Złożone struktury danych

Oliwia Koteluk 145 185
Wojciech Marciniak 145 166

Binarne drzewo poszukiwań (ang. *binary search tree,* BST)

Jest to dynamiczna struktura danych oparta na drzewie binarnym. Składa się z węzłów, z których każdy posiada swoją wartość oraz wskaźniki na lewe i prawe dziecko. Dodatkowo drzewo BST posiada następujące właściwości:

- każdy z jego węzłów może mieć maksymalnie dwóch synów (tzn. stopień wyjściowy każdego wierzchołka musi być mniejszy bądź równy 2),
- wartość zapisana w lewym synie wierzchołka musi być mniejsza niż w węźle-rodzicu,
- wartość zapisana w prawym synie wierzchołka musi być większe niż w węźle-rodzicu.

W celu wykonania zadania zaimplementowano funkcję obsługujące drzewo binarne (budowanie drzewa, wyszukiwanie wartości o wskazanym kluczu czy usuwanie elementów z drzewa). Powyższe operacje zostały wykonane dla różnej wielkości danych wejściowych (13 różnych wielkości). Dane były dodawane z tablicy do drzewa na trzy sposoby: w sposób rosnący, losowo oraz zgodnie z zasadą połowienia binarnego.

Zasada połowienia binarnego oparta jest metodzie dziel i zwyciężaj. Podczas budowania struktury drzewa można ją zastosować jedynie dla tablicy uporządkowanej. Polega na wyszukaniu elementu o indeksie będącego środkiem tablicy i dodaniu go do drzewa. Następnie tablica jest dzielona na coraz mniejsze przedziały przez rekurencyjne wywoływanie tej samej funkcji. Czynność ta jest powtarzana dopóki wszystkie elementy nie zostaną dodane do drzewa. Złożoność czasowa wyszukiwania elementów w tablicy z zastosowaniem zasady połowienia binarnego wynosi O(log2 n). Dodawanie do drzewa elementów zgodnie z tą zasadą gwarantuje utworzenie drzewa zrównoważonego, czyli takie, którego wysokość lewego i prawego poddrzewa każdego wierzchołka różni się co najwyżej o 1.

W celu oceny efektywności czasowej wykonywania powyższych operacji zmierzono czas ich wykonywania. Wyniki przedstawiono poniżej w tabelach oraz na wykresach.

1. <u>Dodawanie elementów do drzewa</u>

Algorytm dodawania elementów do drzewa pobiera wartość klucza danego węzła z tablicy, następnie w przypadku braku korzenia (drzewo wtedy nie istnieje) tworzony jest pierwszy węzeł, a wartość pod zerowym indeksem tej tablicy staje się korzeniem drzewa. Jeśli drzewo istnieje, algorytm sprawdza czy klucz dodawanego elementu jest mniejszy czy większy od wartości przechowywanej w węźle, który już jest elementem drzewa. Jeśli jest on mniejszy to kierujemy się w stronę lewego poddrzewa, w przeciwnym wypadku - do prawego poddrzewa. Operację powtarzana jest aż do momentu, gdy dodawany węzeł znajdzie swoje miejsce w strukturze drzewa BST.

