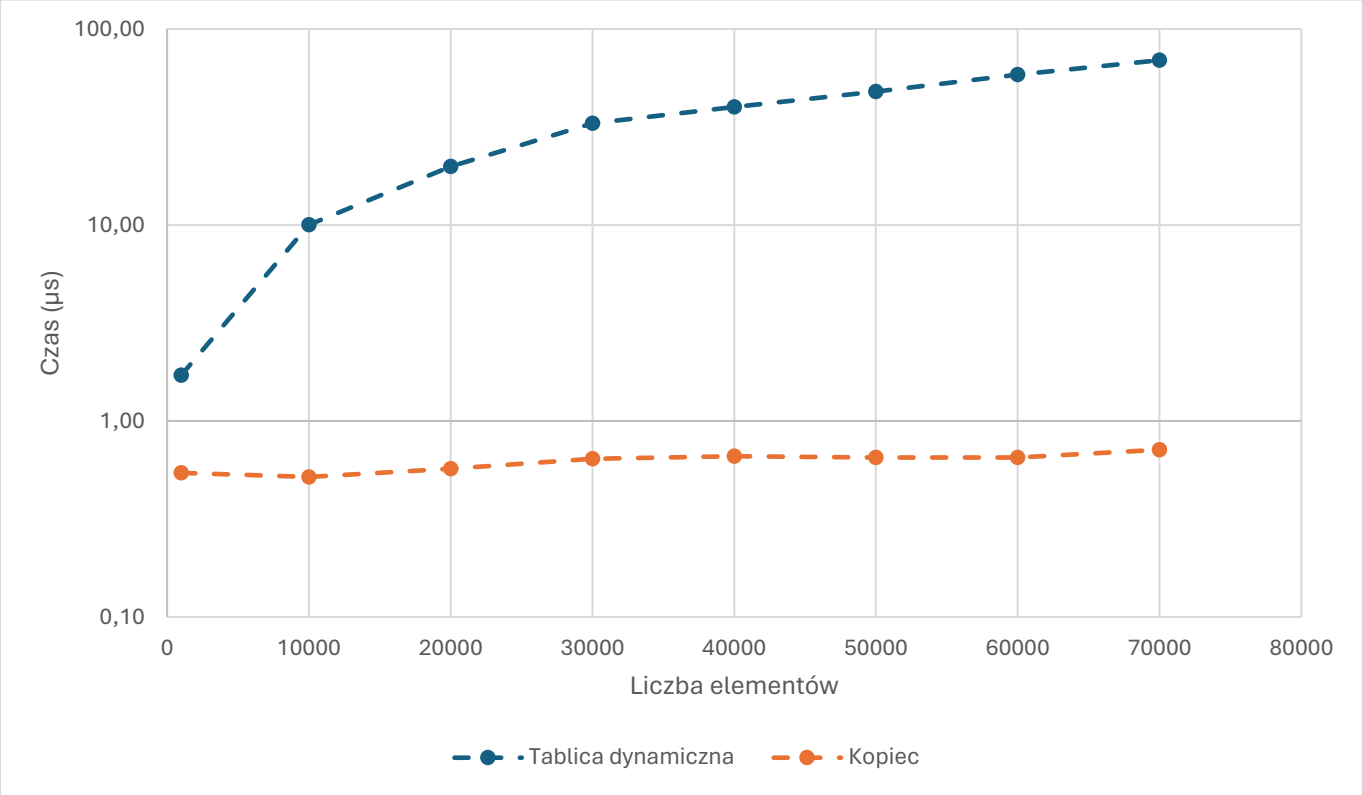
2. Badania

a) Operacja wstawiania (insert)

- Dodawanie z priorytetem 300 000 (TOP)

Tabela 1 Dodawanie z priorytetem 300 000 (TOP) - czas operacji (µs) dla poszczególnej ilości elementów i rodzaju struktury

Czas (µs)	Liczba elementów							
	1000	10 000	20 000	30 000	40 000	50 000	60 000	70 000
Tablica dynamiczna	1,71	10,01	19,86	33,03	40,02	47,88	58,46	69,27
Kopiec	0,54	0,52	0,57	0,64	0,66	0,65	0,65	0,71



