# Struktury danych i złożoność obliczeniowa
# Projekt
23/03/2021

252713 Tomasz Rzymyszkiewicz

(1) Struktury i operacje

## Spis treści

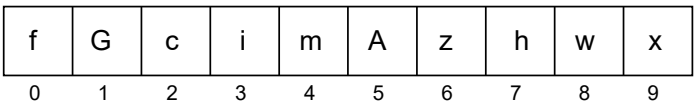
# 1 Sformułowanie zadania

Zadanie ma na celu badawcze sprawdzenie złożoności czasowej wykonania podstawowych operacji (utwórz, wstaw, dodaj, usuń, wyszukaj) w strukturach danych takich jak:

- tablica (struktura statyczna)
- lista (struktura dynamiczna)
- stos (LIFO)
- kolejka (FIFO)

## Tablica

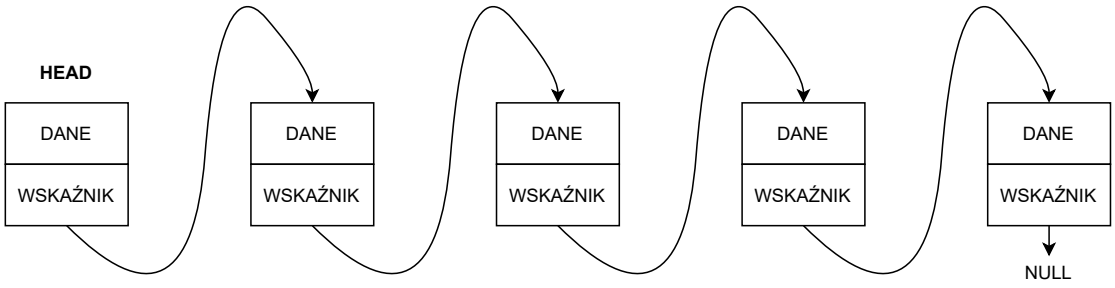
Jest to statyczna struktura danych (tzn. o stałym rozmiarze $n$) umożliwiająca przechowywanie danych stałego typu. W tablicy każdy element posiada swój indeks od 0 do $n-1$.




Rysunek 1: Tablica przechowująca 10 elementów typu znakowego.

## Lista

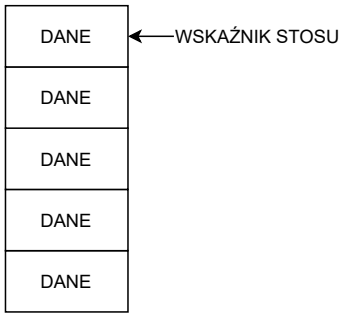
Jest to dynamiczna struktura danych (tzn. o zmieniającym się rozmiarze) połączonych ze sobą węzłów. Każdy z węzłów zawiera daną oraz wskaźnik do kolejnego węzła. Pierwszy węzeł listy jest wskaźnikiem na pierwszy węzeł przechowujący dane. Ostatni węzeł zawiera dane, ale nie wskazuje na kolejny węzeł.




Rysunek 2: Lista przechowująca 4 elementy.

## Stos

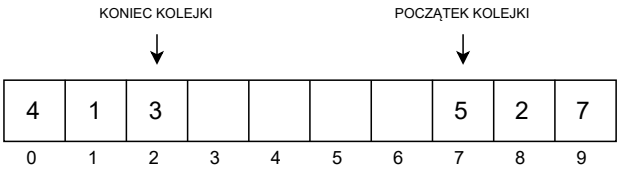
Jest to liniowa struktura danych LIFO (ang. *Last In First Out*), w której możliwy jest dostęp tylko do elementu, który został dodany jako ostatni. Aby dostać się do danych dodanych wcześniej, należy zdjąć ze stosu dane dodane później. Wierzchołek stosu jest wskazywany przez wskaźnik stosu. Operację dodawania elementu do stosu nazywa się z angielskiego *push*, a operację zdejmowania elementu ze stosu *pop*.




Rysunek 3: Stos przechowujący 5 elementów, wskaźnik stosu wskazuje na element na wierzchołku stosu.

## Kolejka

Jest to liniowa struktura danych w której wykorzystywane są oba końce struktury. Odbywa się to zgodnie z zasadą FIFO (ang. *First In First Out*), gdzie elementy dodawane wstawiane są na koniec kolejki, a elementy pobierane zabierane są z początku kolejki. Kolejka może być zaimplementowana z wykorzystaniem tablicy. Tablica może być cykliczna, tzn. kolejne elementy kolejki napotykające na koniec tablicy, będą występowały na początku tablicy.




Rysunek 4: Kolejka przechowująca 6 elementów zaimplementowana z wykorzystaniem tablicy umożliwiającej przechowanie 10 elementów.

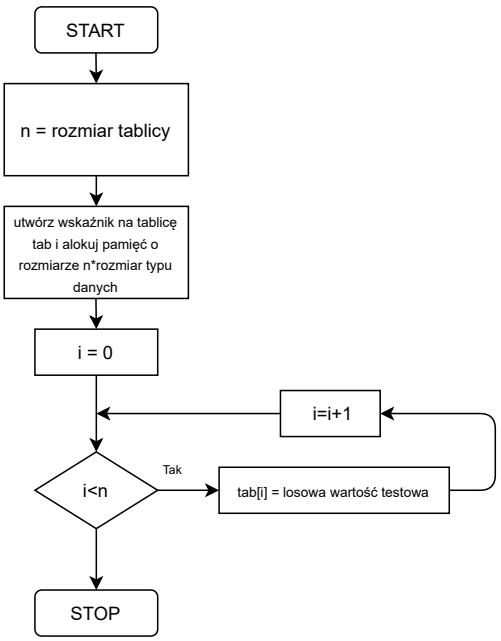
# 2 Algorytmy

Badania złożoności czasowej zostały wykonane z wykorzystaniem programu w języku C++. Operacje na strukturach danych zostały zaimplementowane z wykorzystaniem algorytmów.

## Tablica

### Utwórz

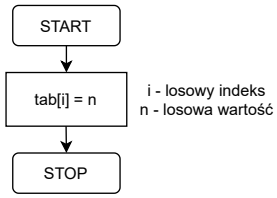
Algorytm tworzenia tablicy wypełnionej danymi testowymi można opisać za pomocą schematu blokowego.




Rysunek 5: Schemat blokowy algorytmu tworzenia tablicy składającej się z $n$ elementów zawierającej losowe dane testowe.

### Wstaw

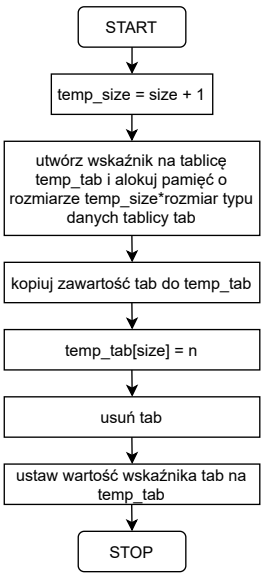
Algorytm wstawienia elementu do tablicy można opisać za pomocą schematu blokowego. Na potrzeby testu zakładamy, że wstawiamy losową wartość w losowe miejsce (w zakresie pojemności tablicy) i mamy te wartości ustalone przed wykonaniem algorytmu wstawiania.




Rysunek 6: Schemat blokowy algorytmu wstawienia losowej wartości $n$ w losowe miejsce o indeksie $i$ w tablicy.

### Dodaj

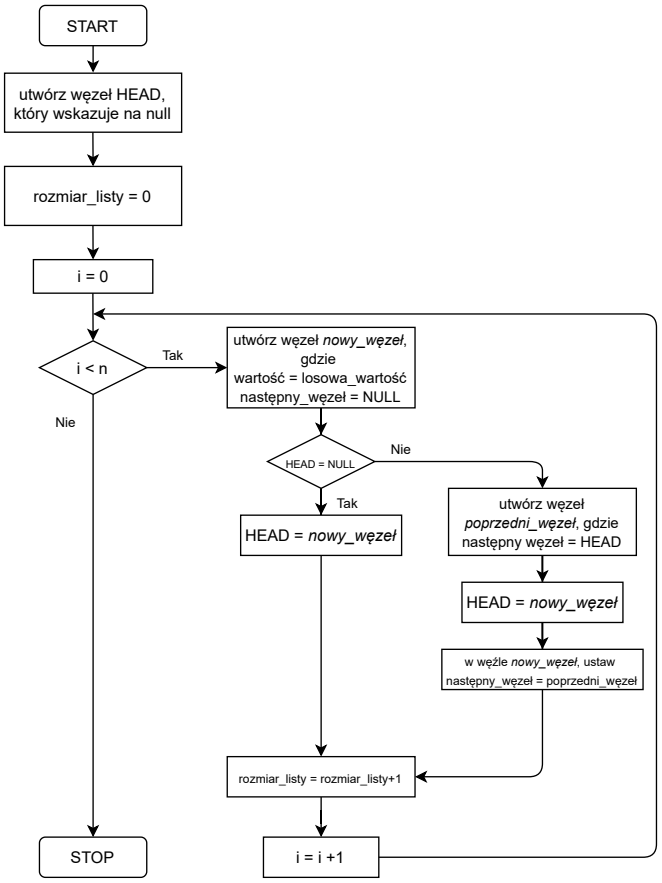
Algorytm dodawania elementu do tablicy można opisać za pomocą schematu blokowego. Na potrzeby testu zakładamy, że dodajemy element do tablicy *tab* o rozmiarze *size*, a wartość dodawana to $n$.




Rysunek 7: Schemat blokowy algorytmu dodawania elementu do tablicy.

### Usuń

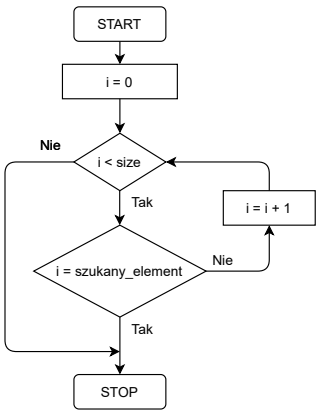
Algorytm usuwania elementu z tablicy można opisać za pomocą schematu blokowego. Na potrzeby testu zakładamy, że usuwamy ostatni element tablicy *tab* o rozmiarze *size*.




Rysunek 8: Schemat blokowy algorytmu usuwania elementu z tablicy.

### Wyszukaj

Algorytm wyszukiwania elementu w tablicy można opisać za pomocą schematu blokowego. Na potrzeby testu zakładamy, że szukamy elementu *szukany_element* w tablicy *tab* o rozmiarze *size*.

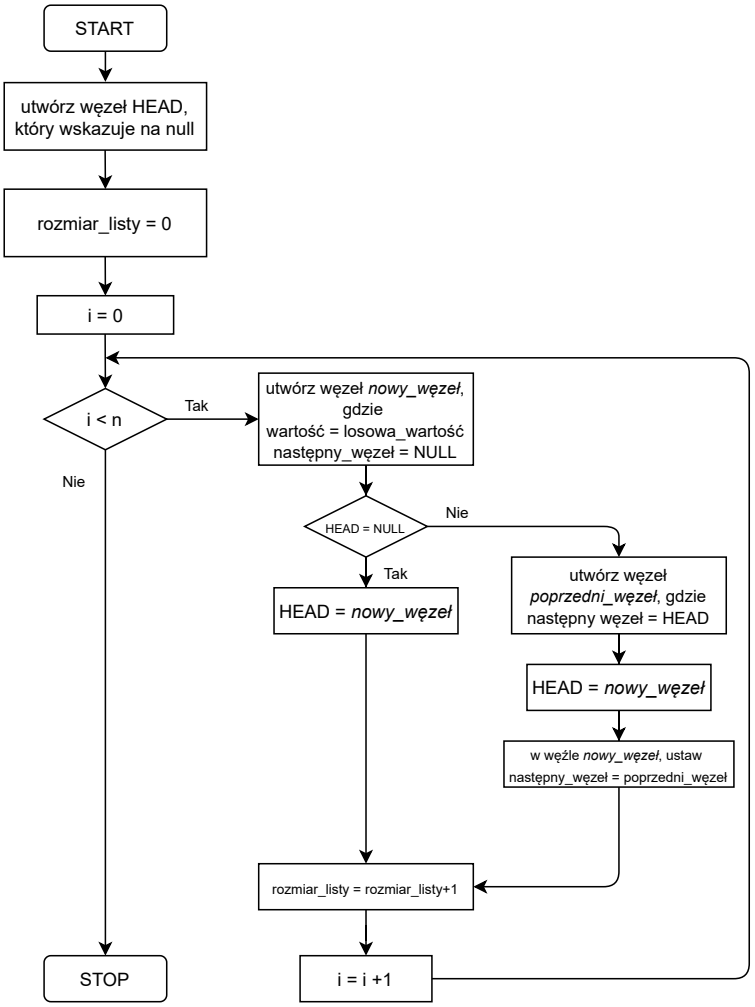



Rysunek 9: Schemat blokowy algorytmu wyszukiwania elementu w tablicy.

# Lista

## Utwórz

Algorytm tworzenia listy można opisać za pomocą schematu blokowego. Na potrzeby testu zakładamy, że tworzymy listę zawierającą $n$ elementów.



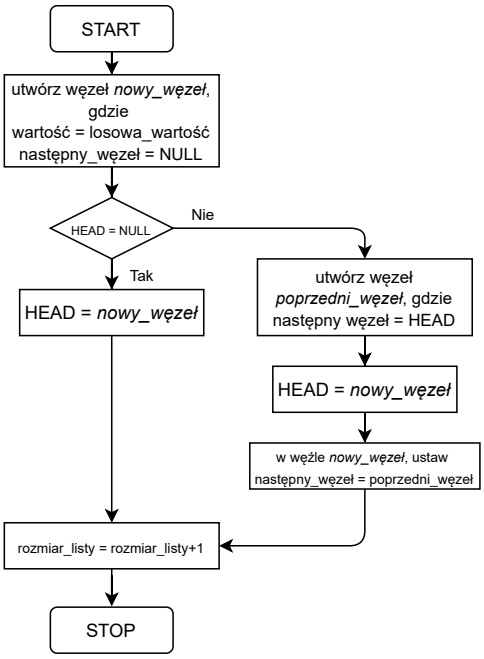

Rysunek 10: Schemat blokowy algorytmu tworzenia listy.

## Wstaw

W tym projekcie operacja wstawiania elementu do listy nie jest uwzględniona. Przy strukturze listy ciężko określić na czym miałby polegać wstawianie elementu.