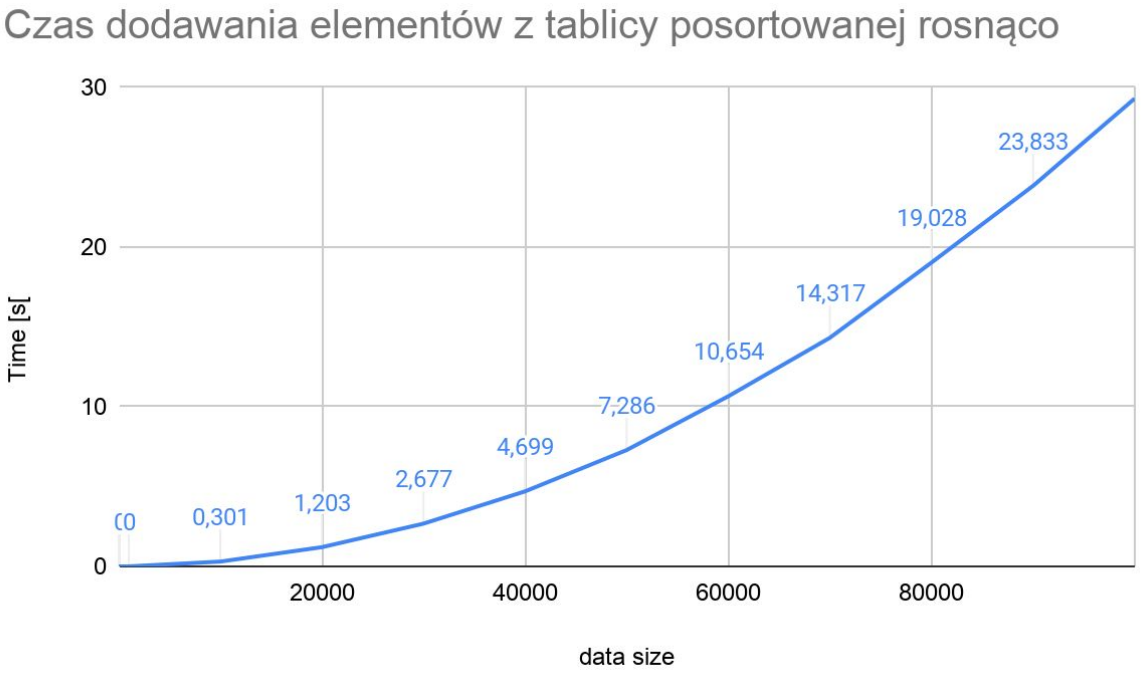
Algorytm działa najwolniej w przypadku dodawania danych posortowanych rosnąco. W tym przypadku każdy węzeł utworzonego drzewa BST będzie posiadał tylko prawe poddrzewa i przez to drzewo będzie najwyższe. Dla pozostałych dwóch rozkładów danych

wejściowych algorytm działa bardzo efektywnie. Jednak trochę lepiej działał dla danych dodawanych zgodnie z zasadą połowienia binarnego, co potwierdza, że ten sposób jest najbardziej optymalny.

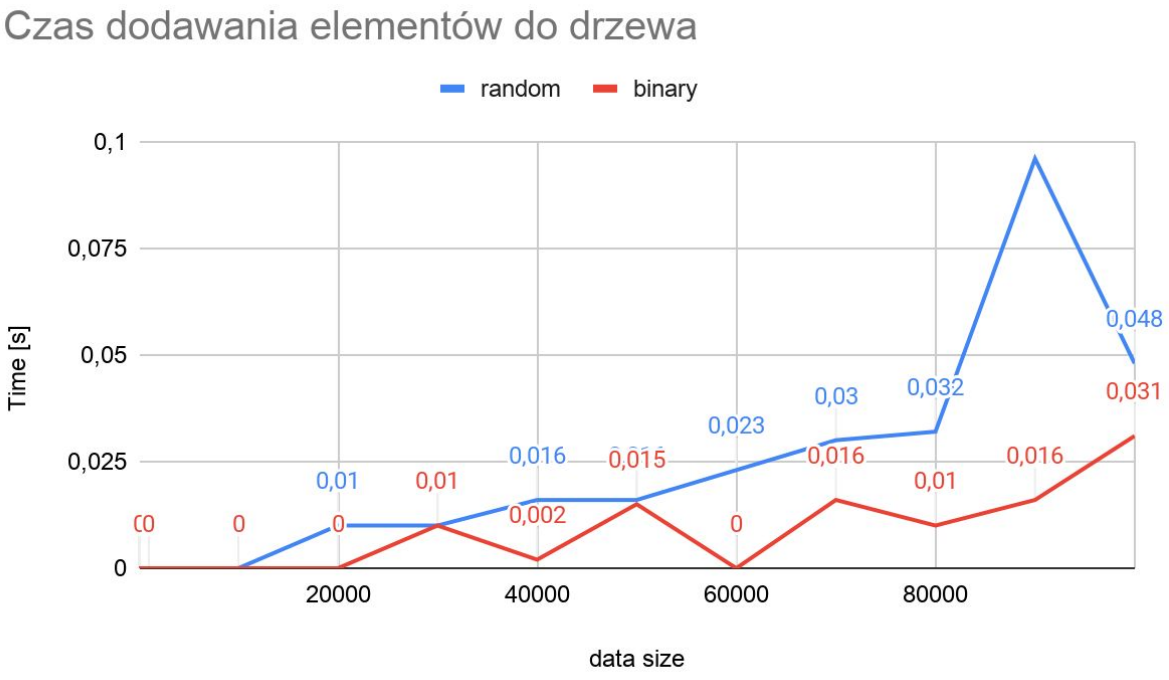
Złożoność czasowa algorytmu dodawania elementu do drzewa BST wynosi O(log(n)) w wariancie optymalnym lub O(n) w wariancie pesymistycznym.