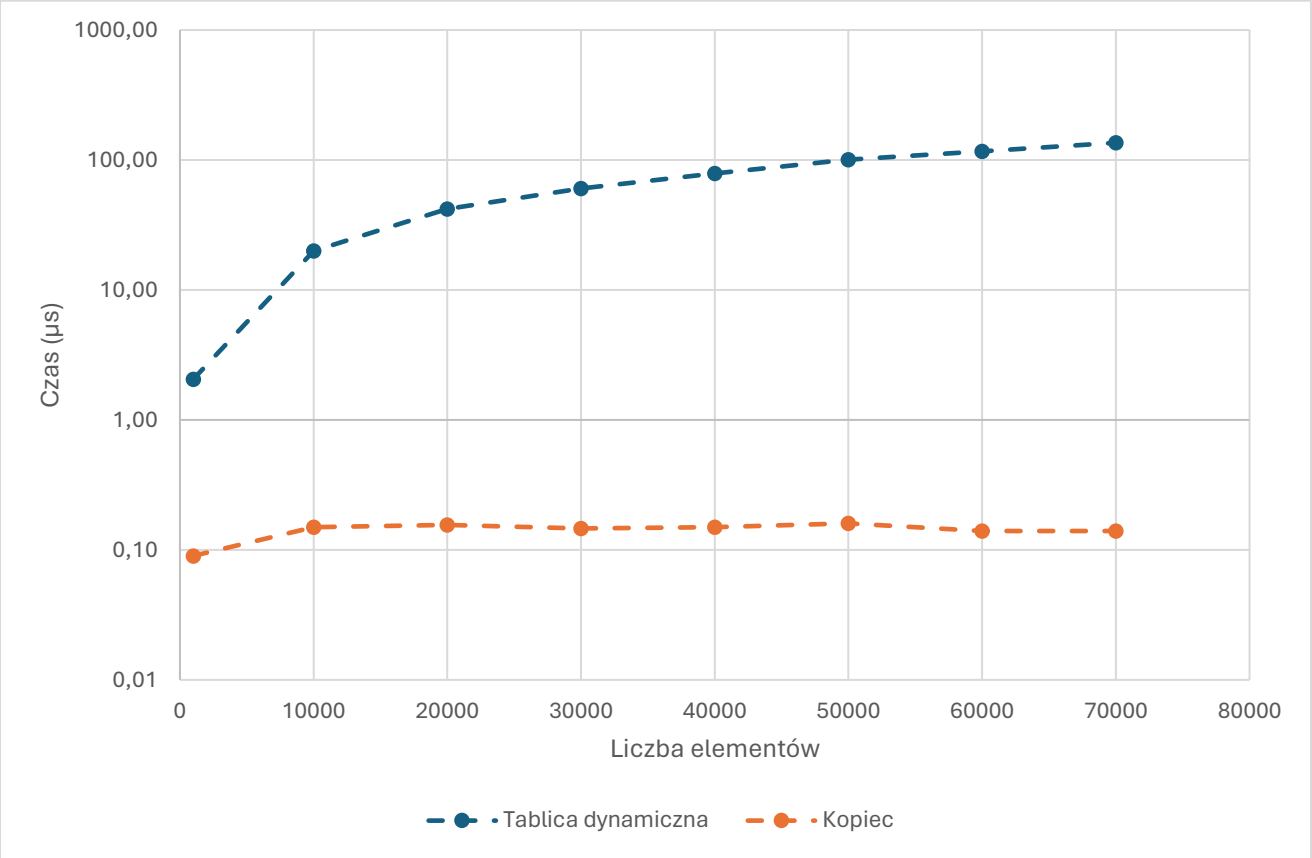
Rysunek 1 Wykres zależności czasu (µs) od liczby elementów dla Tablicy dynamicznej oraz Kopca – operacja wstawiania (TOP)

Politechnika Wrocławska

- Dodawanie z priorytetem 150 000 (MIDDLE)

Tabela 2 Dodawanie z priorytetem 150 000 (MIDDLE) - czas operacji (µs) dla poszczególnej ilości elementów i rodzaju struktury

Czas (µs)	Liczba elementów							
	1000	10 000	20 000	30 000	40 000	50 000	60 000	70 000
Tablica dynamiczna	2,06	19,94	41,97	60,35	78,75	100,51	116,67	135,62
Kopiec	0,09	0,15	0,16	0,15	0,15	0,16	0,14	0,14