### Dodaj

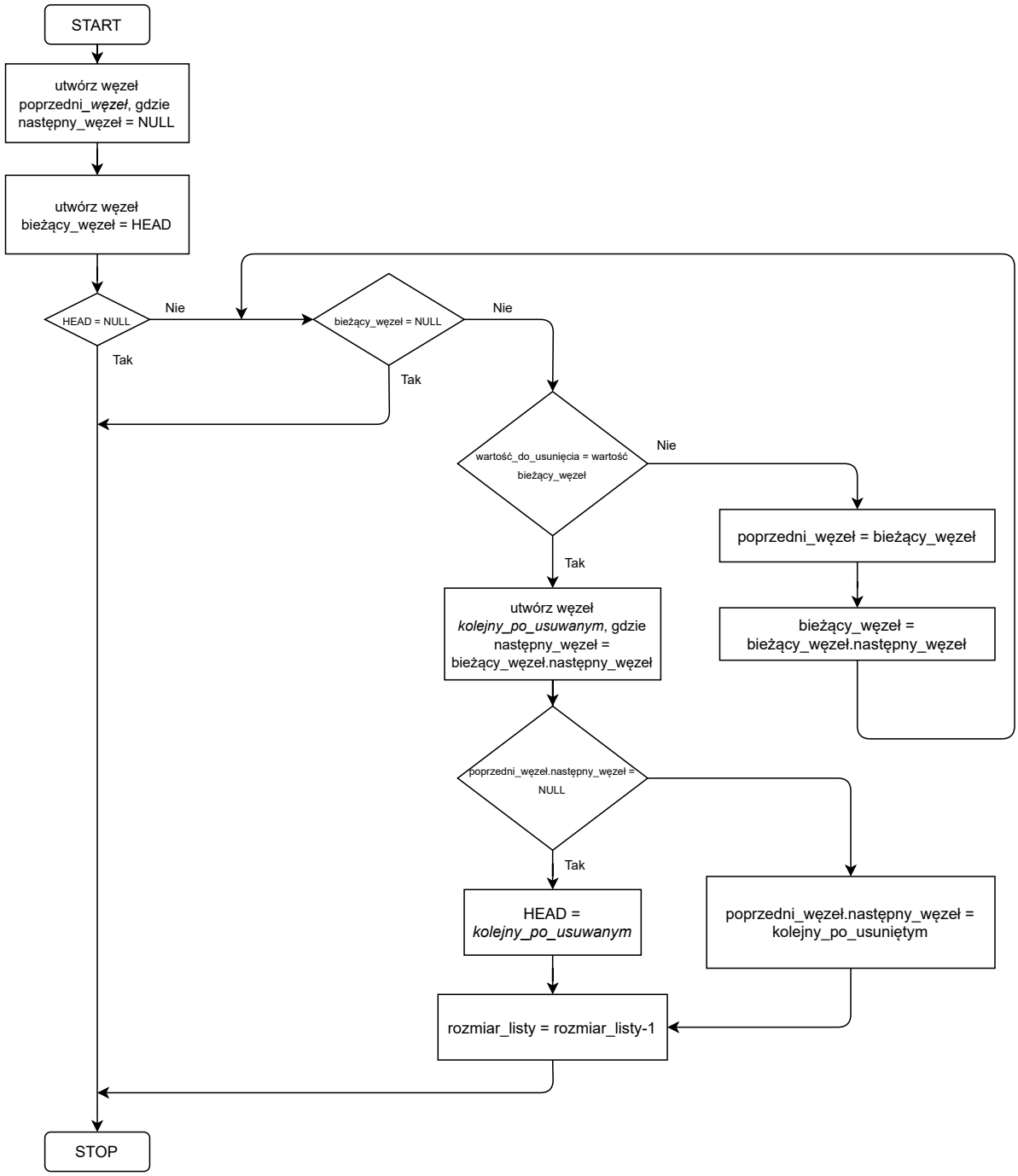
Algorytm dodawania elementu listy można opisać za pomocą schematu blokowego. Na potrzeby testu zakładamy, że dodajemy element o nazwie *losowa_wartosc*.




Rysunek 11: Schemat blokowy algorytmu dodawania elementu do listy.

### Usuń

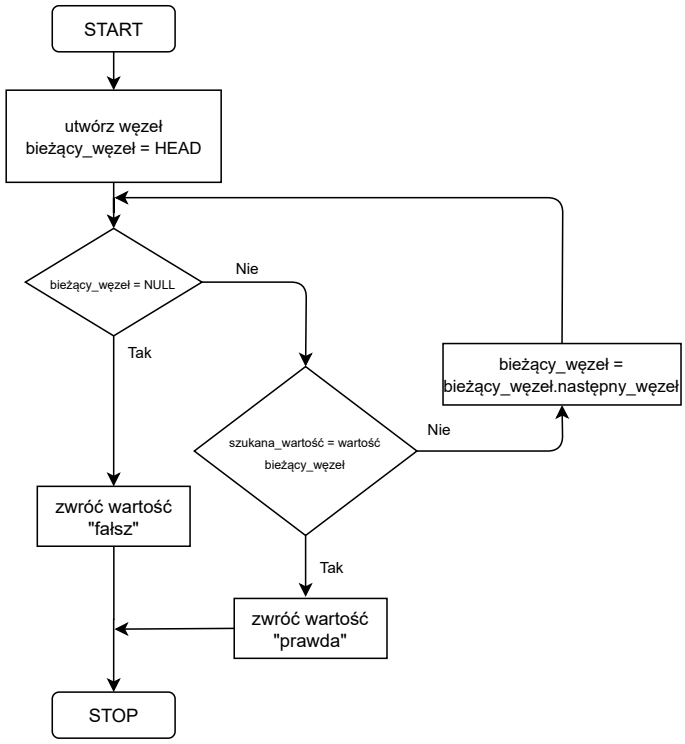
Algorytm usuwania elementu listy można opisać za pomocą schematu blokowego. Na potrzeby testu zakładamy, że usuwamy element o nazwie *wartosc_do_usuniecia*.




Rysunek 12: Schemat blokowy algorytmu usuwania elementu do listy.

### Wyszukaj

Algorytm wyszukiwania elementu listy można opisać za pomocą schematu blokowego. Na potrzeby testu zakładamy, że szukamy elementu o nazwie *szukany_element*.

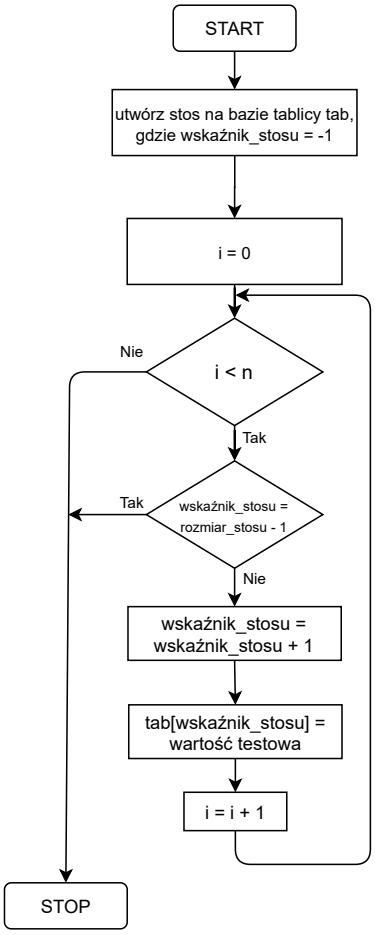



Rysunek 13: Schemat blokowy algorytmu wyszukiwania elementu w liście.

# Stos

## Utwórz

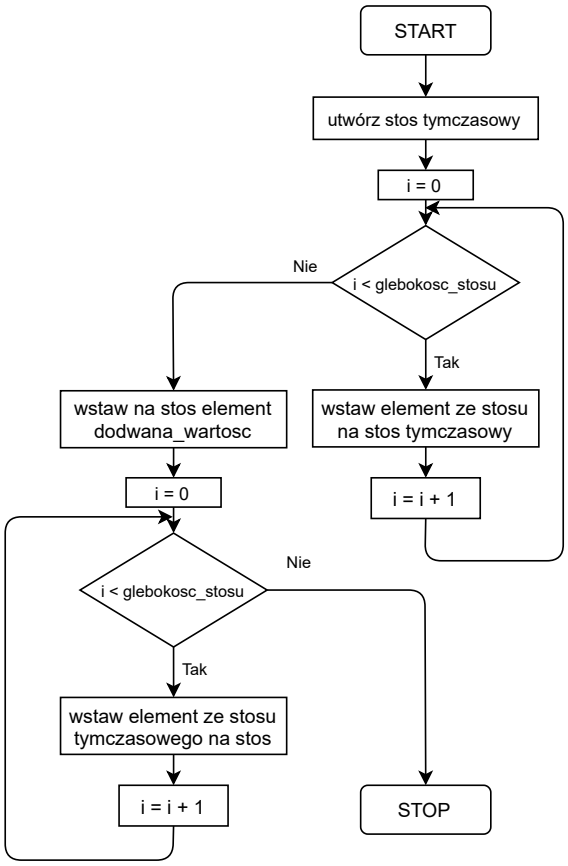
Algorytm tworzenia stosu można opisać za pomocą schematu blokowego. Na potrzeby testu zakładamy, że tworzymy stos zawierający $n$ elementów.




Rysunek 14: Schemat blokowy algorytmu tworzenia stosu zawierającego $n$ elementów.

**Wstaw**

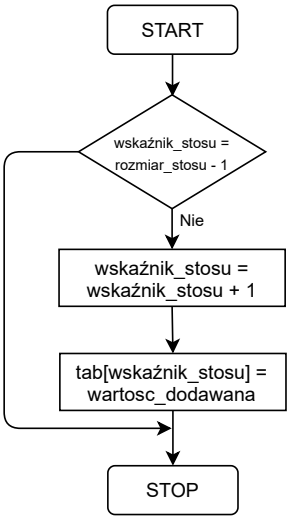
Algorytm wstawiania elementu na stos można opisać za pomocą schematu blokowego. Na potrzeby testu zakładamy, operacja wstawienia elementu polega na umieszczeniu wartości *dodawana_wartosc* na zadanej głębokości stosu (oznaczonej jako *glebokosc_stosu*), na przykład w połowie.




Rysunek 15: Schemat blokowy algorytmu wstawiania elementu *dodawana_wartosc* stosu na głębokość *glebokosc_stosu*.

### Dodaj (push)

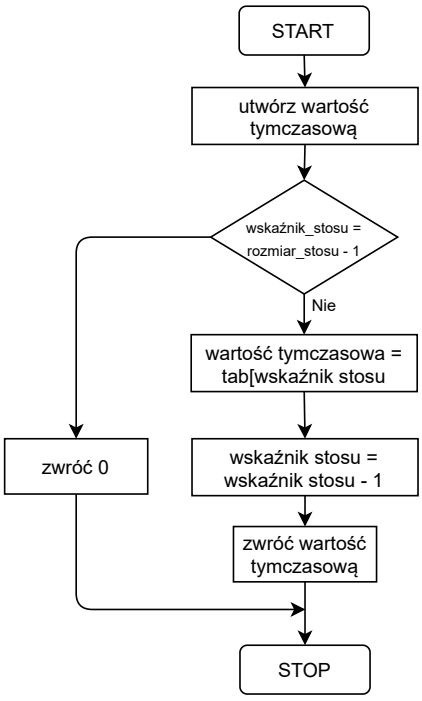
Algorytm dodawania elementu na stos (ang. *push*) można opisać za pomocą schematu blokowego. Na potrzeby testu zakładamy, że dodajemy element $wartosc_dodawana$.




Rysunek 16: Schemat blokowy algorytmu dodawania (*push*) elementu $wartosc_dodawana$ na stos.

### Usuń (pop)

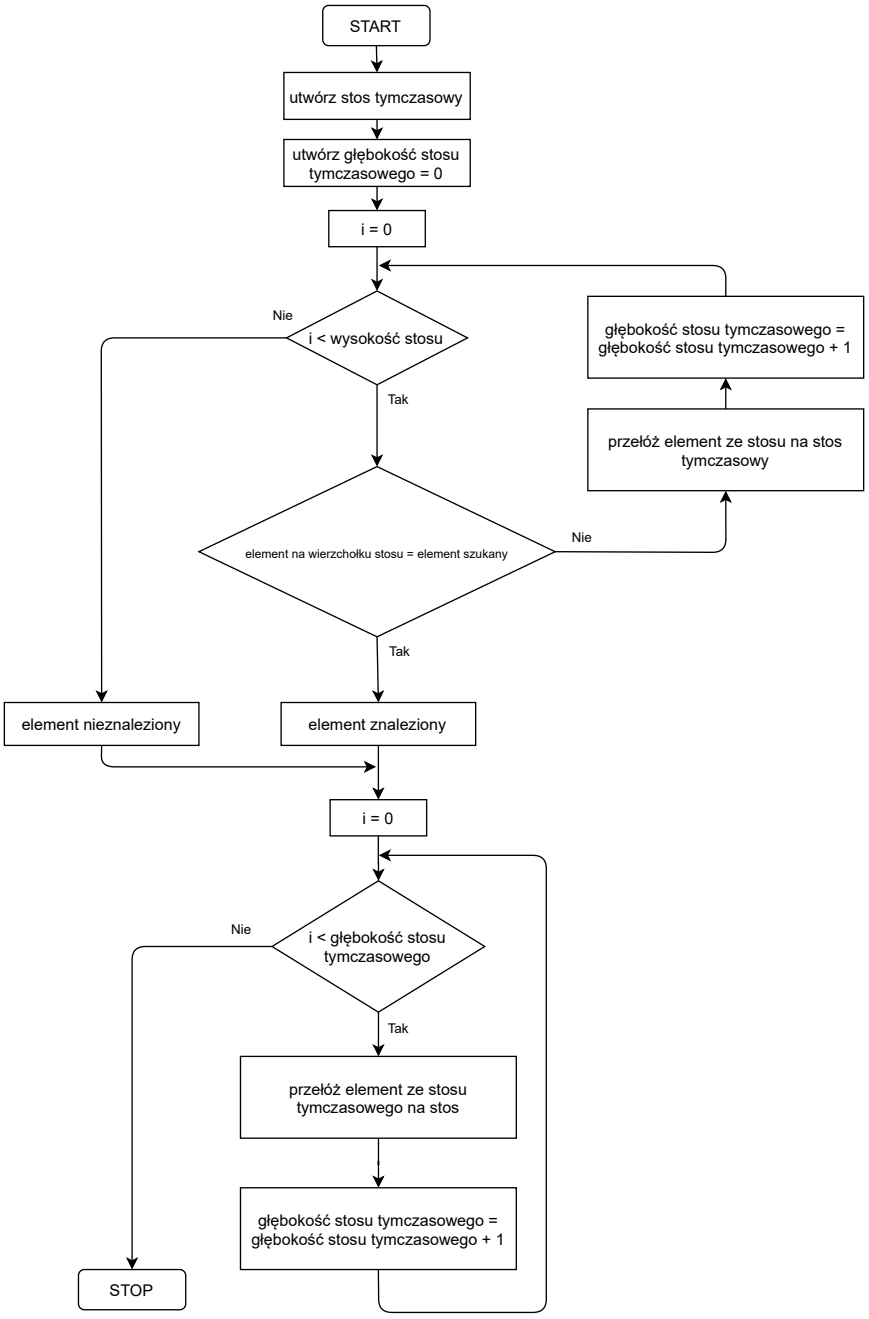
Algorytm usuwania elementu ze stos (ang. *pop*) można opisać za pomocą schematu blokowego.




