Struktury danych	
Kierunek *Informatyczne Systemy Automatyki*	**Termin** *środa TP* $15^{15} - 16^{55}$
Imię, nazwisko, numer albumu *Mikołaj Nowak 280082, Nathan Smyrek 280166*	**Data** *14.05.2025*
Link do projektu `https://github.com/devoid-of-thought/Strunktury_Danych_Projekt_nr_2`	

Sprawozdanie – miniprojekt 2

Spis treści

1 Wstęp

1.1 Kolejka priorytetowa - ogólnie

1.1.1 Opis struktury

Kolejka priorytetowa to struktura, w której elementy są uporządkowane w kolejności zależnej od ich priorytetu: w naszej implementacji począwszy od elementu o najwyższym priorytecie do tego o najniższym priorytecie.

1.1.2 Przykładowe zastosowania

- Algorytmy takie jak np:
  - Algorytm Dijkstry
  - Algorytm Huffmana
  - Algorytmy Best-first
  - Algorytm Priama
- Zarządzanie przepustowością łącza
- Symulacja zdarzeń dyskretnych
- W automatyce i robotyce przy nadawaniu kolejności wykonywanych zadań
- W systemach operacyjnych
- W systemach medyczny i służb ratunkowych by odpowiednio rozdzielać personel w sytuacjach kryzysowych

1.2 Kolejka priorytetowa - Lista wiązana

1.2.1 Opis Struktury

Zaimplementowana kolejka priorytetowa wykorzystuje listę wiązaną. Każdy element kolejki przechowuje trzy wartości: liczbę, priorytet oraz wskaźnik na następny element. W przypadku dodania do kolejki elementu o priorytecie, który już się w niej znajduje nowy element jest dodany po ostatnim istniejącym elemencie o tym samym priorytecie.

1.3 Kolejka priorytetowa - Kopiec

1.3.1 Opis Struktury

Kopiec to drzewo binarne reprezentowane przez tablicę. W kopcu maksymalnym każdy rodzic ma wartość większą lub równą od dzieci.

Dla elementu o indeksie i:

- Rodzic: (i - 1) / 2
- Lewe dziecko: 2i + 1
- Prawe dziecko: 2i + 2

Kopce mogą być używane do implementacji kolejek priorytetowych, ponieważ ustalenie odpowiedniej relacji umożliwia bezpośredni dostęp do wierzchołka o największej lub najmniejszej wartości z danego zbioru zapisanego w kopcu.

1.4 Teoretyczna złożoność obliczeniowa

Tabela 1: Złożoność obliczeniowa obu implementacji kolejki priorytetowej

Operacja	**Lista wiązana**	**Kopiec**
`insert()`	$O(n)$	$O(\log n)$
`extract-max()`	$O(1)$	$O(\log n)$
`find-max()`	$O(1)$	$O(1)$
`modify-key(e, p)`	$O(n)$	$O(n)$
`return-size()`	$O(1)$	$O(1)$

2 Założenia Projektowe

2.1 Parametry użytego sprzętu

- System operacyjny : Linux Mint 21.1 Cinnamon
- Processor : Intel i7-1260P 12-generacji (12 rdzeni, 16 wątków, 3.40-4.70 GHz, 18MB cache)
- Pamięć RAM : 32GB
- Środowisko : Clion 2024.1.3
- Kompiler : g++ 11.4.0