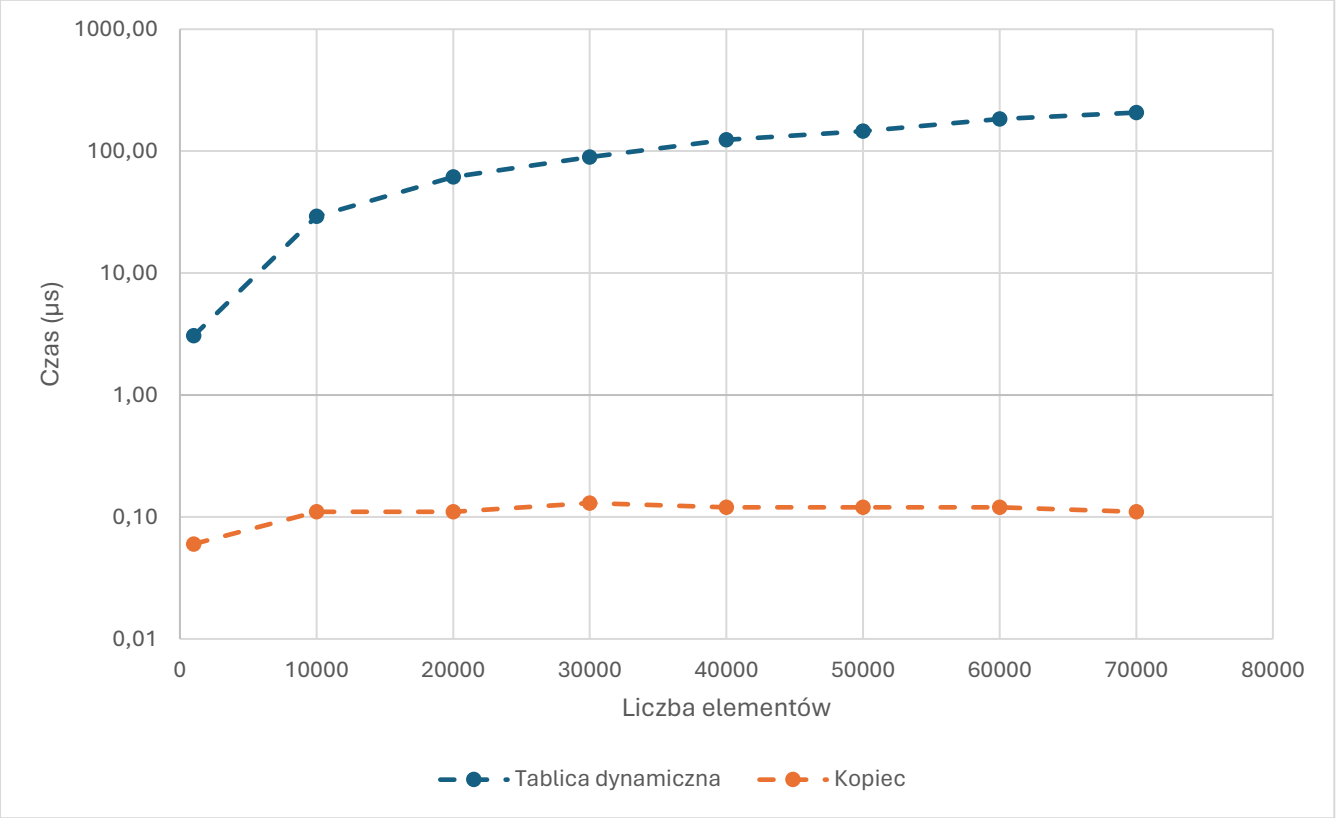
Rysunek 2 Wykres zależności czasu (µs) od liczby elementów dla Tablicy dynamicznej oraz Kopca – operacja wstawiania (MIDDLE)

Politechnika Wrocławska

- Dodawanie z priorytetem 1 (BOTTOM)

Tabela 3 Dodawanie z priorytetem 1 (BOTTOM) - czas operacji (µs) dla poszczególnej ilości elementów i rodzaju struktury

Czas (µs)	Liczba elementów							
	1000	10 000	20 000	30 000	40 000	50 000	60 000	70 000
Tablica dynamiczna	3,06	29,30	61,57	89,53	124,23	145,75	183,44	207,21
Kopiec	0,06	0,11	0,11	0,13	0,12	0,12	0,12	0,11