Rysunek 17: Schemat blokowy algorytmu usuwania (*pop*) elementu ze stos.

### Wyszukaj

Algorytm wyszukiwania elementu na stos można opisać za pomocą schematu blokowego. Na potrzeby testu zakładamy, że szukany element to *element_szukany*.

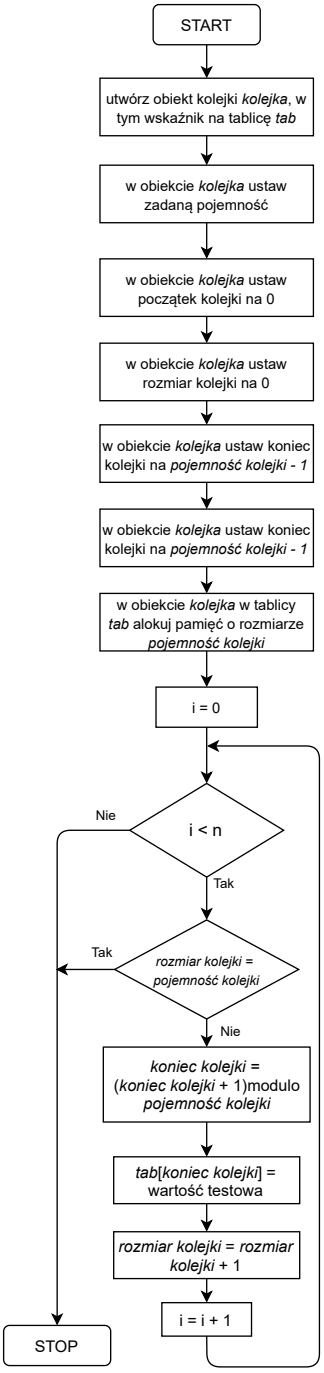



Rysunek 18: Schemat blokowy algorytmu wyszukiwania elementu na stosie.

# Kolejka

## Utwórz

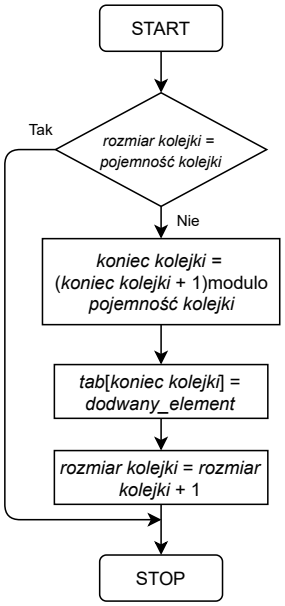
Algorytm tworzenia kolejki można opisać za pomocą schematu blokowego. Na potrzeby testu zakładamy, że tworzymy kolejkę o pojemności *pojemnosc_kolejki* i wypełniamy ją danymi testowymi.




Rysunek 19: Schemat blokowy algorytmu tworzenia kolejki.

### Dodaj (enqueue)

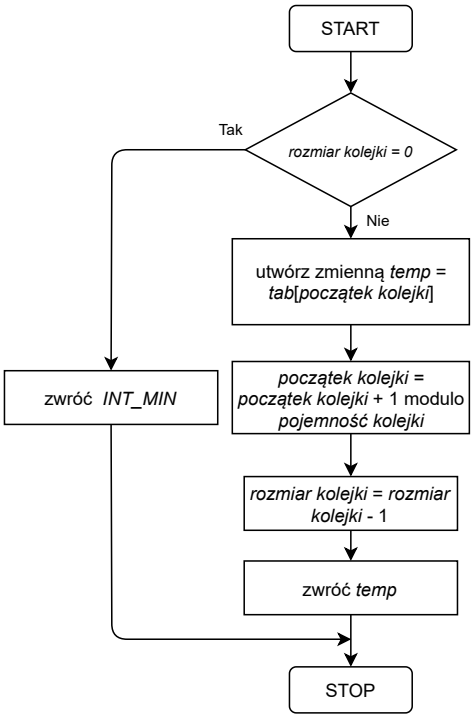
Algorytm dodawania elementu do kolejki (ang. *enqueue*) można opisać za pomocą schematu blokowego. Na potrzeby testu zakładamy, że dodajemy element *dodawany_element*.




Rysunek 20: Schemat blokowy algorytmu dodawania elementu do kolejki.

### Usuń (dequeue)

Algorytm usuwania elementu do kolejki (ang. *dequeue*) można opisać za pomocą schematu blokowego. Na potrzeby testu zakładamy, że usuwamy element *usuwany_element*.



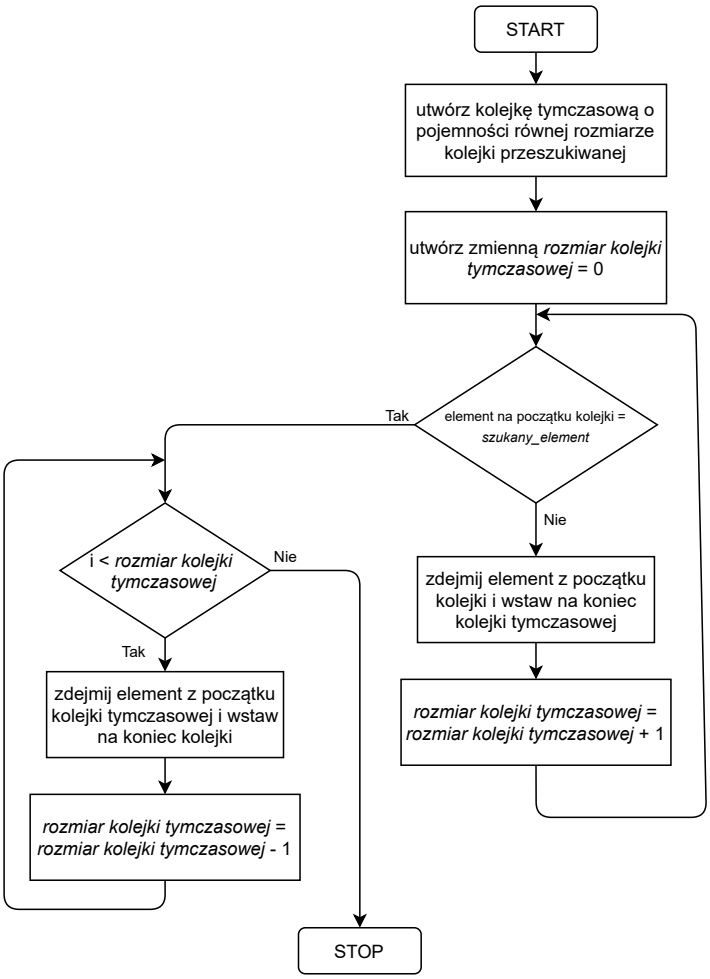

Rysunek 21: Schemat blokowy algorytmu usuwania elementu z kolejki.

### Wstaw

W tym projekcie operacja wstawiania elementu do kolejki nie jest uwzględniona. Przy strukturze kolejki ciężko określić na czym miałby polegać wstawianie elementu.

### Wyszukaj

Algorytm wyszukiwania elementu w kolejce można opisać za pomocą schematu blokowego. Na potrzeby testu zakładamy, że szukany element to *szukany_element*.




Rysunek 22: Schemat blokowy algorytmu usuwania elementu z kolejki.

# 3 Dane wejściowe i wyjściowe

## Plik konfiguracyjny `config.ini`

W celu prawidłowego uruchomienia programu mierzącego czasy wykonywania operacji na strukturach danych należy prawidłowo skonfigurować plik `config.ini`.

W pierwszej linii pliku konfiguracyjnego należy podać nazwę pliku CSV zawierającego dane testowe oraz, po spacji, liczbę danych które chcemy wczytać do pamięci programu. Jeżeli podany plik nie będzie istniał, program wygeneruje dane i zapisze je w pliku o tej samej nazwie. Jeżeli w podanym pliku będzie się znajdowała niewystarczająca ilość danych (mniejsza niż podana liczba danych do wczytania), program nadpisze plik z odpowiednią liczbą wygenerowanych danych.

W drugiej linii należy podać nazwę pliku (z rozszerzeniem .csv) do którego zostaną zapisane wyniki działania programu. Jeżeli plik o podanej nazwie już istnieje, zostanie on nadpisany.

W kolejnych liniach należy podawać kolejne zadania do wykonania przez program. Każda linia oznacza osobne zadanie. Zadania powinny być opisywane w następującej składni:
`struktura operacja start stop krok powtórzenia`
gdzie:

- `struktura` - struktura na której będzie wykonana operacja
- `operacja` - operacja, która będzie wykonana na strukturze danych
- `start` - najmniejszy rozmiar instancji na której będzie wykonana operacja
- `stop` - największy rozmiar instancji na której będzie wykonana operacja
- `krok` - wartość o którą zwiększany będzie rozmiar instancji w zakresie od start do stop
- `powtórzenia` - liczba razy ile ma być powtórzona operacja na instancji

Przykładowy plik `config.ini`:

```
dane.csv 10000000
wyniki_operacji.csv
stack search 1 10 1 1000
array create 10000 100000 100 10
list delete 100 1000 10 100
queue dequeue 100 1000 1 1
```

Możliwe struktury i operacje:

- array - create, put, add, delete, search
- list - create, add, delete, search
- stack - create, put, push, pop, search
- queue - create, enqueue, dequeue, search

## Dane wejściowe

Dane wejściowe to zbiór liczb całkowitych typu Integer zapisanych w pliku csv. W każdej linii zapisana jest kolejna liczba. Program podczas generowania danych testowych domyślnie generuje liczby z zakresu od -1000000 do 1000000. Każda badana struktura była wypełniona danymi ze zbioru testowego.

## Dane wyjściowe

Dane wyjściowe zapisywane są do pliku csv jako zbiór wierszy, gdzie każdy wiersz oznacza wykonanie operacji na danej instancji określoną ilość razy. Oznacza to, że jeśli nakazaliśmy programowi wykonać operację 100 razy to czas operacji jest równy czasowy wykonania 100 takich operacji na danej instancji. W celu uzyskania czasu wykonania pojedynczej operacji należy podzielić czas wykonania wszystkich powtórzeń przez ilość powtórzeń. Zabieg ten pozwala lepiej zmierzyć czasy wykonywania bardzo szybkich operacji, w przypadku których pomiar pojedynczego wykonania jest zbyt mały, mniejszy niż skala jednostki pomiarowej.

Wyniki operacji poprzedzone są nagłówkiem:
`data_structure,operation,size_of_structure,time_of_operation_s,number_of_repetitions`
gdzie:

- `data_structure` (struktura danych) - struktura na której wykonana była operacja
- `operation` (operacja) - operacja, która była wykonana na strukturze danych
- `size_of_structure` (rozmiar struktury) - rozmiar instancji na której przeprowadzana była operacja
- `time_of_operation_s` (czas operacji) - czas (w sekundach) wykonania wszystkich powtórzeń operacji z dokładnością do 1 milisekundy
- `number_of_repetitions` (liczba powtórzeń) - liczba powtórzeń operacji

# 4 Procedura badawcza

Badania złożoności czasowej wykonywania operacji zostały przeprowadzone z wykorzystaniem algorytmów zaimplementowanych w języku C++. Struktury danych inne niż tablica zostały zaimplementowane na potrzeby badania (z wykorzystaniem tablicy) w celu zagwarantowania zgodności działania programu z zaimplementowanymi algorytmami.
Pomiar czasu wykonywania operacji został wykonany przy pomocy klasy `std::chrono::high_resolution_clock`, gdzie do pobrania bieżącego czasu wykorzystana została metoda `std::chrono::high_resolution_clock::now()`.
Czas wykonywania operacji był zapisywany w strukturze `std::chrono::duration<double>`.

## Tablica

### Utwórz

Badanie wpływu rozmiaru instancji na czas wykonywania operacji tworzenia tablicy podzielono na dwa etapy. W pierwszym etapie pomiar został wykonany w zakresie od 10 do 1000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 10000 razy. Zależność została pokazana na rysunku 23. W drugim etapie pomiar został wykonany w zakresie od 1000000 do 10000000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 100 razy. Zależność została pokazana na rysunku 24.

### Wstaw

Badanie wpływu rozmiaru instancji na czas wykonywania operacji wstawiania elementu do tablicy podzielono na dwa etapy. W pierwszym etapie pomiar został wykonany w zakresie od 10 do 1000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 10000000 razy. Zależność została pokazana na rysunku 25. W drugim etapie pomiar został wykonany w zakresie od 1000000 do 10000000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 10000000 razy. Zależność została pokazana na rysunku 26.

### Dodaj

Badanie wpływu rozmiaru instancji na czas wykonywania operacji dodawania elementu do tablicy podzielono na dwa etapy. W pierwszym etapie pomiar został wykonany w zakresie od 10 do 1000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 100000 razy. Zależność została pokazana na rysunku 27. W drugim etapie pomiar został wykonany w zakresie od 1000000 do 10000000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 100 razy. Zależność została pokazana na rysunku 28.

### Usuń

Badanie wpływu rozmiaru instancji na czas wykonywania operacji usuwania elementu z tablicy podzielono na dwa etapy. W pierwszym etapie pomiar został wykonany w zakresie od 10 do 1000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 100000 razy. Zależność została pokazana na rysunku 29. W drugim etapie pomiar został wykonany w zakresie od 1000000 do 10000000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 100 razy. Zależność została pokazana na rysunku 30.

### Wyszukaj

Badanie wpływu rozmiaru instancji na czas wykonywania operacji usuwania elementu z tablicy podzielono na dwa etapy. W pierwszym etapie pomiar został wykonany w zakresie od 10 do 1000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 100000 razy. Zależność została pokazana na rysunku 31. W drugim etapie pomiar został wykonany w zakresie od 1000000 do 10000000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 100 razy. Zależność została pokazana na rysunku 32.