Tabela 1. Czas działania algorytmu dodawania do drzewa elementów (w sekundach) dla różnego rozkładu oraz różnej wielkości danych wejściowych.

Data size	Increasing	Random	Binary
10	0,000	0,000	0,000
100	0,000	0,000	0,000
1000	0,000	0,000	0,000
10000	0,301	0,000	0,000
20000	1,203	0,010	0,000
30000	2,677	0,010	0,010
40000	4,699	0,016	0,002
50000	7,286	0,016	0,015
60000	10,654	0,023	0,000
70000	14,317	0,030	0,016
80000	19,028	0,032	0,010
90000	23,833	0,096	0,016
100000	29,301	0,048	0,031



Wykres 1. Algorytm dodawania elementów do drzewa elementów z tablicy posortowanej rosnąco - wykres zależności rozmiaru danych od szybkości [sekundy] działania algorytmu.



Wykres 2. Algorytm dodawania elementów do drzewa- wykres zależności rozmiaru danych/rozkładu danych od szybkości [sekundy] działania algorytmu.

2. Szukanie elementu

Następnie zmierzono czas wyszukiwania każdego elementu dodanego wcześniej do drzewa. Elementy były wyszukiwane w losowej kolejności, a oznaczenie rodzaju danych (increasing, random, binary) w tym przypadku oznacza rozkład danych w tablicy, z której dane były dodawane do drzewa.

Mechanizm działania tej czynności polega na porównywaniu klucza szukanego węzła z kluczami węzłów w drzewie. Jeśli klucze są sobie równe zwracane są dane węzła. Natomiast jeśli klucz jest mniejszy przeszukiwane jest lewe poddrzewo węzła, w przeciwnym wypadku - prawe poddrzewo. Czynność powtarzana jest dopóki program nie przejdzie przez całe drzewo, wtedy zwracany jest węzeł o szukanej wartości lub w przypadku jego braku program zwraca zero.