2.2 Plan eksperymentu

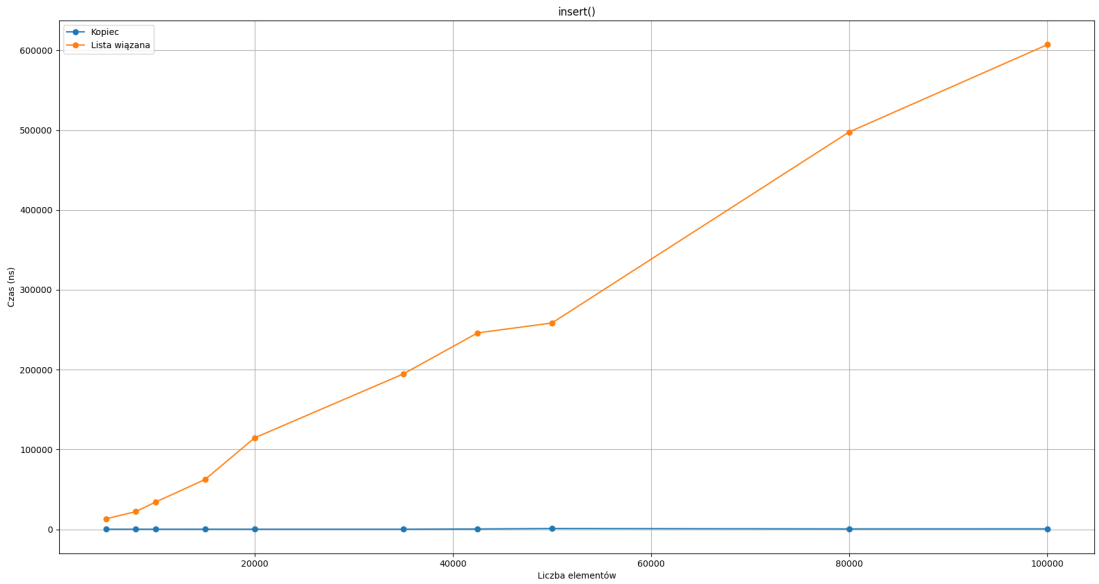
Przeprowadzona badania dla 10 różnych rozmiar struktur : 5000, 8000, 10000, 15000, 20000, 35000, 42500, 50000, 80000, 100000. Zostały one wypełnione losowymi liczbami z zakresu -2 147 483 647 do 2 147 483 647. Dla operacji 5000, 10000, 20000, 35000 zostało wykonane 50 prób, natomiast dla operacji 42500, 50000,80000, 100000 jedynue 10 ze względu na wolną prędkość tworzenia kopii kolejki priorytetowej opartej na liście wiązanej, po czym wyniki zostały uśrednione.

3 Badania

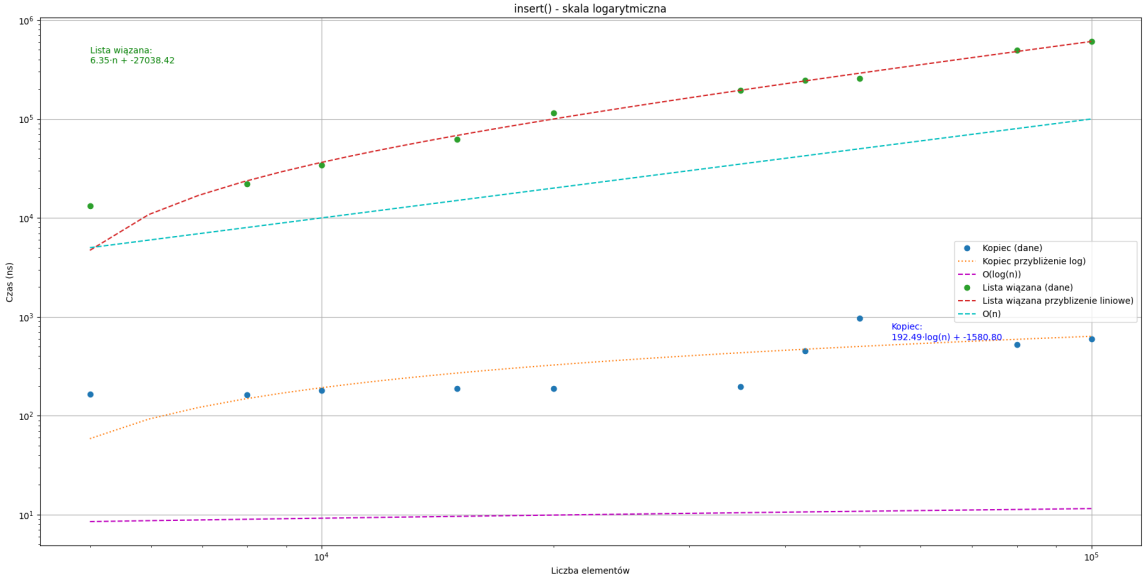
3.1 Operacja insert()