## Lista

### Utwórz

Badanie wpływu rozmiaru instancji na czas wykonywania operacji tworzenia listy podzielono na dwa etapy. W pierwszym etapie pomiar został wykonany w zakresie od 10 do 1000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 10000 razy. Zależność została pokazana na rysunku 33. W drugim etapie pomiar został wykonany w zakresie od 1000000 do 10000000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 10 razy. Zależność została pokazana na rysunku 34.

### Dodaj

Badanie wpływu rozmiaru instancji na czas wykonywania operacji dodawania elementu listy zostało wykonane w zakresie od 10 do 1000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 1000000 razy. Ze względu na brak optymalizacji programu nie udało się przeprowadzić pomiarów dla większych instancji. Zależność została pokazana na rysunku 35.

### Usuń

Badanie wpływu rozmiaru instancji na czas wykonywania operacji dodawania elementu listy zostało wykonane w zakresie od 10 do 1000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 10000 razy. Ze względu na brak optymalizacji programu nie udało się przeprowadzić pomiarów dla większych instancji. Zależność została pokazana na rysunku 36.

### Wyszukaj

Badanie wpływu rozmiaru instancji na czas wykonywania operacji dodawania elementu listy zostało wykonane w zakresie od 100 do 10000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 100000 razy. Ze względu na brak optymalizacji programu nie udało się przeprowadzić pomiarów dla większych instancji. Zależność została pokazana na rysunku 37.

## Stos

### Utwórz

Badanie wpływu rozmiaru instancji na czas wykonywania operacji tworzenia stosu podzielono na dwa etapy. W pierwszym etapie pomiar został wykonany w zakresie od 10 do 1000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 1000 razy. Zależność została pokazana na rysunku 38. W drugim etapie pomiar został wykonany w zakresie od 1000000 do 10000000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 100 razy. Zależność została pokazana na rysunku 39.

### Wstaw

Badanie wpływu rozmiaru instancji na czas wykonywania operacji wstawiania elementu do stosu podzielono na dwa etapy. W pierwszym etapie pomiar został wykonany w zakresie od 100 do 10000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 100 razy. Zależność została pokazana na rysunku 40. W drugim etapie pomiar został wykonany w zakresie od 1000000 do 10000000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 100 razy. Zależność została pokazana na rysunku 41.

### Dodaj (push)

Badanie wpływu rozmiaru instancji na czas wykonywania operacji dodawania elementu na stos (*push*) zostało wykonane w zakresie od 10 do 100 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 100000 razy. Ze względu na brak optymalizacji programu nie udało się przeprowadzić pomiarów dla większych instancji. Zależność została pokazana na rysunku 42.

### Usuń (pop)

Badanie wpływu rozmiaru instancji na czas wykonywania operacji usuwania elementu ze stosu (*pop*) zostało wykonane w zakresie od 10 do 100 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 100000 razy. Ze względu na brak optymalizacji programu nie udało się przeprowadzić pomiarów dla większych instancji. Zależność została pokazana na rysunku 43.

### Wyszukaj

Badanie wpływu rozmiaru instancji na czas wykonywania operacji wyszukiwania elementu na stosie zostało wykonane w zakresie od 1000 do 10000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 10000 razy. Ze względu na brak optymalizacji programu nie udało się przeprowadzić pomiarów dla większych instancji. Zależność została pokazana na rysunku 44.

## Kolejka

### Utwórz

Badanie wpływu rozmiaru instancji na czas wykonywania operacji tworzenia kolejki zostało wykonane w zakresie od 100 do 1000 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 10000 razy. Ze względu na brak optymalizacji programu nie udało się przeprowadzić pomiarów dla większych instancji. Zależność została pokazana na rysunku 45.

### Dodaj (enqueue)

Badanie wpływu rozmiaru instancji na czas wykonywania operacji dodawania elementu do kolejki (*enqueue*) zostało wykonane w zakresie od 10 do 100 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 1000000 razy. Ze względu na brak optymalizacji programu nie udało się przeprowadzić pomiarów dla większych instancji. Zależność została pokazana na rysunku 46.

### Usuń (dequeue)

Badanie wpływu rozmiaru instancji na czas wykonywania operacji usuwania elementu z kolejki (*dequeue*) zostało wykonane w zakresie od 10 do 100 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 1000000 razy. Ze względu na brak optymalizacji programu nie udało się przeprowadzić pomiarów dla większych instancji. Zależność została pokazana na rysunku 47.

### Wyszukaj

Badanie wpływu rozmiaru instancji na czas wykonywania operacji wyszukiwania elementu w kolejce zostało wykonane w zakresie od 10 do 100 elementów. Dla zwiększenia dokładności każdy pomiar powtórzono 1000000 razy. Ze względu na brak optymalizacji programu nie udało się przeprowadzić pomiarów dla większych instancji. Zależność została pokazana na rysunku 48.

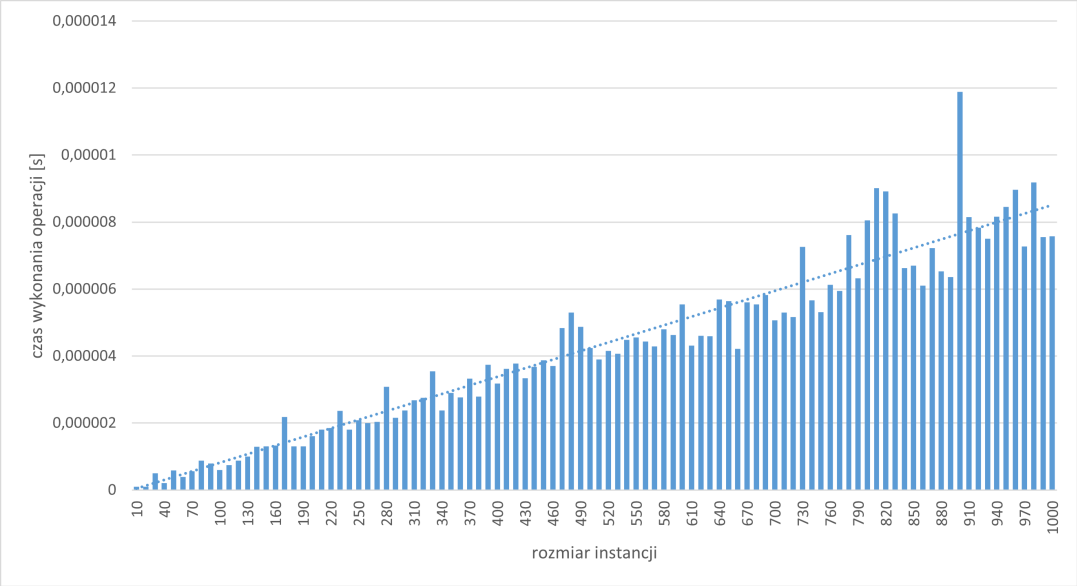
### Metoda analizy zgromadzonych danych

Pliki csv wygenerowane przez program zostały zaimportowane do programu Microsoft Excel. W programie Excel wykorzystując informację o łącznym czasie operacji i liczbie operacji wyznaczyłem poszczególne czasy wykonania operacji dla danych instancji. Następnie z wykorzystaniem tabel przestawnych utworzyłem wykresy.

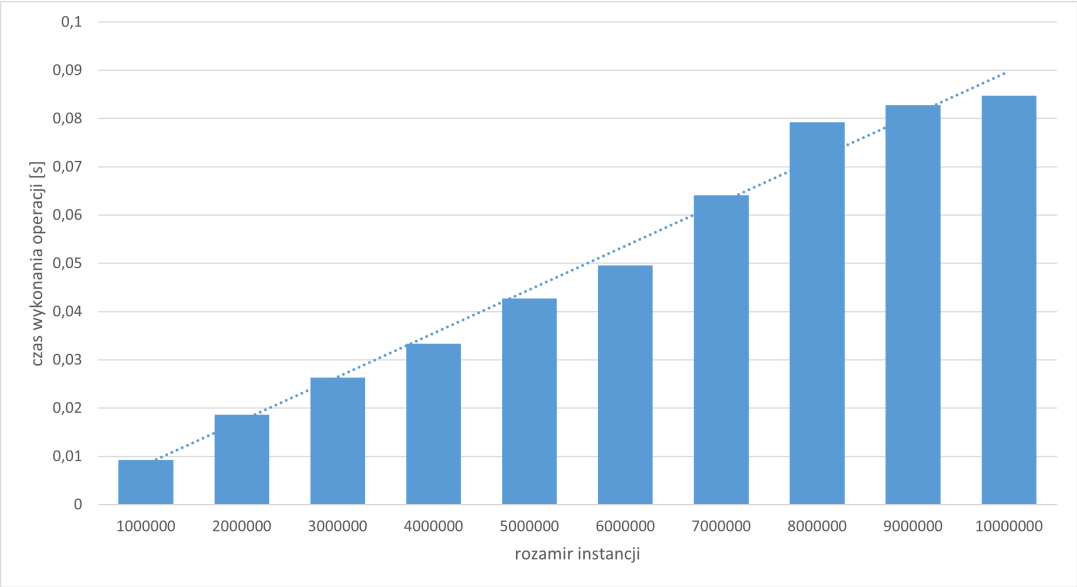
# 5 Wyniki

## Tablica

### Utwórz

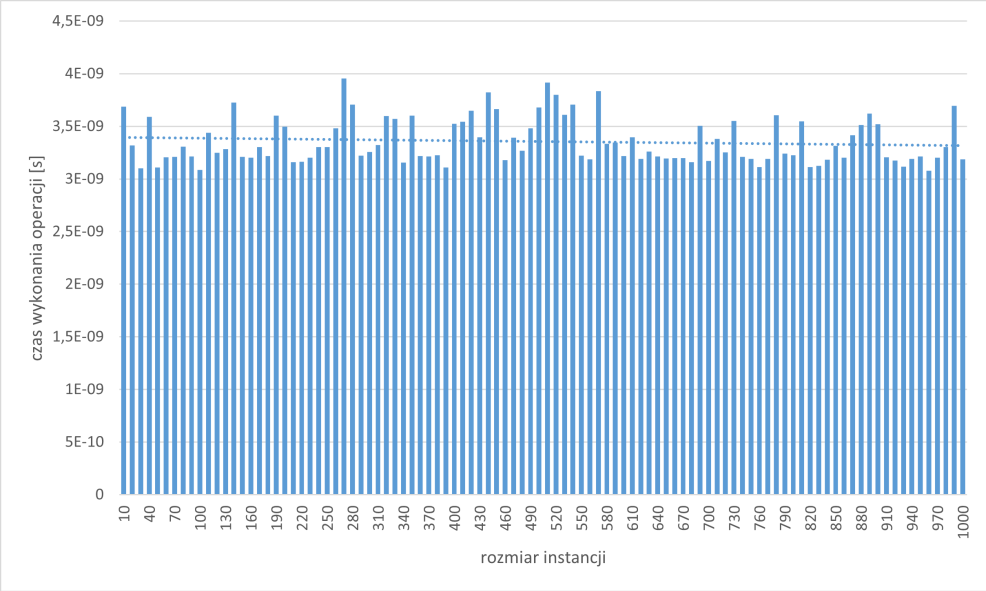



Rysunek 23: Wpływ wielkości instancji (w zakresie 10 - 1000) na czas tworzenia tablicy.

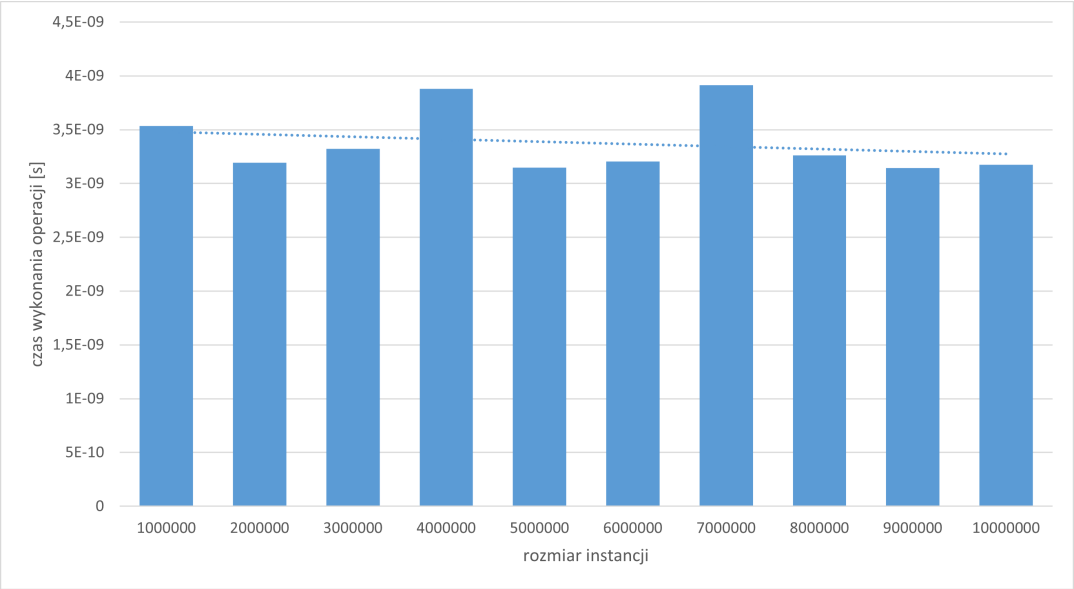



Rysunek 24: Wpływ wielkości instancji (w zakresie 1000000 - 10000000) na czas tworzenia tablicy.