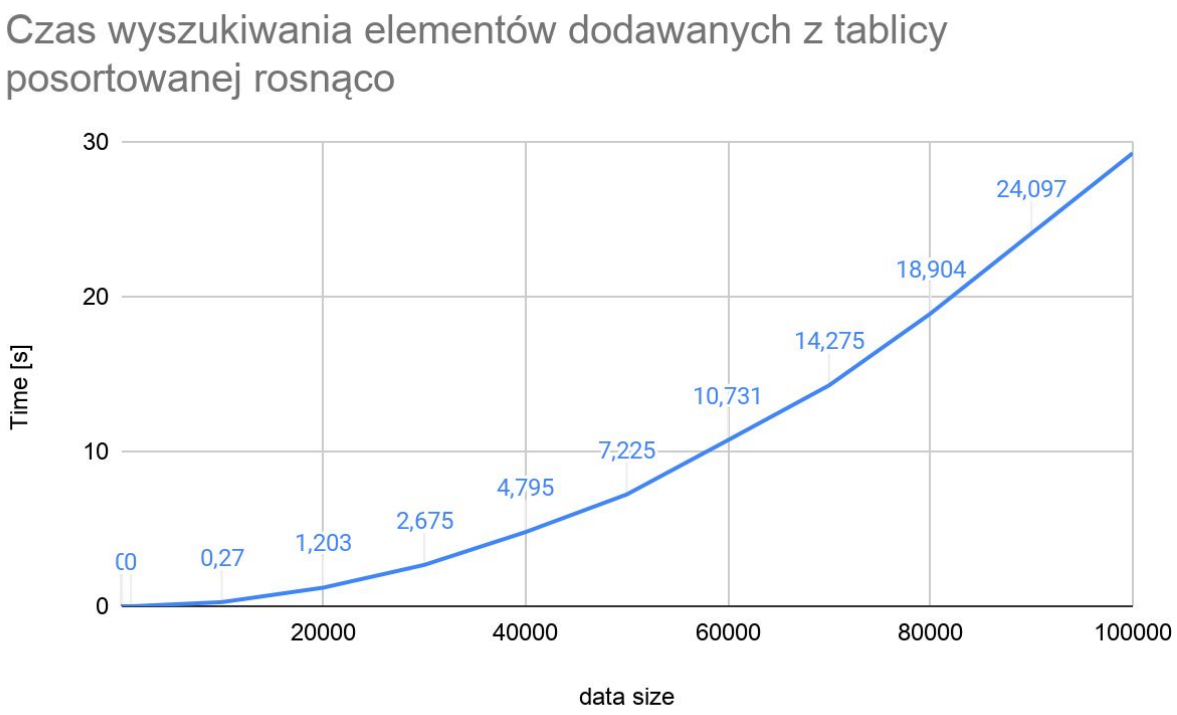
Wydajność tego algorytmu jest powiązana z rozkładem danych w tablicy, z której elementy do drzewa były dodawane. Podobnie jak w przypadku dodawania do drzewa, algorytm działa najwolniej w przypadku danych posortowanych rosnąco. Dla danych dodawanych losowo oraz zgodnie z zasadą połowienia binarnego algorytm działa z podobno wydajnością, jednak trochę lepiej wydaje się działać w przypadku wyszukiwania danych dodawanych zgodnie z zasadą połowienia binarnego.

Operacja wyszukiwania elementu w drzewie BST zależy od ilości poziomów w drzewie i posiada złożoność obliczeniową O(k), gdzie przez k oznacza się ilość poziomów. Stąd w przypadku drzewa tworzonego z losowej tabeli czas działania algorytmu jest najdłuższy. Takie drzewo posiada najwięcej poziomów, ponieważ każdy węzeł posiada tylko prawe poddrzewa. W drzewie zrównoważonym jest to klasa O(log n), natomiast w drzewie losowanym O($\sqrt{n}$), gdzie n oznacza liczbę węzłów.

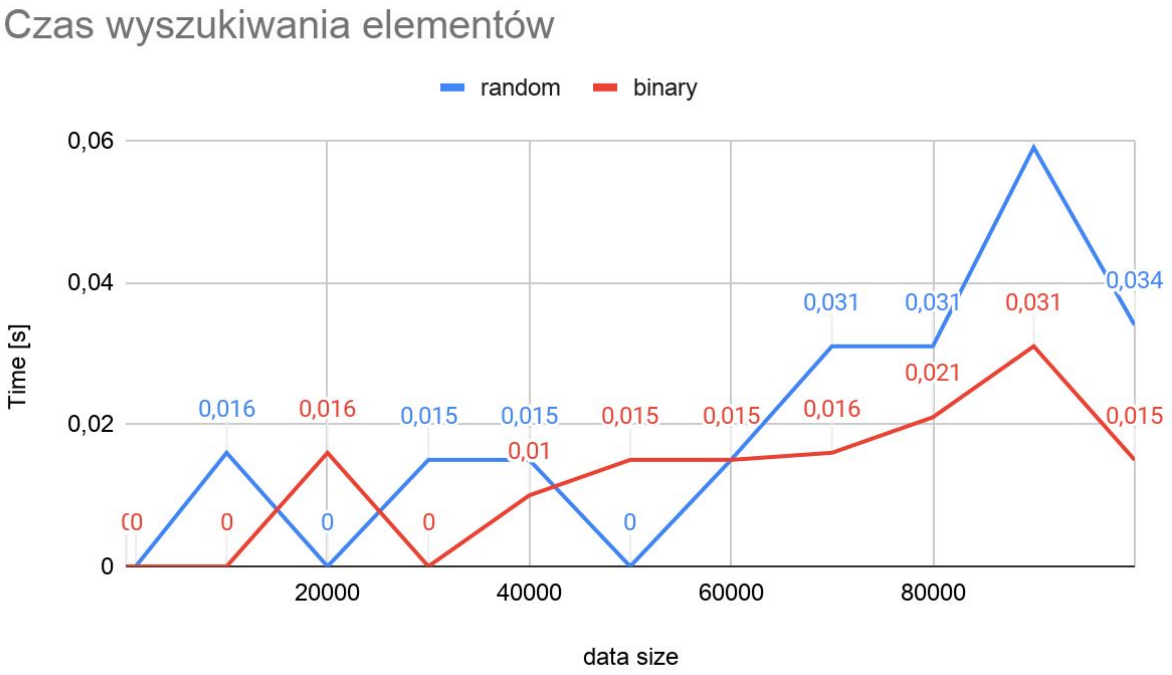
Tabela 2. Czas działania algorytmu wyszukiwania każdego elementu w losowej kolejności (w sekundach) dla różnej wielkości danych wejściowych w zależności od rodzaju zastosowanego rozkładu danych przy dodawaniu tych elementów.