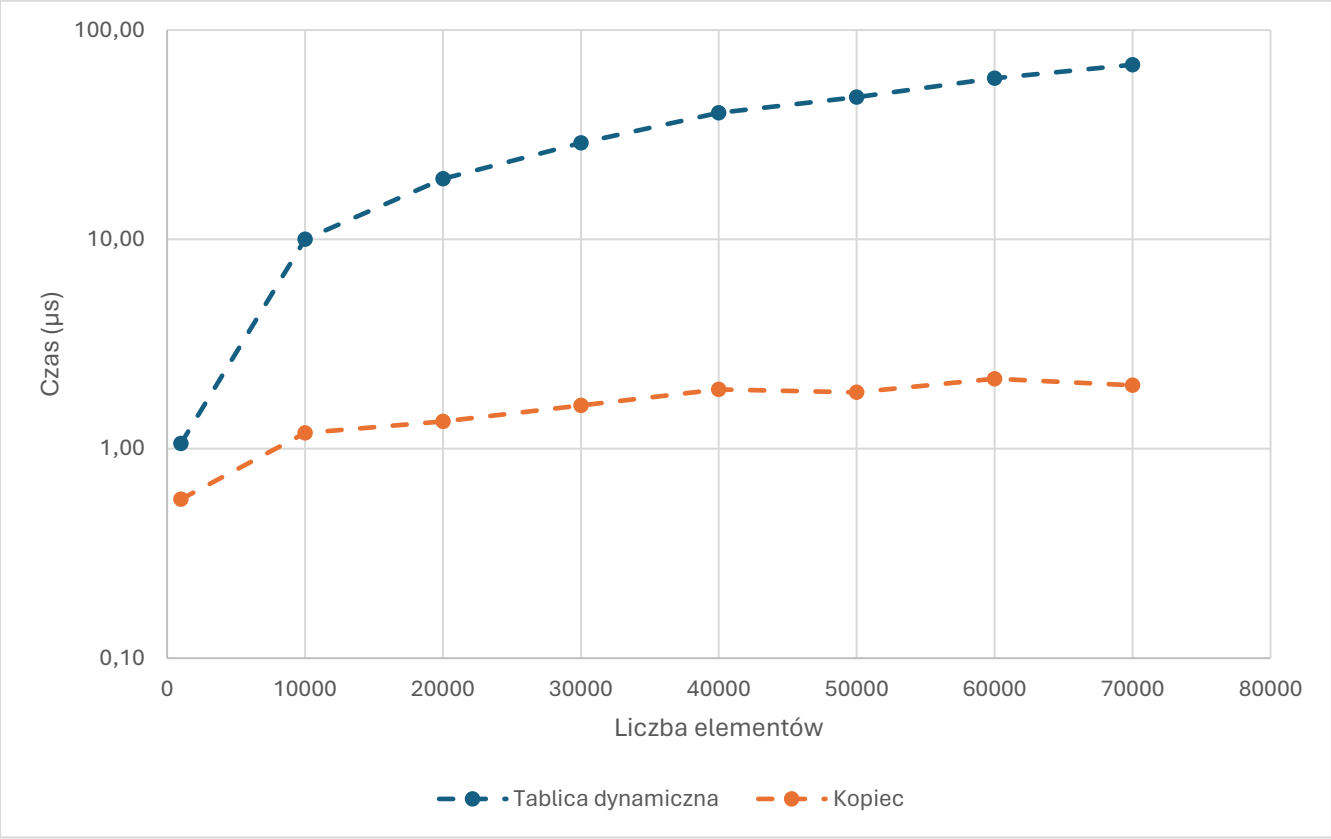
Rysunek 3 Wykres zależności czasu (µs) od liczby elementów dla Tablicy dynamicznej oraz Kopca – operacja wstawiania (BOTTOM)

Jak możemy zauważyć powyżej w każdym z przypadków zarówno w w dodawaniu z priorytetem 1, jak i w dodawaniu z priorytetem 150 000 a także w dodawaniu z priorytetem 300 000 znacznie lepiej radzi sobie kopiec osiągając nieporównywalnie niższe czasy operacji względem tablicy.

b) Operacja ekstrakcji maksimum (extract-max)

Tabela 4 Ekstrakcja maksimum - czas operacji (µs) dla poszczególnej ilości elementów i rodzaju struktury