### Wstaw

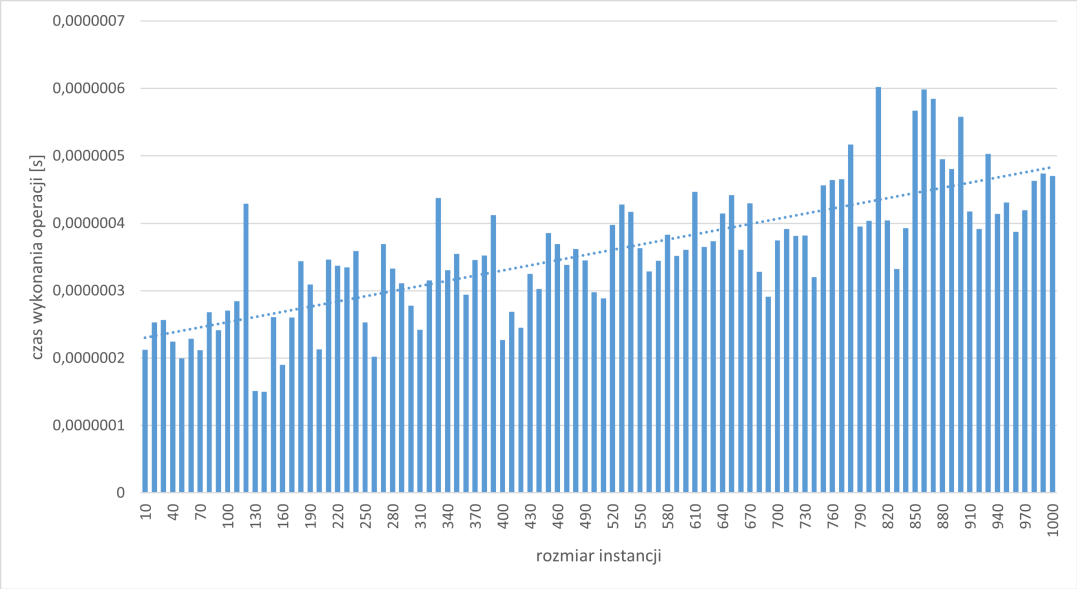



Rysunek 25: Wpływ wielkości instancji (w zakresie 10 - 1000) na czas wstawiania elementu do tablicy.

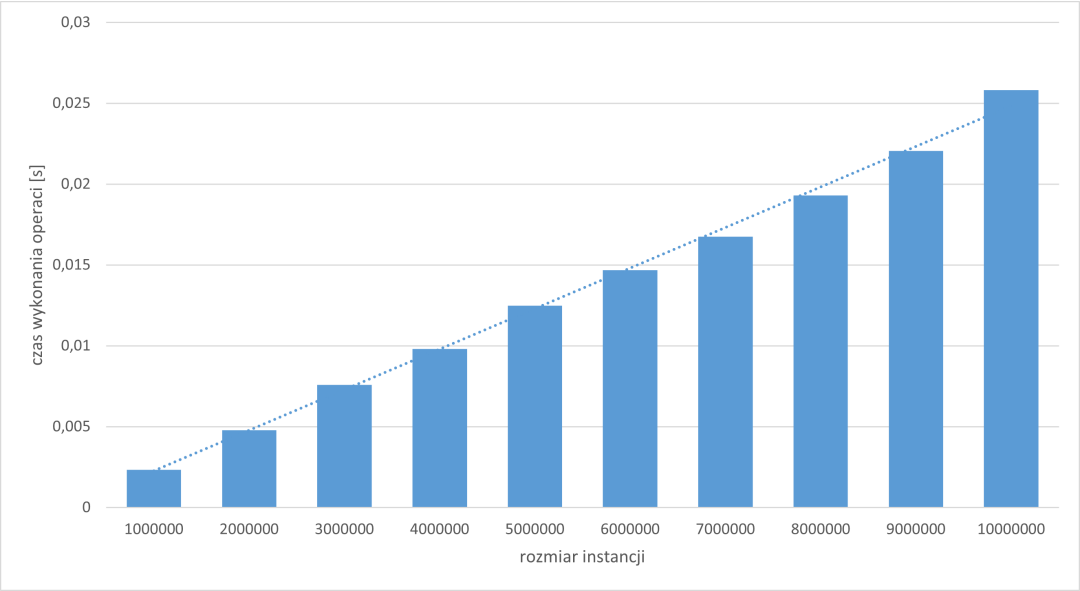



Rysunek 26: Wpływ wielkości instancji (w zakresie 1000000 - 10000000) na czas wstawiania elementu do tablicy.

### Dodaj

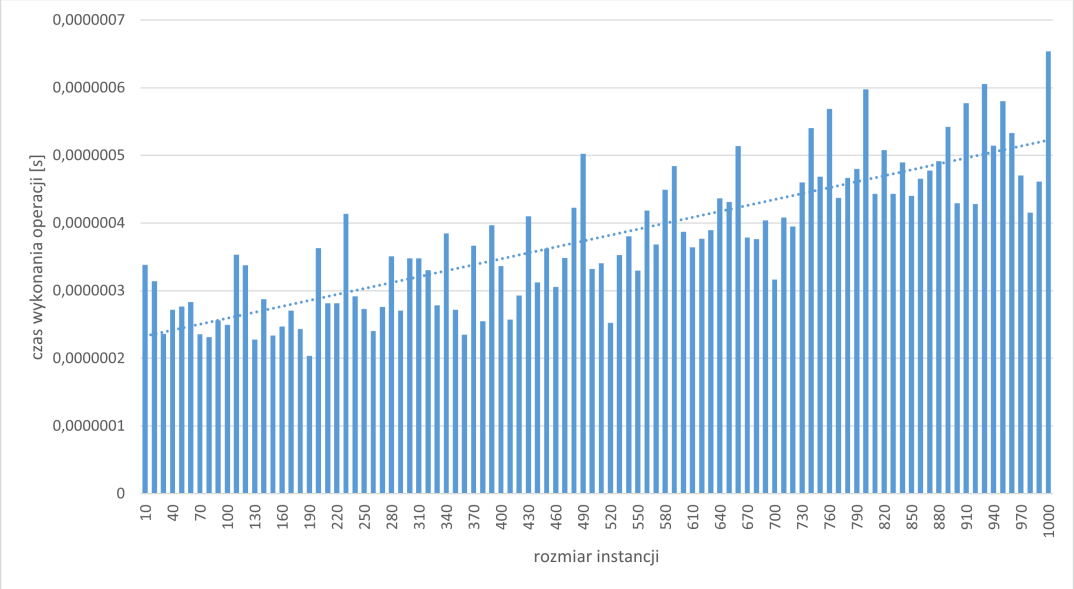



Rysunek 27: Wpływ wielkości instancji (w zakresie 10 - 1000) na czas dodawania elementu do tablicy.

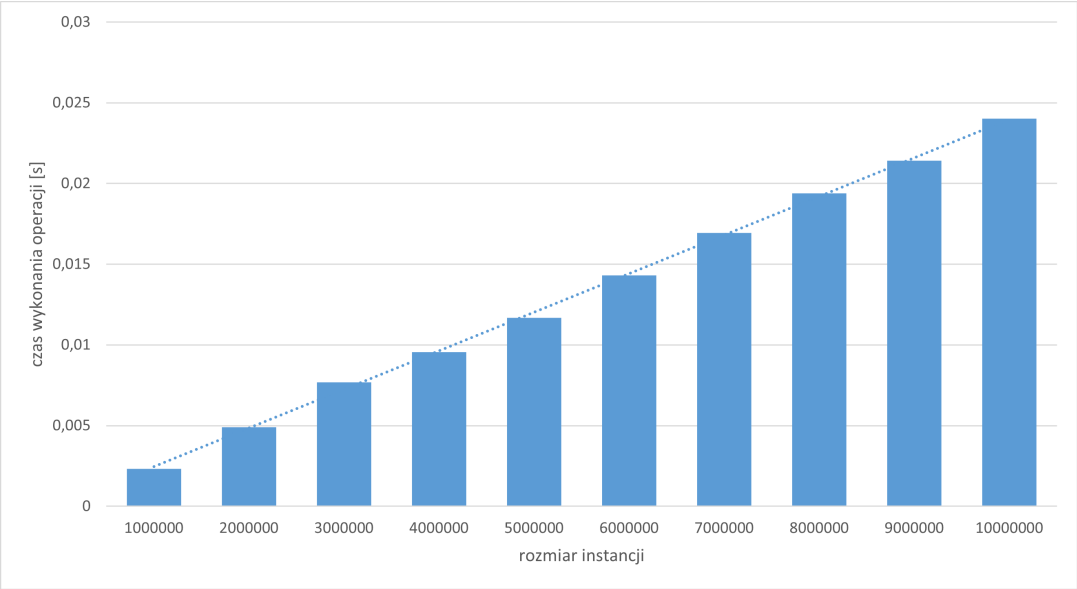



Rysunek 28: Wpływ wielkości instancji (w zakresie 1000000 - 10000000) na czas dodawania elementu do tablicy.

### Usuń

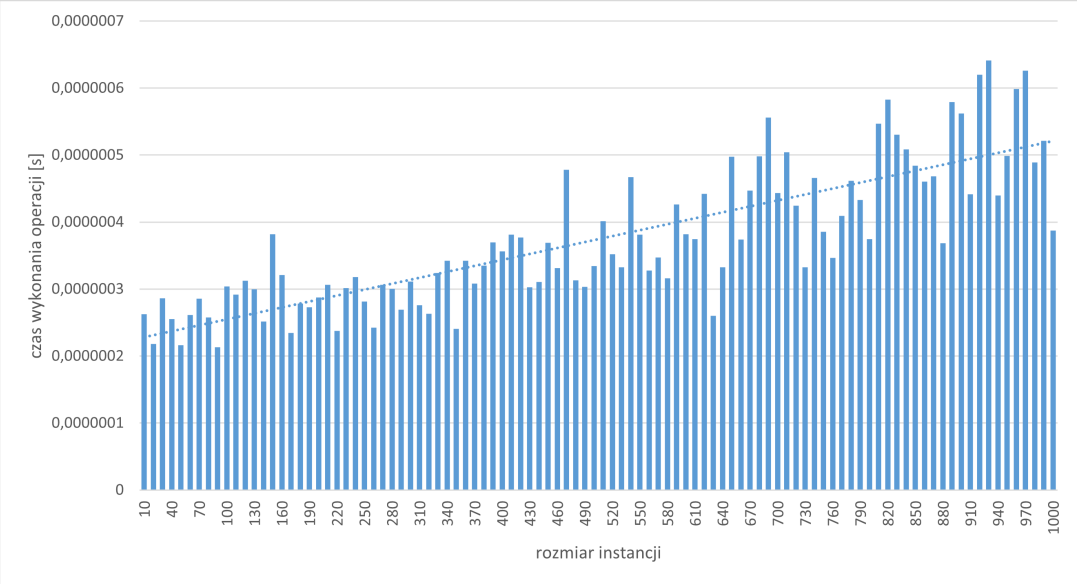



Rysunek 29: Wpływ wielkości instancji (w zakresie 10 - 1000) na czas usuwania elementu z tablicy.

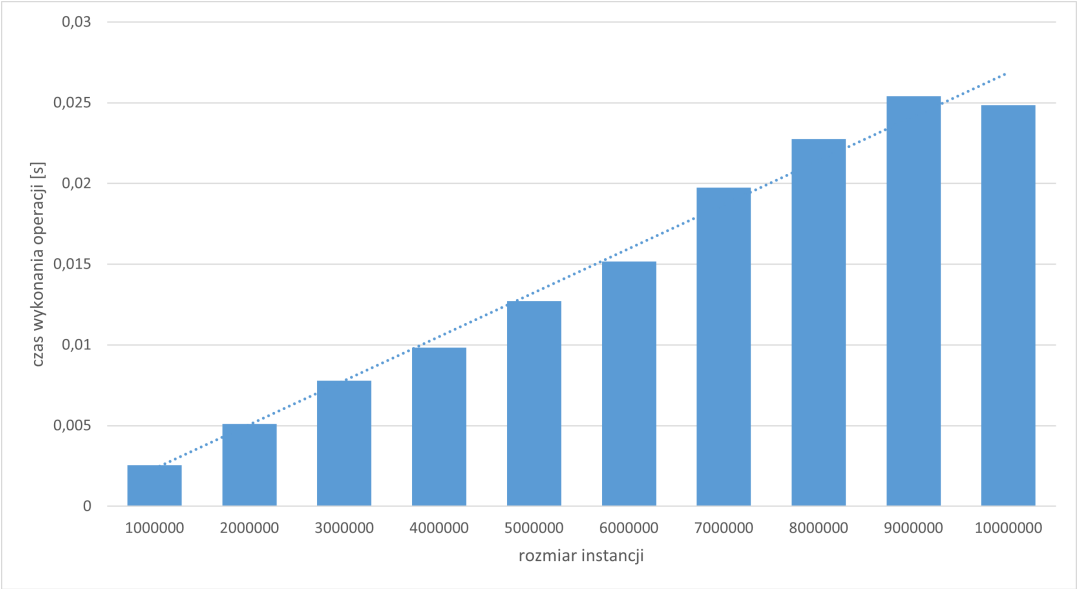



Rysunek 30: Wpływ wielkości instancji (w zakresie 1000000 - 10000000) na czas usuwania elementu z tablicy.

### Wyszukaj



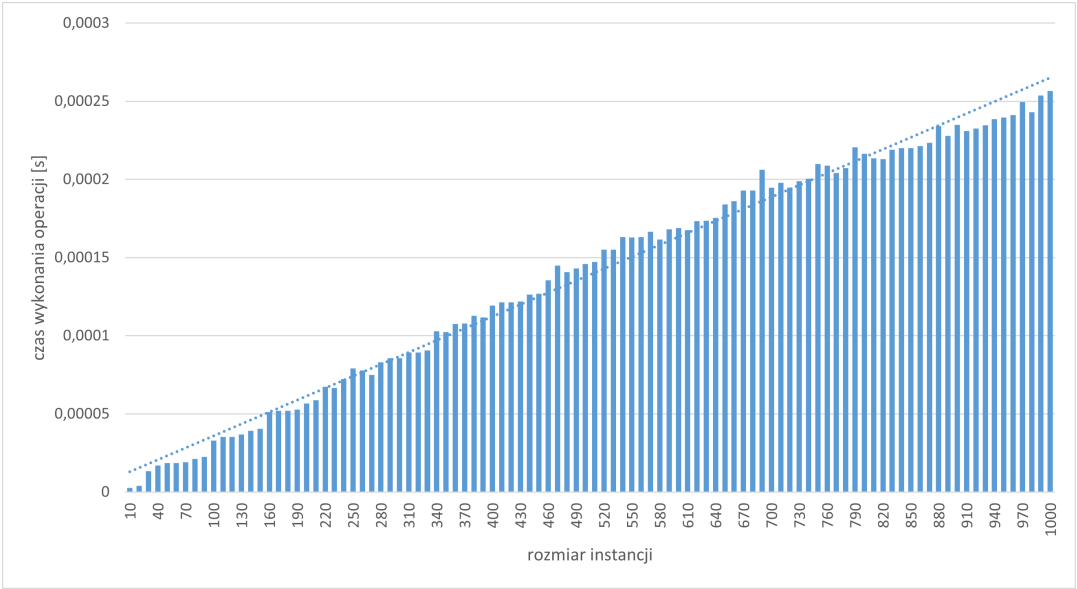

Rysunek 31: Wpływ wielkości instancji (w zakresie 10 - 1000) na czas wyszukiwania elementu w tablicy.