Data size	Increasing	Random	Binary
10	0,000	0,000	0,000
100	0,000	0,000	0,000
1000	0,000	0,000	0,000
10000	0,270	0,016	0,000
20000	1,203	0,000	0,016
30000	2,675	0,015	0,000
40000	4,795	0,015	0,010
50000	7,225	0,000	0,015
60000	10,731	0,015	0,015
70000	14,275	0,031	0,016

80000	18,904	0,031	0,021
90000	24,097	0,059	0,031
100000	29,276	0,034	0,015



Wykres 3. Algorytm wyszukiwania elementów w drzewie dodanych wcześniej z tablicy posortowanej rosnąco- wykres zależności rozmiaru danych od szybkości [sekundy] działania algorytmu.



Wykres 4. Algorytm wyszukiwania elementów w drzewie- wykres zależności rozmiaru danych/rozkładu danych od szybkości [sekundy] działania algorytmu.

Warto również zauważyć, że czasy działania algorytmów dodawania oraz wyszukiwania elementów są bardzo do siebie zbliżone dla tego samego rozkładu danych.

Lista jednokierunkowa (ang. *linked list*)

Jest dynamiczną strukturą danych, w której elementy ułożone są liniowo jeden za drugim (podobnie do wagonów w pociągu). Każdy element przechowuje swoją wartość oraz wskaźnik na następny element. Cechą listy jednokierunkowej jest możliwość przechodzenia z elementu na element tylko w jednym kierunku.

W celu wykonania zadania zaimplementowano funkcję obsługujące listę (tworzenie listy, dodawanie elementów do listy, wyszukiwanie wartości o wskazanym kluczu czy usuwanie elementów z listy). Powyższe operacje zostały wykonane dla różnej wielkości danych wejściowych (13 różnych wielkości). Dane były dodawane do drzewa na dwa sposoby: w sposób rosnący oraz losowo.

W celu oceny efektywności czasowej wykonywania powyższych operacji zmierzono czas wykonywania funkcji dodawania elementów do listy oraz wyszukiwania elementów na liście). Wyniki przedstawiono poniżej w tabelach oraz na wykresach.