Tabela 2: Wyniki dla metody insert()

Liczba elementów	**Kopiec [ns]**	**Lista wiązana [ns]**
5,000	166	13154
8,000	162	22108
10,000	181	34105
15,000	188	62353
20,000	189	114546
35,000	196	194556
42,500	454	246018
50,000	972	258403
80,000	523	497685
100,000	598	607042



Rysunek 1: Wstawienie elementu o losowym priorytecie do kolejki

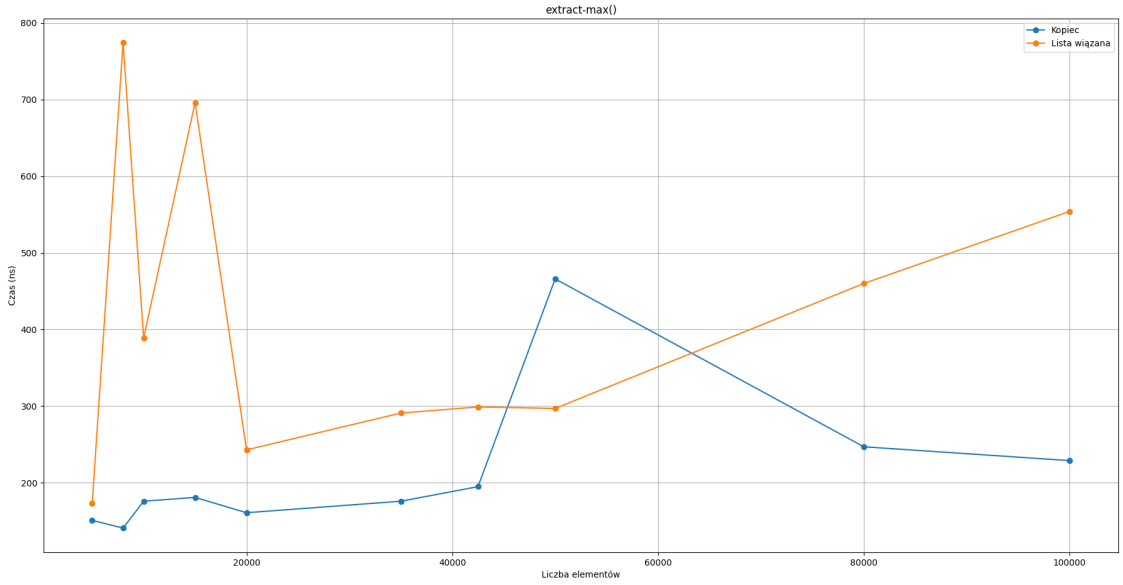


Rysunek 2: Wstawienie elementu do kolejki o losowym priorytecie - skala logarytmiczna na osi x i y

3.2 Operacja extract-max()

Tabela 3: Wyniki dla metody extract-max()

Liczba elementów	Kopiec [ns]	Lista wiązana [ns]
5,000	151	173
8,000	141	774
10,000	176	389
15,000	181	696
20,000	161	243
35,000	176	291
42,500	195	299
50,000	466	297
80,000	247	460
100,000	229	554