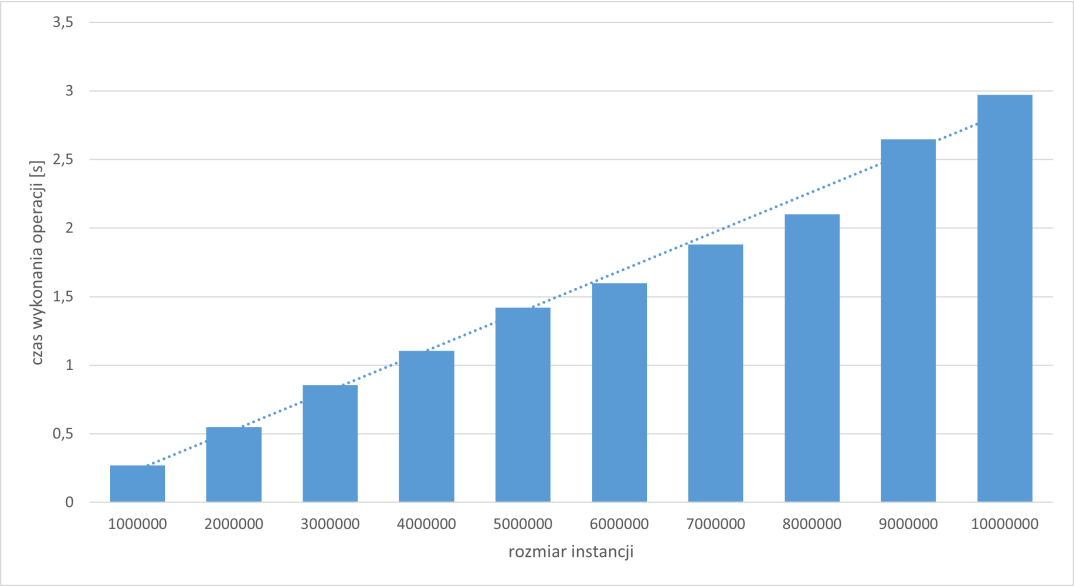

Rysunek 32: Wpływ wielkości instancji (w zakresie 1000000 - 10000000) na czas wyszukiwania elementu w tablicy.

## Lista

### Utwórz

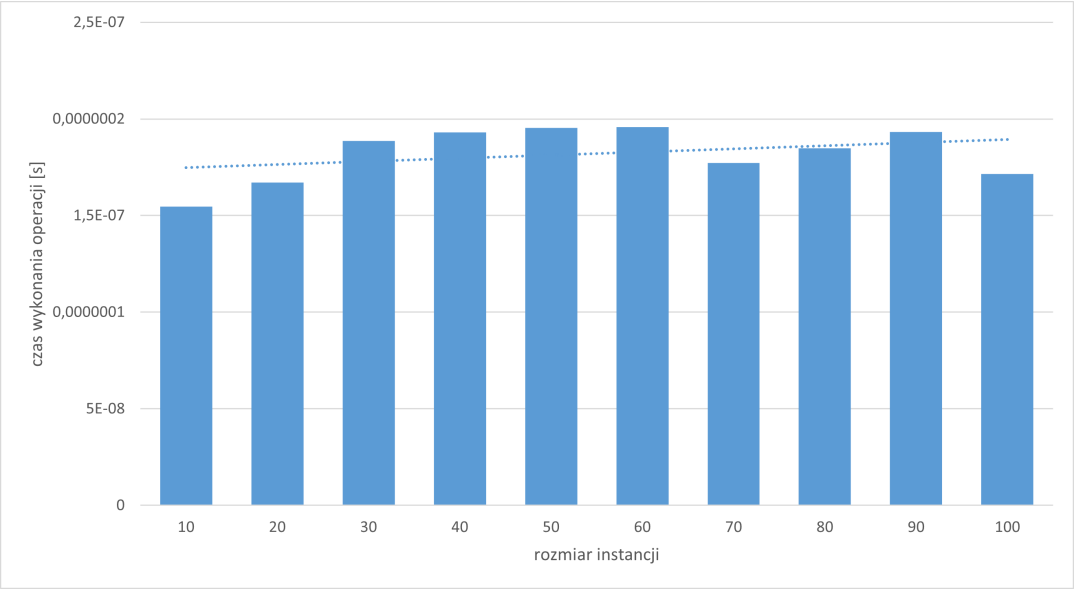



Rysunek 33: Wpływ wielkości instancji (w zakresie 10 - 1000) na czas tworzenia listy.



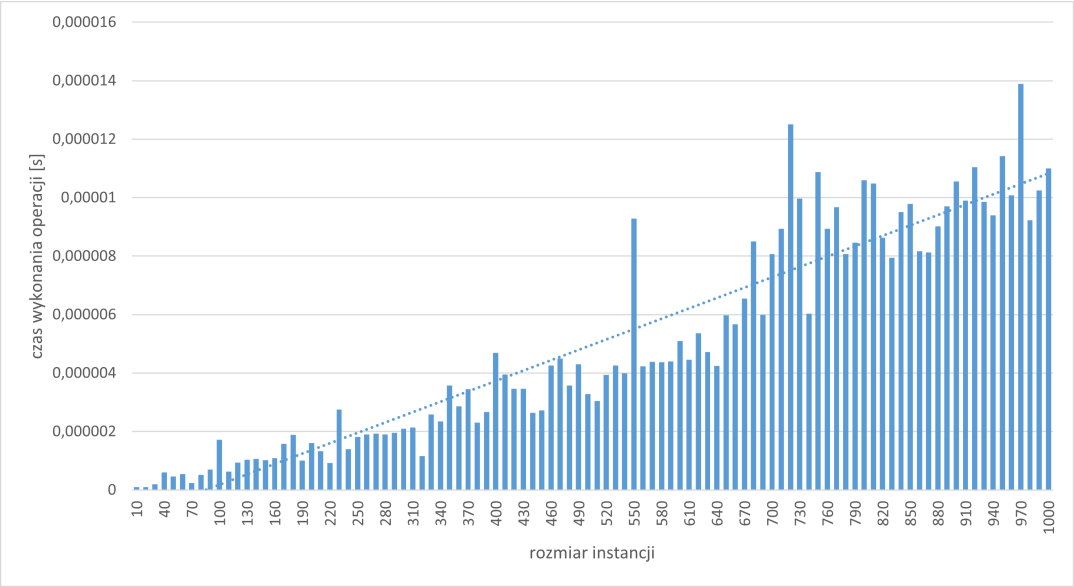

Rysunek 34: Wpływ wielkości instancji (w zakresie 1000000 - 10000000) na czas tworzenia listy.

### Dodaj



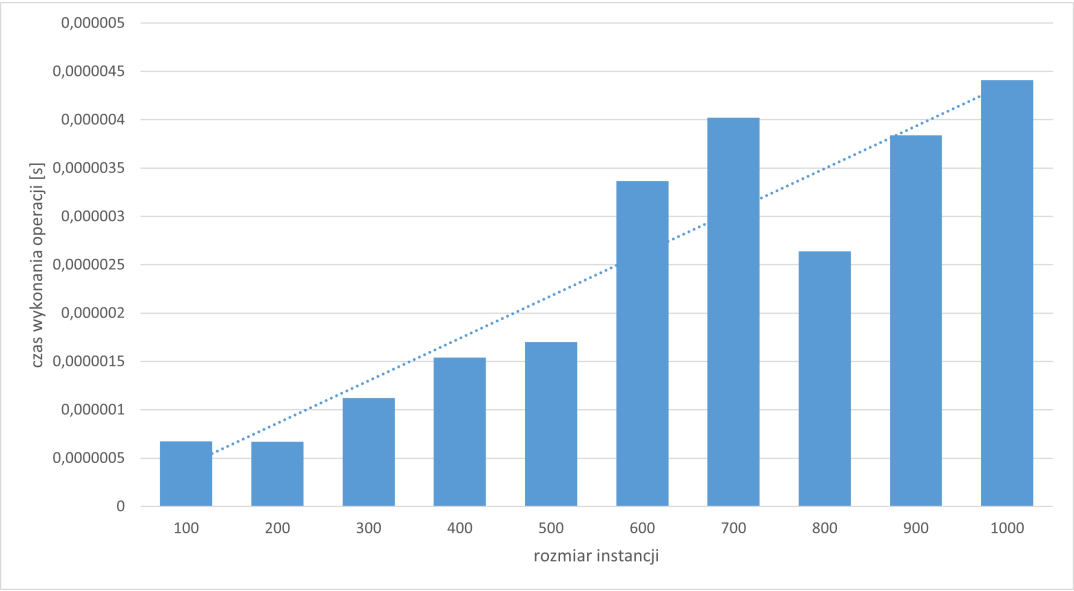

Rysunek 35: Wpływ wielkości instancji (w zakresie 10 - 100) na czas dodawania elementu do listy.

### Usuń




Rysunek 36: Wpływ wielkości instancji (w zakresie 10 - 1000) na czas usuwania elementu z listy.

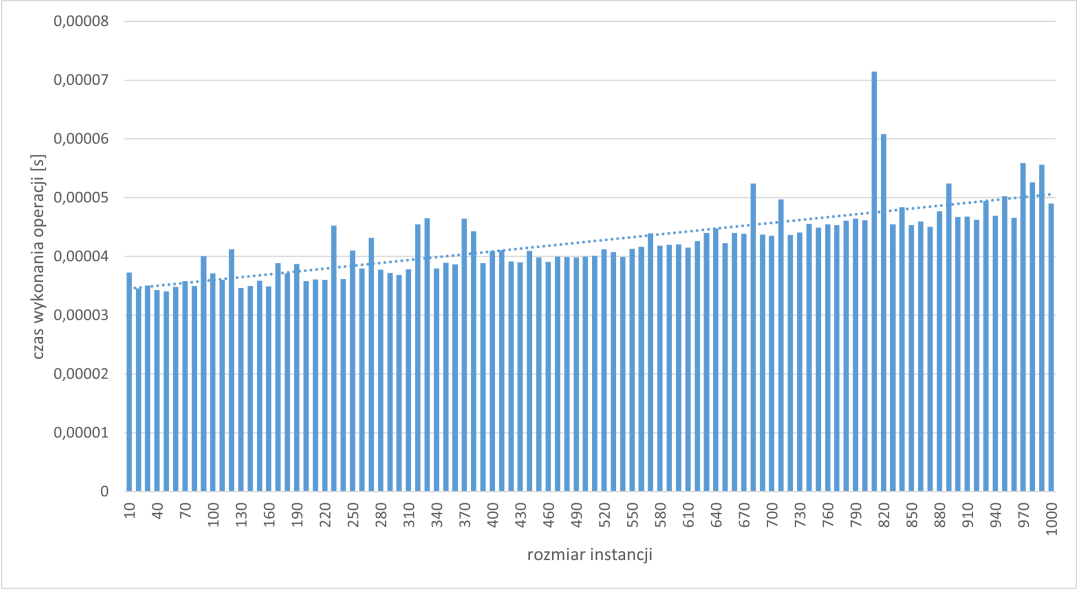
### Wyszukaj



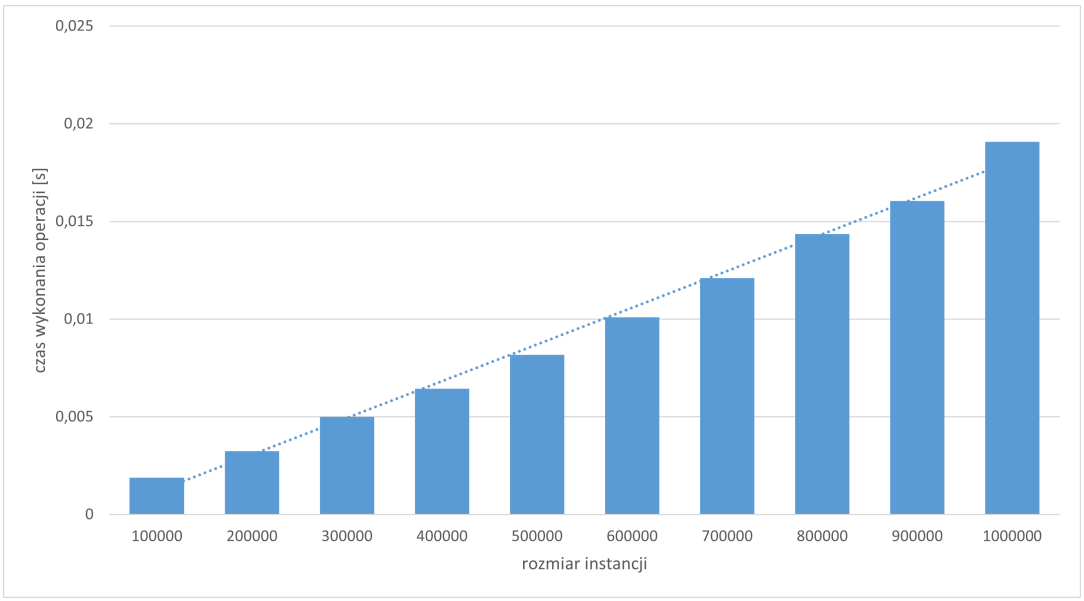

Rysunek 37: Wpływ wielkości instancji (w zakresie 100 - 1000) na czas wyszukiwania elementu w listy.

# Stos

## Utwórz

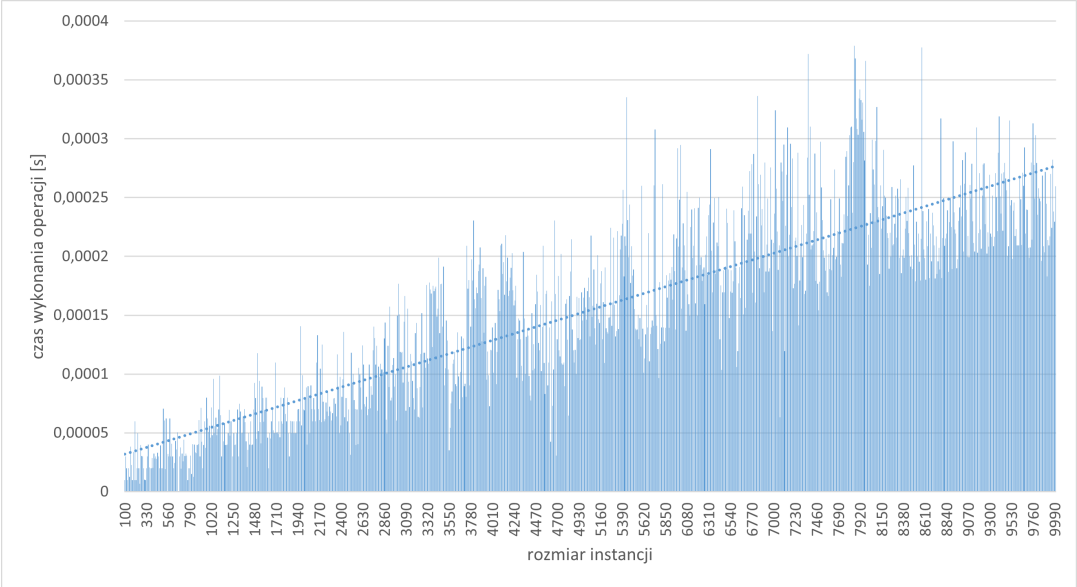



Rysunek 38: Wpływ wielkości instancji (w zakresie 10 - 1000) na czas tworzenia stosu.