1. Dodawanie elementów do listy

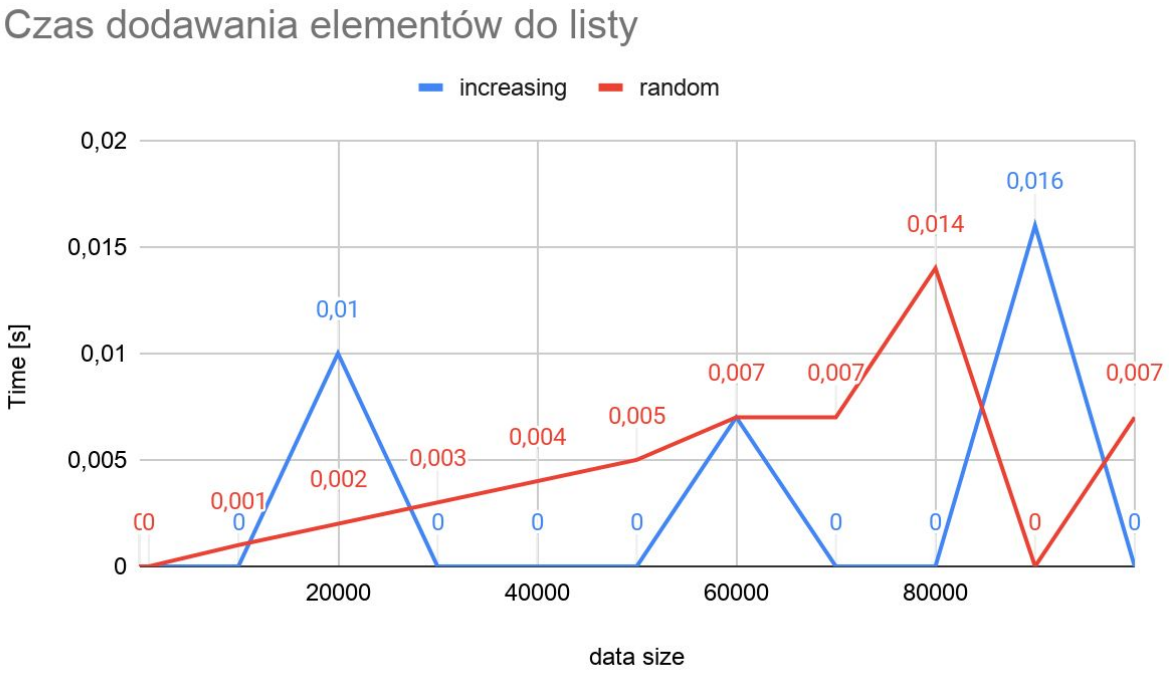
Struktura listy jednoelementowej nie narzuca relacji między elementami wchodzącymi w skład listy (w przeciwieństwie do drzewa BST), dlatego algorytm dodający element do listy pobiera wartość klucza dodawanego węzła i umieszcza ten element na końcu listy oraz ustawia wskaźnik poprzedniego węzła na ten, który chcemy dodać. Operację tą powtarza się dopóki wszystkie elementy nie zostaną dodane.

Algorytm ten działa bardzo szybko dla obu typów rozkładu danych, jednak trochę efektywniej dodaje elementy w porządku rosnącym. Na podstawie zmierzonych czasów działania algorytmów można stwierdzić, że algorytm dodawania elementu na koniec listy posiada praktycznie stałą złożoność czasową O(1).

Algorytm dodawania elementów na koniec listy posiada stałą złożoność obliczeniową O(1).

Tabela 3. Czas działania algorytmu dodawania elementów do listy (w sekundach) dla różnego rozkładu oraz różnej wielkości danych wejściowych.

Data size	Increasing	Random
10	0,000	0,000
100	0,000	0,000
1000	0,000	0,000
10000	0,000	0,001
20000	0,001	0,002
30000	0,000	0,003
40000	0,000	0,004
50000	0,000	0,005
60000	0,007	0,007
70000	0,000	0,007
80000	0,000	0,014
90000	0,016	0,000
100000	0,000	0,007



Wykres 5. Algorytm dodawania elementów do listy- wykres zależności rozmiaru danych/rozkładu danych od szybkości [sekundy] działania algorytmu.