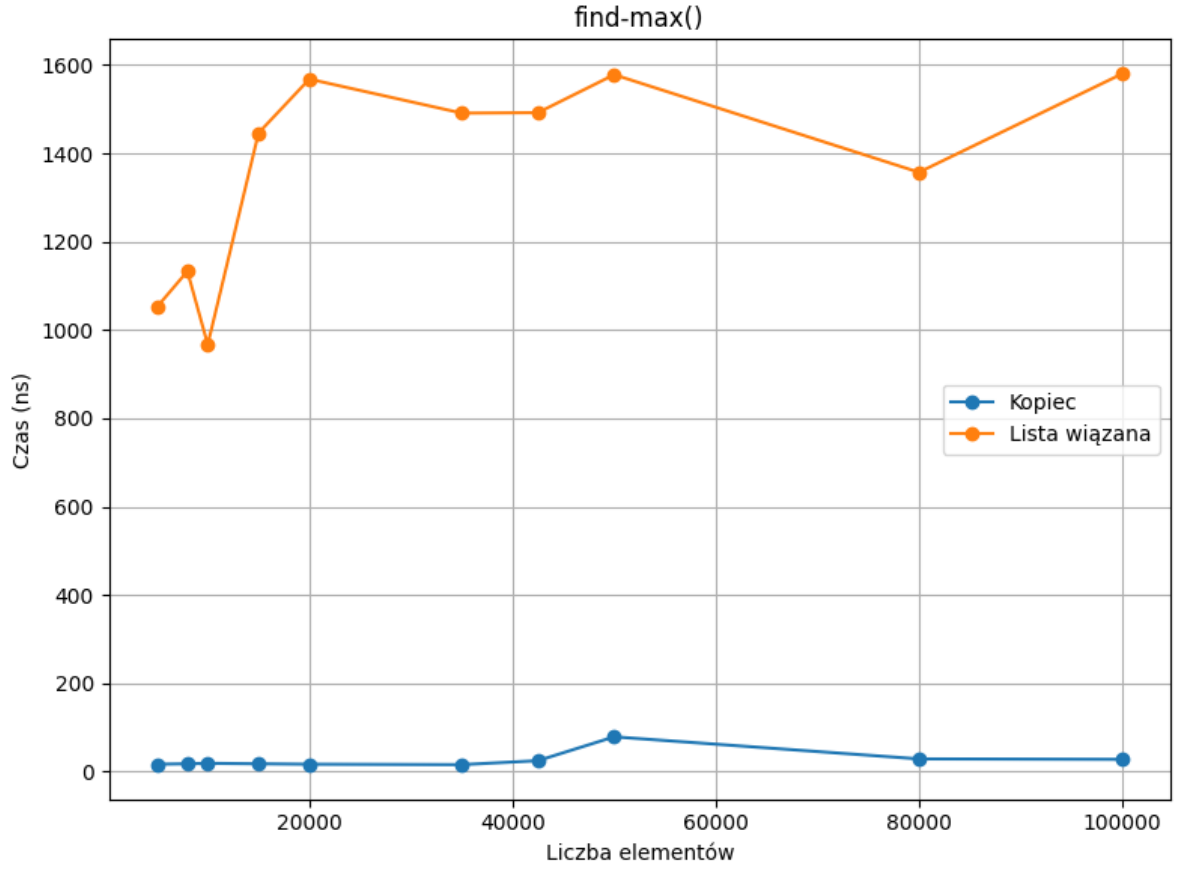


Rysunek 3: Wyjęcie z kolejki elementu o najwyższym priorytecie

3.3 Operacja find-max()

Tabela 4: Wyniki dla metody find-max()

Liczba elementów	Kopiec [ns]	Lista wiązana [ns]
5,000	17	1052
8,000	18	1133
10,000	19	967
15,000	18	1446
20,000	17	1568
35,000	16	1491
42,500	25	1492
50,000	79	1578
80,000	29	1357
100,000	28	1580