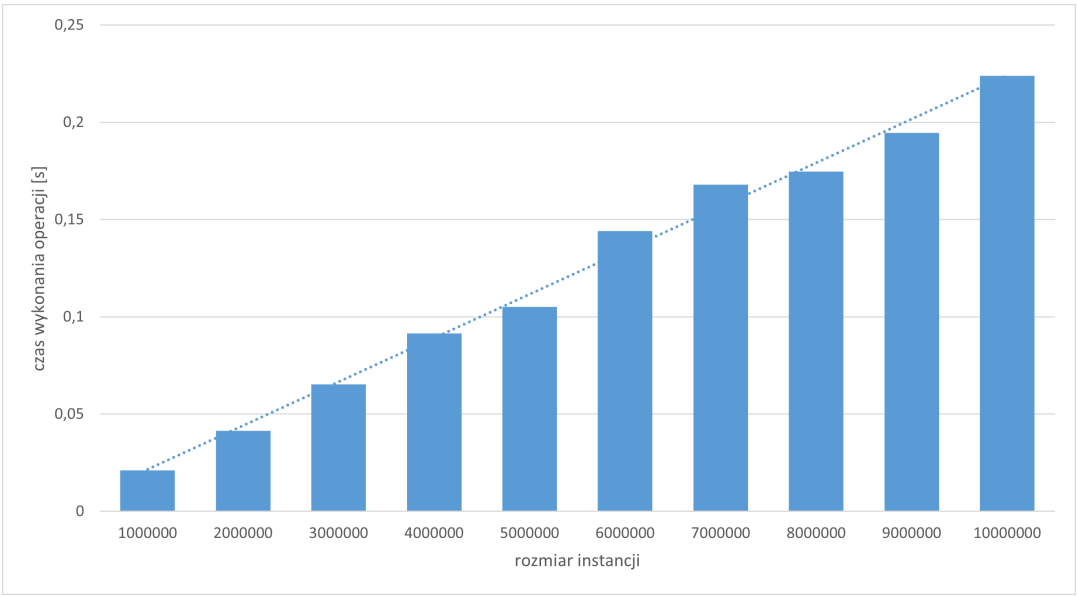



Rysunek 39: Wpływ wielkości instancji (w zakresie 100000 - 1000000) na czas tworzenia stosu.

**Wstaw**

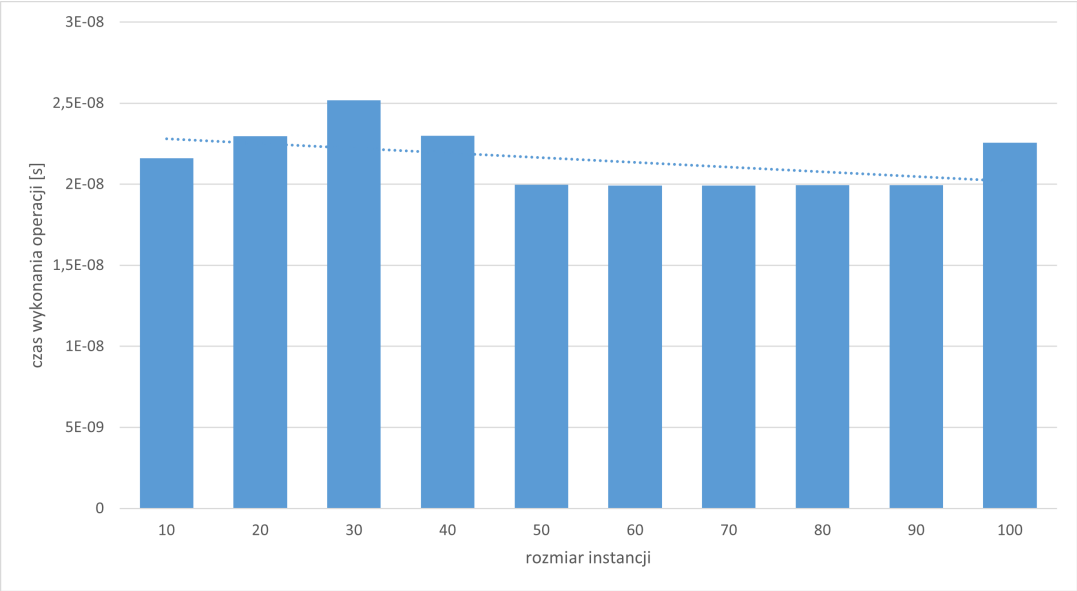



Rysunek 40: Wpływ wielkości instancji (w zakresie 100 - 10000) na czas wstawiania elementu na stos.



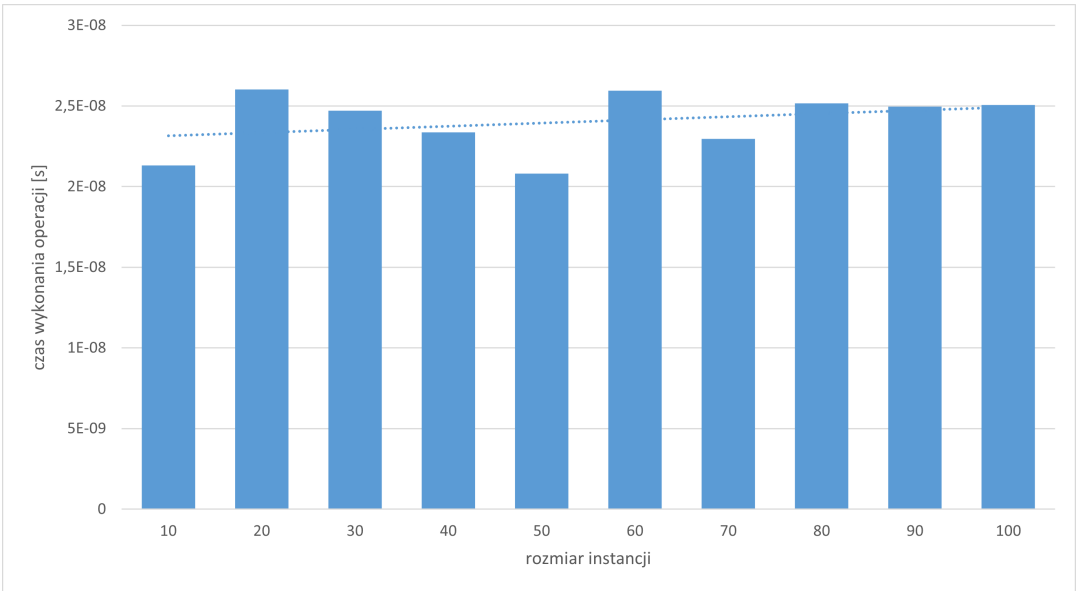

Rysunek 41: Wpływ wielkości instancji (w zakresie 1000000 - 10000000) na czas wstawiania elementu na stos.

### Dodaj (push)



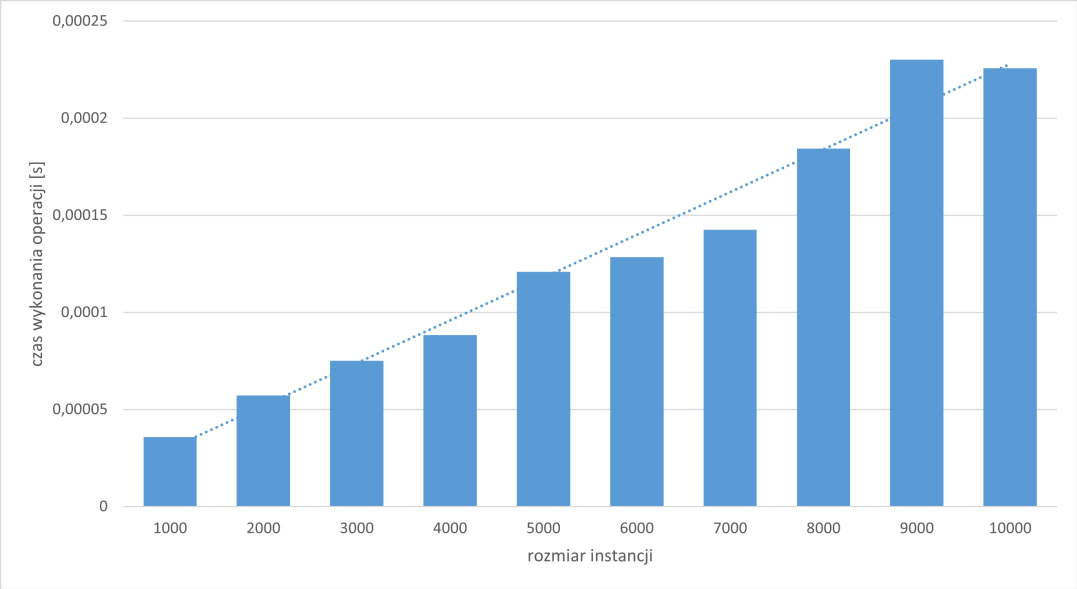

Rysunek 42: Wpływ wielkości instancji (w zakresie 10 - 100) na czas dodawania elementu do stosu (*push*).

### Usuń (pop)




Rysunek 43: Wpływ wielkości instancji (w zakresie 10 - 100) na czas usuwania elementu ze stosu (*pop*).

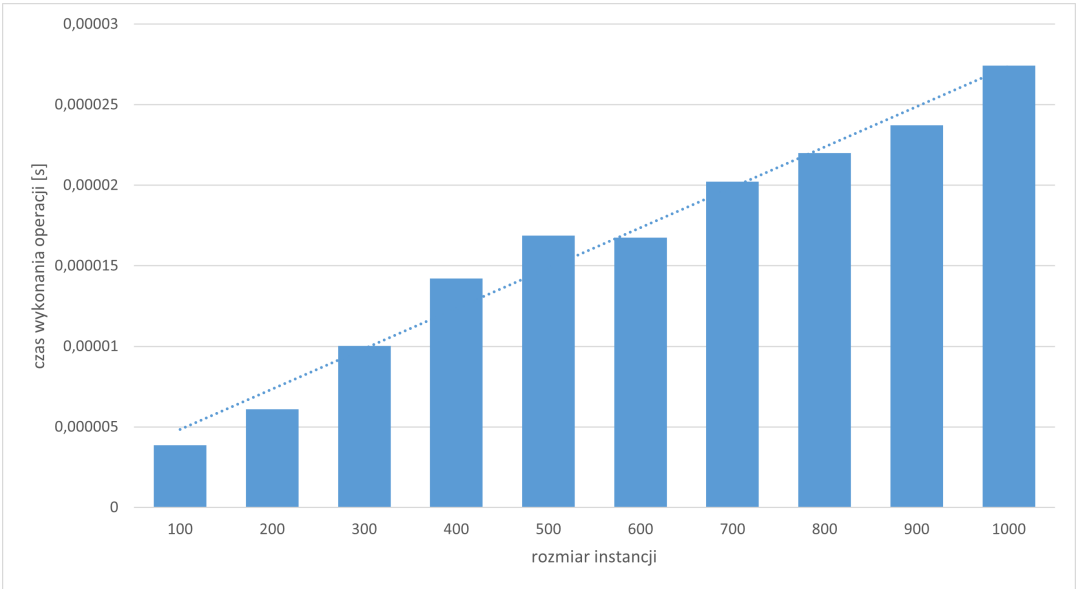
### Wyszukaj




Rysunek 44: Wpływ wielkości instancji (w zakresie 1000 - 10000) na czas wyszukiwania elementu na stosie.

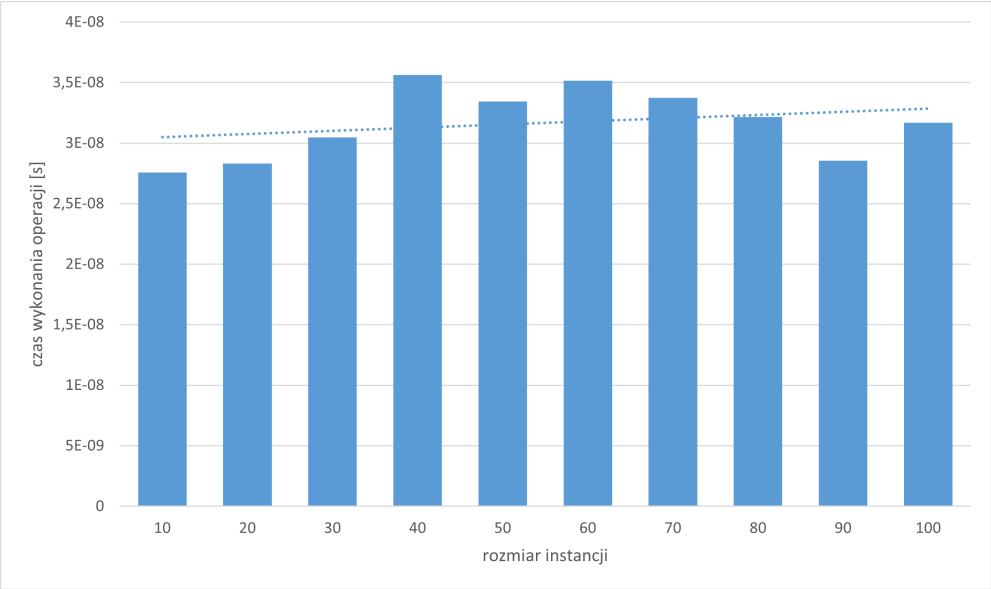
## Kolejka

### Utwórz



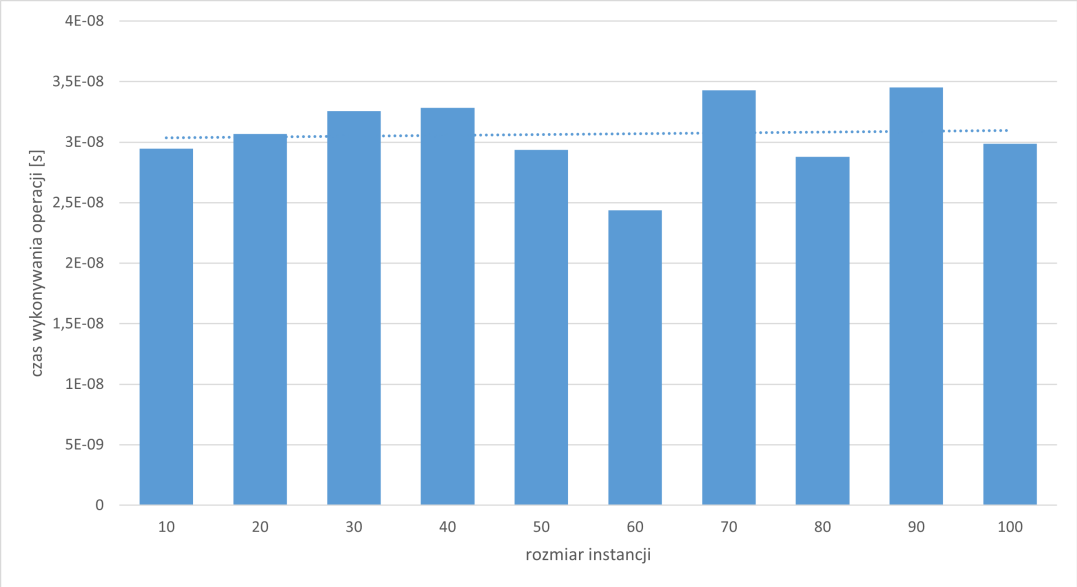

Rysunek 45: Wpływ wielkości instancji (w zakresie 100 - 1000) na czas tworzenia kolejki.