Czas (µs)	Liczba elementów							
	1000	10 000	20 000	30 000	40 000	50 000	60 000	70 000
Tablica dynamiczna	1,06	10,02	19,46	28,92	40,24	47,81	58,87	68,31
Kopiec	0,57	1,19	1,35	1,61	1,92	1,86	2,16	2,01



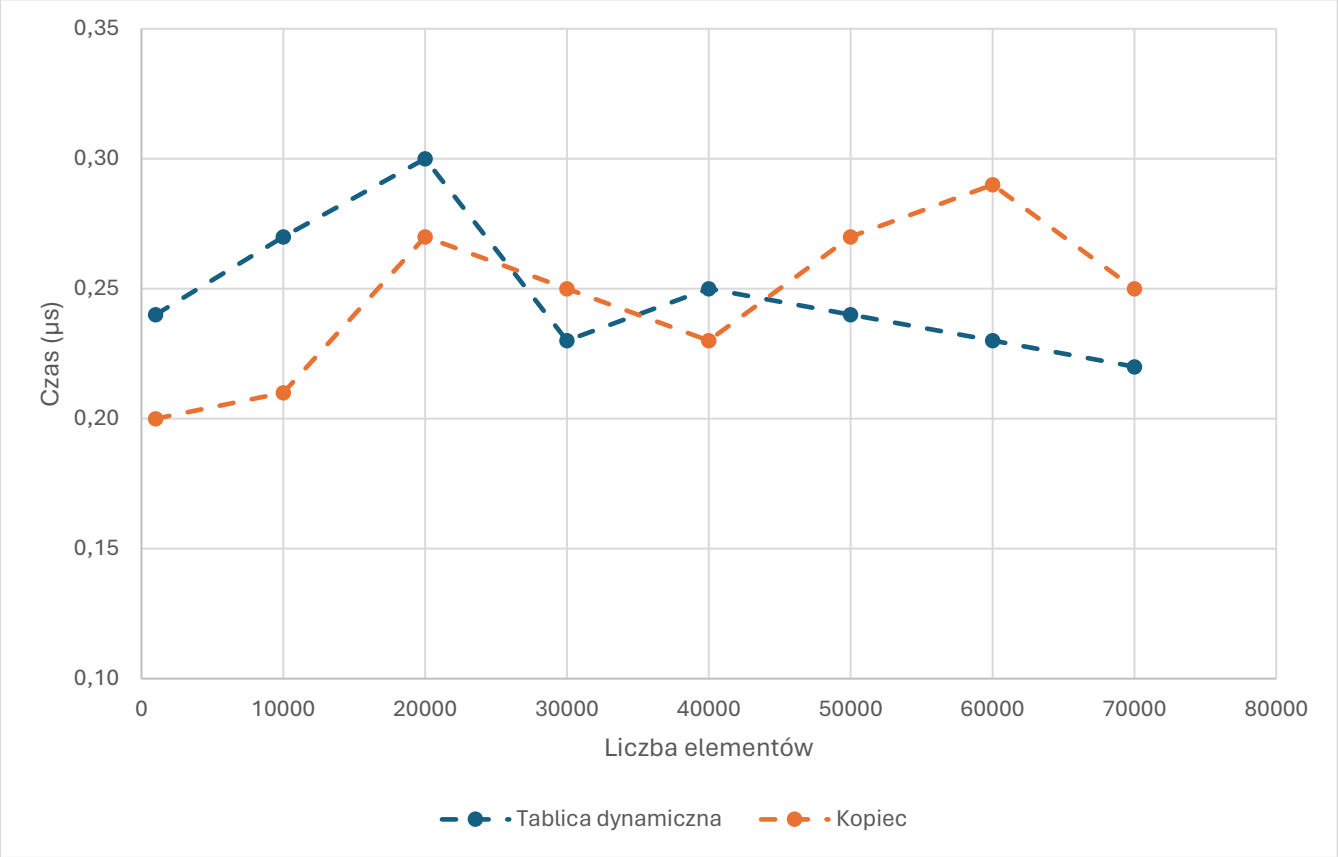
Rysunek 4 Wykres zależności czasu (µs) od liczby elementów dla Tablicy dynamicznej oraz Kopca – operacja usunięcia i zwrócenia