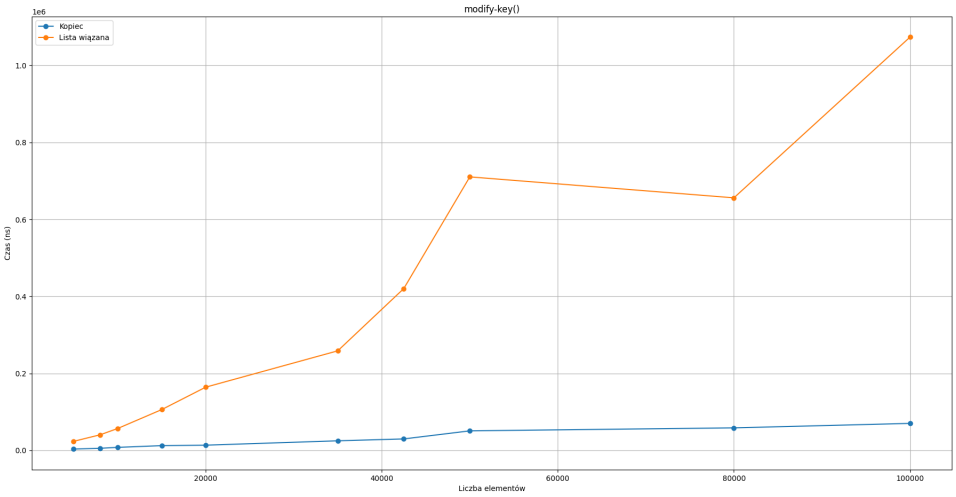


Rysunek 4: Znalezienie w kolejce elemencie o najwyższym priorytecie

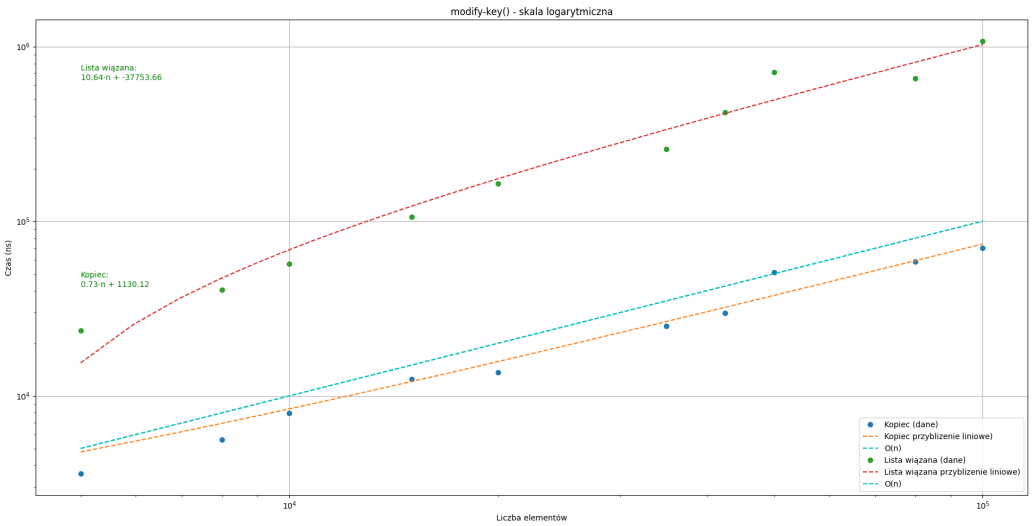
3.4 Operacja modify-key()

Tabela 5: Wyniki dla metody modify-key()

Liczba elementów	Kopiec [ns]	Lista wiązana [ns]
5,000	3592	23682
8,000	5624	40547
10,000	7990	56993
15,000	12436	106022
20,000	13645	164335
35,000	25029	258708
42,500	29890	420393
50,000	50870	710516
80,000	58624	656107
100,000	70242	1073468