### Dodaj (enqueue)



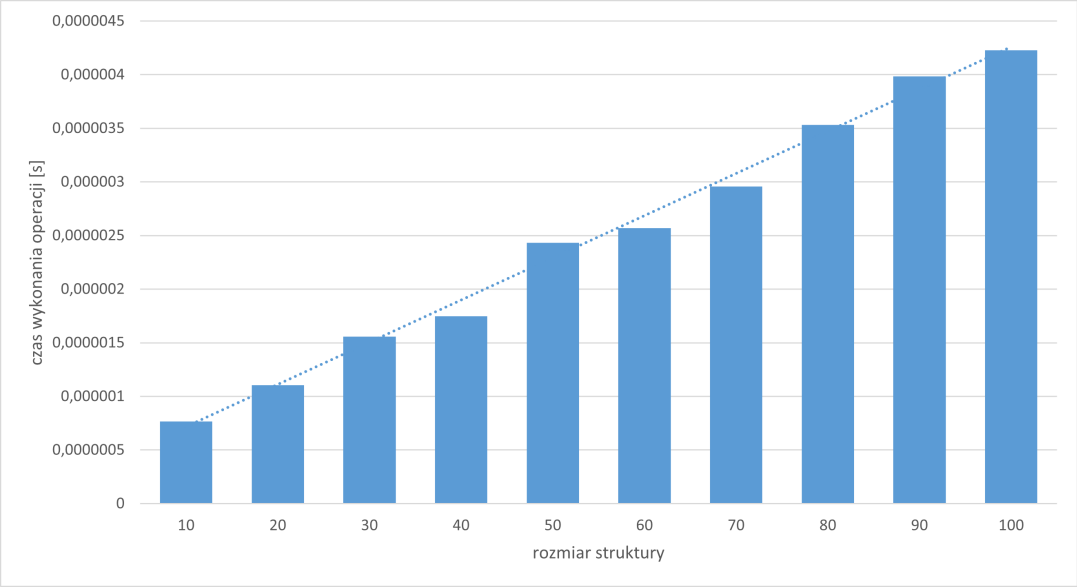

Rysunek 46: Wpływ wielkości instancji (w zakresie 10 - 100) na czas dodawania elementu do kolejki (*enqueue*).

### Usuń (dequeue)




Rysunek 47: Wpływ wielkości instancji (w zakresie 10 - 100) na czas usuwania elementu z kolejki (*dequeue*).

### Wyszukaj




Rysunek 48: Wpływ wielkości instancji (w zakresie 10 - 100) na czas wyszukiwania elementu w kolejce.

# 6 Analiza wyników i wnioski

## Tablica

### Utwórz

Prosta najlepszego dopasowania zależności wzrostu czasu wykonywania operacji tworzenia tablicy względem rozmiaru instancji ma charakter liniowy (rysunek 23 i 24), co potwierdza założenie teoretyczne, że górne ograniczenie złożoności czasowej algorytmu tworzenia tablicy jest równe $O(n)$. Wynika to z konieczności alokowania obszaru pamięci o długości $n * rozmiartypudanych$ oraz przypisania wartości elementom tablicy, co jest powiązane z rozmiarem instancji.

### Wstaw

Prosta najlepszego dopasowania zależności wzrostu czasu wykonywania operacji wstawiania elementu do tablicy względem rozmiaru instancji ma charakter stały (rysunek 25 i 26), co potwierdza założenie teoretyczne, że górne ograniczenie złożoności czasowej algorytmu wstawiania elementu do tablicy jest równe $O(1)$. Wynika to z niezależności operacji od rozmiaru instancji, ponieważ za każdym razem operacja przeprowadzana jest tylko na jednym elemencie struktury.

### Dodaj

Prosta najlepszego dopasowania zależności wzrostu czasu wykonywania operacji dodawania elementu do tablicy względem rozmiaru instancji ma charakter liniowy (rysunek 27 i 28), co potwierdza założenie teoretyczne, że górne ograniczenie złożoności czasowej algorytmu dodawania elementu do tablicy jest równe $O(n)$. Wynika to z konieczności alokowania obszaru pamięci o długości$(n+1) * rozmiar\ typu\ danych$, kopiowania pamięci oraz zwolnienia pamięci alokowanej wcześniej, co jest powiązane z rozmiarem instancji.

### Usuń

Prosta najlepszego dopasowania zależności wzrostu czasu wykonywania operacji usuwania elementu z tablicy względem rozmiaru instancji ma charakter liniowy (rysunek 29 i 30), co potwierdza założenie teoretyczne, że górne ograniczenie złożoności czasowej algorytmu usuwania elementu z tablicy jest równe $O(n)$. Wynika to z konieczności alokowania obszaru pamięci o długości$(n-1) * rozmiar\ typu\ danych$, kopiowania pamięci oraz zwolnienia pamięci alokowanej wcześniej, co jest powiązane z rozmiarem instancji.

### Wyszukaj

Prosta najlepszego dopasowania zależności wzrostu czasu wykonywania operacji wyszukiwania elementu w tablicy względem rozmiaru instancji ma charakter liniowy (rysunek 31 i 32), co potwierdza założenie teoretyczne, że górne ograniczenie złożoności czasowej algorytmu wyszukiwania elementu w tablicy jest równe $O(n)$. Wynika to z konieczność porównania (w najgorszym przypadku) wszystkich elementów tablicy z elementem szukanym, co jest powiązane z rozmiarem instancji.

## Lista

### Utwórz

Prosta najlepszego dopasowania zależności wzrostu czasu wykonywania operacji tworzenia listy względem rozmiaru instancji ma charakter liniowy (rysunek 33 i 34), co potwierdza założenie teoretyczne, że górne ograniczenie złożoności czasowej algorytmu tworzenia listy jest równe $O(n)$. Wynika to z konieczności utworzenia $n$ węzłów, co jest powiązane z rozmiarem instancji.

### Dodaj

Prosta najlepszego dopasowania zależności wzrostu czasu wykonywania operacji dodawania elementu do listy względem rozmiaru instancji ma charakter stały (rysunek 35), co potwierdza założenie teoretyczne, że górne ograniczenie złożoności czasowej algorytmu dodawania elementu do listy jest równe $O(1)$. Wynika to z konieczności utworzenia tylko jednego węzła, co nie jest powiązane z rozmiarem instancji.

### Usuń

Prosta najlepszego dopasowania zależności wzrostu czasu wykonywania operacji usuwania elementu z listy względem rozmiaru instancji ma charakter liniowy (rysunek 36), co potwierdza założenie teoretyczne, że górne ograniczenie złożoności czasowej algorytmu usuwania elementu z listy jest równe $O(n)$. Wynika to z konieczności znalezienia w liście elementu do usunięcia (w najgorszym przypadku przeszukanie całej listy), co jest powiązane z rozmiarem instancji. złożoność operacji usunięcia wybranego węzła nie jest powiązana z rozmiarem instancji.

### Wyszukaj

Prosta najlepszego dopasowania zależności wzrostu czasu wykonywania operacji wyszukiwania elementu w liście względem rozmiaru instancji ma charakter liniowy (rysunek 37), co potwierdza założenie teoretyczne, że górne ograniczenie złożoności czasowej algorytmu wyszukiwania elementu w liście jest równe $O(n)$. Wynika to z konieczność porównania (w najgorszym przypadku) wszystkich elementów listy z elementem szukanym, co jest powiązane z rozmiarem instancji.

## Stos

### Utwórz

Prosta najlepszego dopasowania zależności wzrostu czasu wykonywania operacji tworzenia stosu względem rozmiaru instancji ma charakter liniowy (rysunek 38 i 39), co potwierdza założenie teoretyczne, że górne ograniczenie złożoności czasowej algorytmu tworzenia stosu jest równe $O(n)$. Wynika to z konieczności umieszczenia $n$ elementów na stosie, co jest powiązane z jego rozmiarem.

### Wstaw

Prosta najlepszego dopasowania zależności wzrostu czasu wykonywania operacji wstawiania elementu do stosu względem rozmiaru instancji ma charakter liniowy (rysunek 40 i 41), co potwierdza założenie teoretyczne, że górne ograniczenie złożoności czasowej algorytmu wstawiania elementu do stosu jest równe $O(n)$. Wynika to z konieczności uzyskania dostępu do wybranego elementu na stosie, co jest powiązane rozmiarem instancji. Wymusza to konieczność przełożenia części elementów na stos pomocniczy i, po dodaniu elementu, odłożeniu ich z powrotem.

### Dodaj (push)

Prosta najlepszego dopasowania zależności wzrostu czasu wykonywania operacji dodawania elementu do stosu względem rozmiaru instancji ma charakter stały (rysunek 42), co potwierdza założenie teoretyczne, że górne ograniczenie złożoności czasowej algorytmu dodawania elementu do stosu jest równe $O(1)$. Wynika to z niezależności operacji od wysokości stosu (rozmiaru instancji).

### Usuń (pop)

Prosta najlepszego dopasowania zależności wzrostu czasu wykonywania operacji usuwania elementu ze stosu względem rozmiaru instancji ma charakter stały (rysunek 43), co potwierdza założenie teoretyczne, że górne ograniczenie złożoności czasowej algorytmu usuwania elementu ze stosu jest równe $O(1)$. Wynika to z niezależności operacji od wysokości stosu (rozmiaru instancji).

### Wyszukaj

Prosta najlepszego dopasowania zależności wzrostu czasu wykonywania operacji wyszukiwania elementu na stosie względem rozmiaru instancji ma charakter liniowy (rysunek 44), co potwierdza założenie teoretyczne, że górne ograniczenie złożoności czasowej algorytmu wyszukiwania elementu na stosie jest równe $O(n)$. Wynika to z konieczności (w najgorszym przypadku) porównania wszystkich elementów stosu z wartością szukaną. Struktura stosu wymusza również odkładanie wartości przeszukiwanych na stos tymczasowy, a następnie odłożenie ich z powrotem.

## Kolejka

### Utwórz

Prosta najlepszego dopasowania zależności wzrostu czasu wykonywania operacji tworzenia kolejki względem rozmiaru instancji ma charakter liniowy (rysunek 45), co potwierdza założenie teoretyczne, że górne ograniczenie złożoności czasowej algorytmu tworzenia kolejki jest równe $O(n)$. Wynika to z konieczności alokowania obszaru pamięci o długości $n*rozmiar\ typu\ danych$ oraz przypisania wartości elementom kolejki, co jest powiązane z rozmiarem instancji.

### Dodaj (enqueue)

Prosta najlepszego dopasowania zależności wzrostu czasu wykonywania operacji dodawania elementu do kolejki względem rozmiaru instancji ma charakter stały (rysunek 46), co potwierdza założenie teoretyczne, że górne ograniczenie złożoności czasowej algorytmu dodawania elementu do kolejki jest równe $O(1)$. Wynika to z niezależności operacji od rozmiaru instancji.

### Usuń (dequeue)

Prosta najlepszego dopasowania zależności wzrostu czasu wykonywania operacji usuwania elementu z kolejki względem rozmiaru instancji ma charakter stały (rysunek 47), co potwierdza założenie teoretyczne, że górne ograniczenie złożoności czasowej algorytmu usuwania elementu z kolejki jest równe $O(1)$. Wynika to z niezależności operacji od rozmiaru instancji.

### Wyszukaj

Prosta najlepszego dopasowania zależności wzrostu czasu wykonywania operacji wyszukiwania elementu w kolejce względem rozmiaru instancji ma charakter liniowy (rysunek 48), co potwierdza założenie teoretyczne, że górne ograniczenie złożoności czasowej algorytmu wyszukiwania elementu w kolejce jest równe $O(n)$. Wynika to z konieczności (w najgorszym przypadku) porównania wszystkich elementów kolejki z wartością szukaną. Struktura kolejki wymusza również odkładanie wartości przeszukiwanych do kolejki tymczasowej, a następnie odłożenie ich z powrotem.

## Wnioski

Powyższe zadanie umożliwiło dogłębne zaznajomienie się z podstawowymi strukturami danych oraz algorytmami operacji wykonywanych na nich. Pozornie i z założenia (być może nieuzasadnionego) proste zagadnienia przy implementacji wymagały wnikliwego zapoznania się z materiałami źródłowymi oraz możliwościami programowymi języków programowania co było niewątpliwie pouczające.
Mimo nie spełnienia jednego z zakładanych celów, jakim było wykonanie badań wszystkich struktur danych na instancjach sięgających rozmiarem 10 milionów elementów, udało się spełnić kluczowy cel jakim było wyznaczenie czasowej złożoności obliczeniowej badanych algorytmów. Badania na instancjach sięgających rozmiarem 10 milionów elementów udało się przeprowadzić w połowie przypadków.
W początkowej fazie realizacji zadania program potrzebny do wykonania pomiarów zaimplementowałem w języku Python. W czasie testów okazał się on jednak zbyt wolny do przeprowadzenia rzetelnych pomiarów, co zmusiło mnie do napisania całego programu od nowa w języku C++. Była to pouczająca lekcje, szczególnie w kontekście badanego tematu.