Tutaj ponownie przy operacji ekstrakcji maksimum kopiec osiąga znacznie lepsze czasy niż tablica pomimo wzrostu ilości elementów.

c) Operacja znajdowania maksimum (find-max/peek)

Tabela 5 Znajdowanie maksimum - czas operacji (µs) dla poszczególnej ilości elementów i rodzaju struktury

Czas (µs)	Liczba elementów							
	1000	10 000	20 000	30 000	40 000	50 000	60 000	70 000
Tablica dynamiczna	0,24	0,27	0,30	0,23	0,25	0,24	0,23	0,22
Kopiec	0,20	0,21	0,27	0,25	0,23	0,27	0,29	0,25



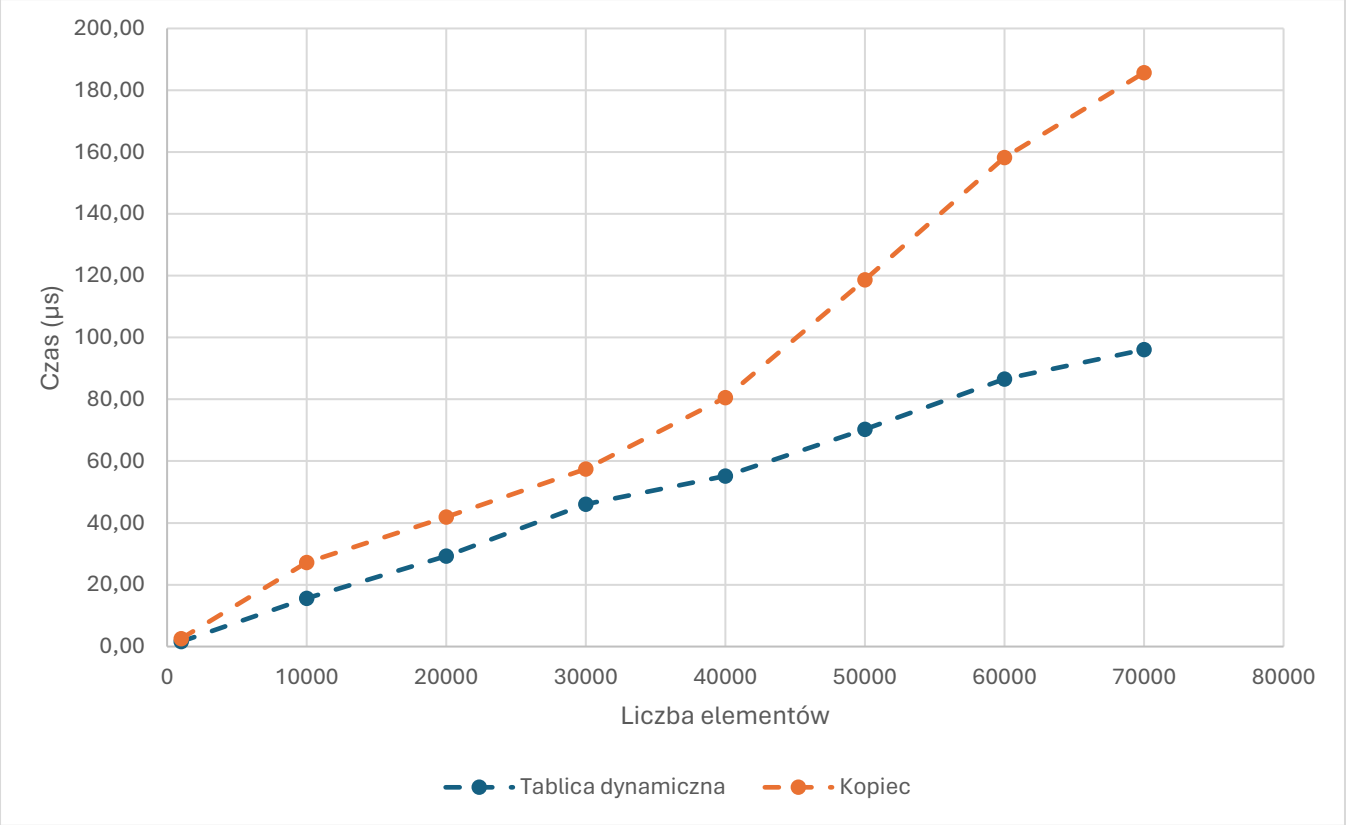
Rysunek 5 Wykres zależności czasu (µs) od liczby elementów dla Tablicy dynamicznej oraz Kopca – operacja znalezienia elementu o max priorytecie