2. Wyszukiwanie elementów na liście

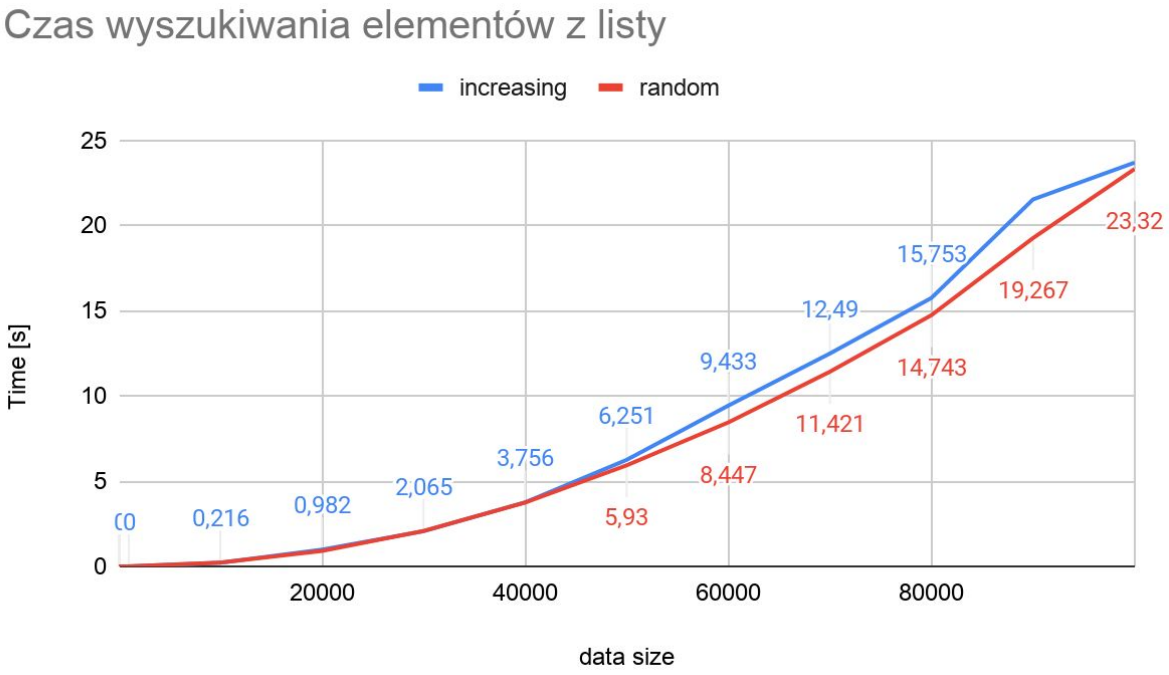
Algorytm wyszukiwania elementu na liście polega na porównywaniu poszukiwanego klucza oraz klucza elementu listy. W przypadku, gdy elementy te są sobie równe zwracany jest ten element, w przeciwnym wypadku algorytm porównuje klucz następnego elementu. W ten sposób program ,,przechodzi" przez całą listę. Jednak w przypadku listy jednokierunkowej przemieszczać się po niej można tylko w jednym kierunku (nie dozwolone jest cofanie się), więc aby wyszukać każdy kolejny element poszukiwanie należy rozpocząć od początku listy.

W tym przypadku można zaobserwować zależność wielkości danych wejściowych i szybkości działania algorytmu. Im większe dane, tym więcej czasu jest potrzebne do jego wykonania. Taką zależność można zaobserwować dla danych, które do tablicy były dodawane rosnąco i dla danych dodawanych w sposób losowy. Dla obu typów danych algorytm działa z podobną efektywnością, jednak z niewielką przewagą działa lepiej dla danych dodawanych losowo.

Złożoność obliczeniowa algorytmu wyszukiwania elementów z listy jednokierunkowej wynosi O(n).

Tabela 4. Czas działania algorytmu wyszukiwania każdego elementu w losowej kolejności (w sekundach) dla różnej wielkości danych wejściowych w zależności od rodzaju zastosowanego rozkładu danych przy dodawaniu tych elementów

Data size	Increasing	Random
10	0,000	0,000
100	0,000	0,000
1000	0,000	0,001
10000	0,216	0,222
20000	0,982	0,905
30000	2,065	2,076
40000	3,756	3,746
50000	6,251	5,930
60000	9,433	8,447
70000	12,490	11,421
80000	15,753	14,743
90000	21,534	19,267
100000	23,695	23,320



Wykres 6. Algorytm wyszukiwania elementów z listy- wykres zależności rozmiaru danych/rozkładu danych od szybkości [sekundy] działania algorytmu.