Rysunek 5: Zmodyfikonanie priorytetu elementu o losowym indeksie

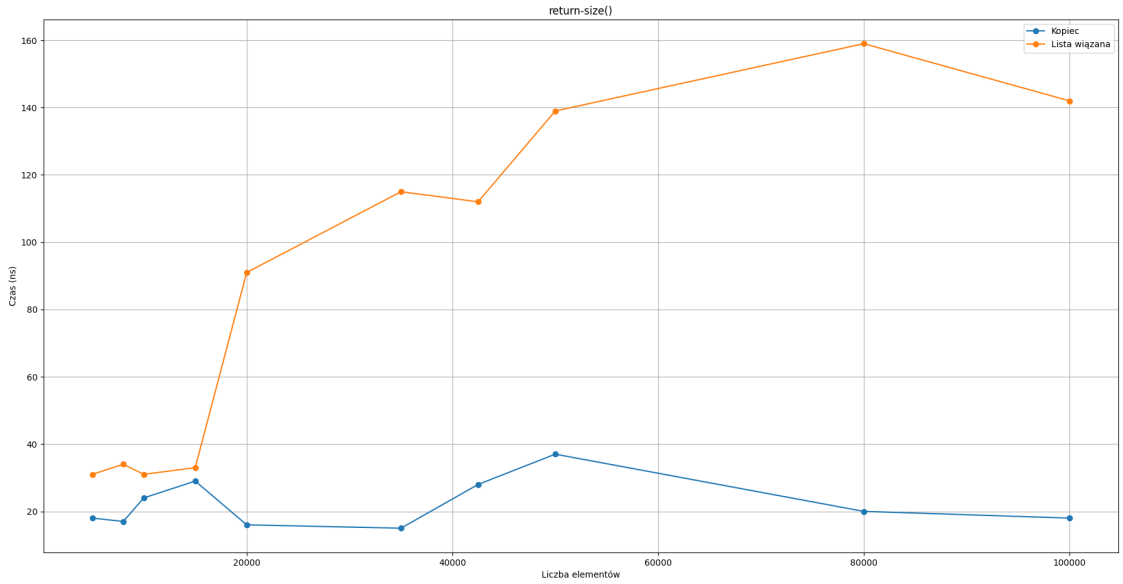


Rysunek 6: Zmodyfikonanie priorytetu elementu o losowym indeksie - skala logarytmiczna na osi x i y

3.5 Operacja return-size

Tabela 6: Wyniki dla metody return-size()

Liczba elementów	**Kopiec [ns]**	**Lista wiązana [ns]**
5,000	18	31
8,000	17	34
10,000	24	31
15,000	29	33
20,000	16	91
35,000	15	115
42,500	28	112
50,000	37	139
80,000	20	159
100,000	18	142



Rysunek 7: Zwrócenie rozmiaru

4 Wnioski

- Struktury zachowują się podobnie do złożoności teoretycznych.
- Implementacja kolejki priorytetowej na liście wiązanej jest dużo wolniejsza od implementacji na kopcu.
- Kopiec ze względu na swoje własności idealnie nadaje się do implementacji kolejki priorytetowej i znacząco przyspiesza działanie struktury.
- Dla operacji extract-max(),find-max(),returnsize oraz pojawiają się zakłócenia, które mogą wynikać z obciążenia szprzętu użytego do przeprowadzenia badań lub niewystarczającej liczby wykonanych pomiarów dla większych rozmiarów struktur. Ponadto dla find-max() może wynikać to z użycia couta w funkcji.

5 Źródła

Źródła wykorszystane do złożoności teoretycznej

- `https://kam.pwr.edu.pl/jaroslaw-rudypwr-edu-pl/files/sd/w4.pdf`
- `https://kam.pwr.edu.pl/jaroslaw-rudypwr-edu-pl/files/sd/w5.pdf`
- `https://en.wikipedia.org/wiki/Priority_queue`
- `https://www.geeksforgeeks.org/priority-queue-set-1-introduction/`
- `https://www.geeksforgeeks.org/applications-priority-queue/`
- `https://www.bigocheatsheet.com`