W przypadku operacji znajdywania maksimum możemy zaobserwować nieco inną sytuacje niż powyższe. W przeciwieństwie do wcześniejszych operacji kopiec oraz tablica dynamiczna działają ze złożonością $O(1)$, a różnice między nimi są bardzo niewielkie.

d) Operacja modyfikacji klucza (modify-key)

Tabela 6 Modyfikacja klucza (Priorytetu) - czas operacji (µs) dla poszczególnej ilości elementów i rodzaju struktury

Czas (µs)	Liczba elementów							
	1000	10 000	20 000	30 000	40 000	50 000	60 000	70 000
Tablica dynamiczna	1,59	15,63	29,31	46,11	55,20	70,27	86,54	96,07
Kopiec	2,56	27,23	41,95	57,51	80,56	118,67	158,26	185,67



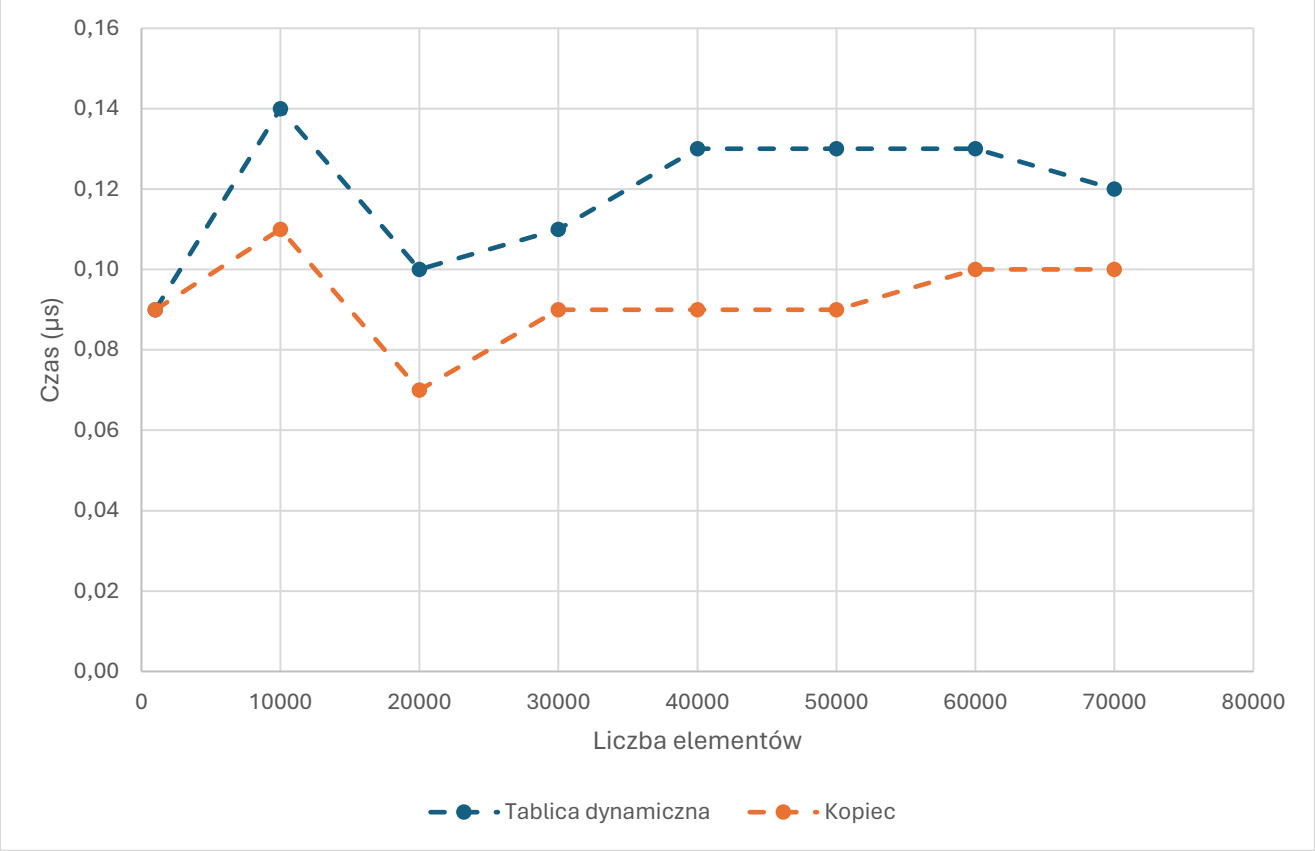
Rysunek 6 Wykres zależności czasu (µs) od liczby elementów dla Tablicy dynamicznej oraz Kopca – operacja modyfikowania klucza (priorytetu)

Gdy mamy do czynienia z operacją modyfikowania klucza (priorytetu) widzimy że Tablica sprawuje się lepiej i osiąga mniejsze czasy niż kopiec zarówno dla mniejszych jak i większych wartości. Zależy to jednak od wielu czynników przykładowo od wartości przechowywanych elementów czy kolejności wstawiania szukanego w późniejszej kolejności elementu.

e) Operacja zwracania rozmiaru (return-size)

Tabela 7 Zwracanie rozmiaru - czas operacji (µs) dla poszczególnej ilości elementów i rodzaju struktury

Czas (µs)	Liczba elementów							
	1000	10 000	20 000	30 000	40 000	50 000	60 000	70 000
Tablica dynamiczna	0,09	0,14	0,10	0,11	0,13	0,13	0,13	0,12
Kopiec	0,09	0,11	0,07	0,09	0,09	0,09	0,10	0,10



Rysunek 7 Wykres zależności czasu (µs) od liczby elementów dla Tablicy dynamicznej oraz Kopca – operacja zwrócenia rozmiaru

W ostatniej już badanej operacji można zaobserwować, że ponownie kopiec okazuje się w praktyce lepszy, lecz różnice są bardzo małe, wręcz znikome, a obie struktury działają ze złożonością $O(n)$.

3. Wnioski

- Kopiec jest efektywny dla operacji związanych z kolejka priorytetową, szczególnie w kontekście ekstrakcji elementów o najwyższym priorytecie oraz wstawiania nowych elementów jak mogliśmy zauważyć na wykresach, gdzie złożoność oscylowała w okolicy $O(1)$. Jednak w przypadku *modify-key* od pewnego momentu zaobserwowaliśmy gorsze rezultaty niż zakładana złożoność średnia $O(n)$. Dzięki strukturze drzewiastej, operacje takie jak *extract-max* i *insert* mają małą złożoność, w najgorszym przypadku $O(\log n)$, co czyni kopiec idealnym wyborem dla programów, które wymagają szybkiego zwracania elementu o najwyższym priorytecie. Będą więc idealne dla systemów planowania i algorytmów zachłannych, które często wykorzystują operacje typu "wyjmij największy".

- Tablica choć prosta w implementacji i skuteczna dla operacji *find-max* ze złożonością $O(1)$ przy pojedynczym przeszukiwaniu, jest mniej efektywna dla operacji *extract-max* i *insert*, które mają złożoność $O(n)$ z powodu potrzeby przeszukiwania całej tablicy lub przesuwania elementów. Tablica może być użyteczna w scenariuszach, gdzie kolejka priorytetowa jest rzadko aktualizowana, ale często odczytywana, choć generalnie kopiec oferuje lepsze ogólne wydajności.

- Skalowalność i efektywność obu struktur danych mogą różnić się w zależności od specyfikacji sprzętowej i implementacji. Dla przykładui w systemach z ograniczoną pamięcią, dodatkowy narzut pamięciowy związany z kopcem może być gorszy i lepiej sprawdzi się prosta tablica.

- Wybór struktury danych dla implementacji kolejki priorytetowej powinien być podyktowany konkretnymi wymaganiami, takimi jak częstość aktualizacji danych, potrzeba szybkiego dostępu do najwyższego priorytetu, oraz przewidywana wielkość danych. Kopiec jest zdecydowanie lepszym wyborem dla większości zastosowań ze względu na jego wydajność i efektywność zarządzania priorytetami.

4. Bibliografia

- http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/files/sd/w3.pdf
- http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/files/sd/w4.pdf
- http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/files/sd/w5.pdf
- https://www.geeksforgeeks.org/priority-queue-set-1-introduction/
- https://en.cppreference.com/w/cpp/container/priority_queue