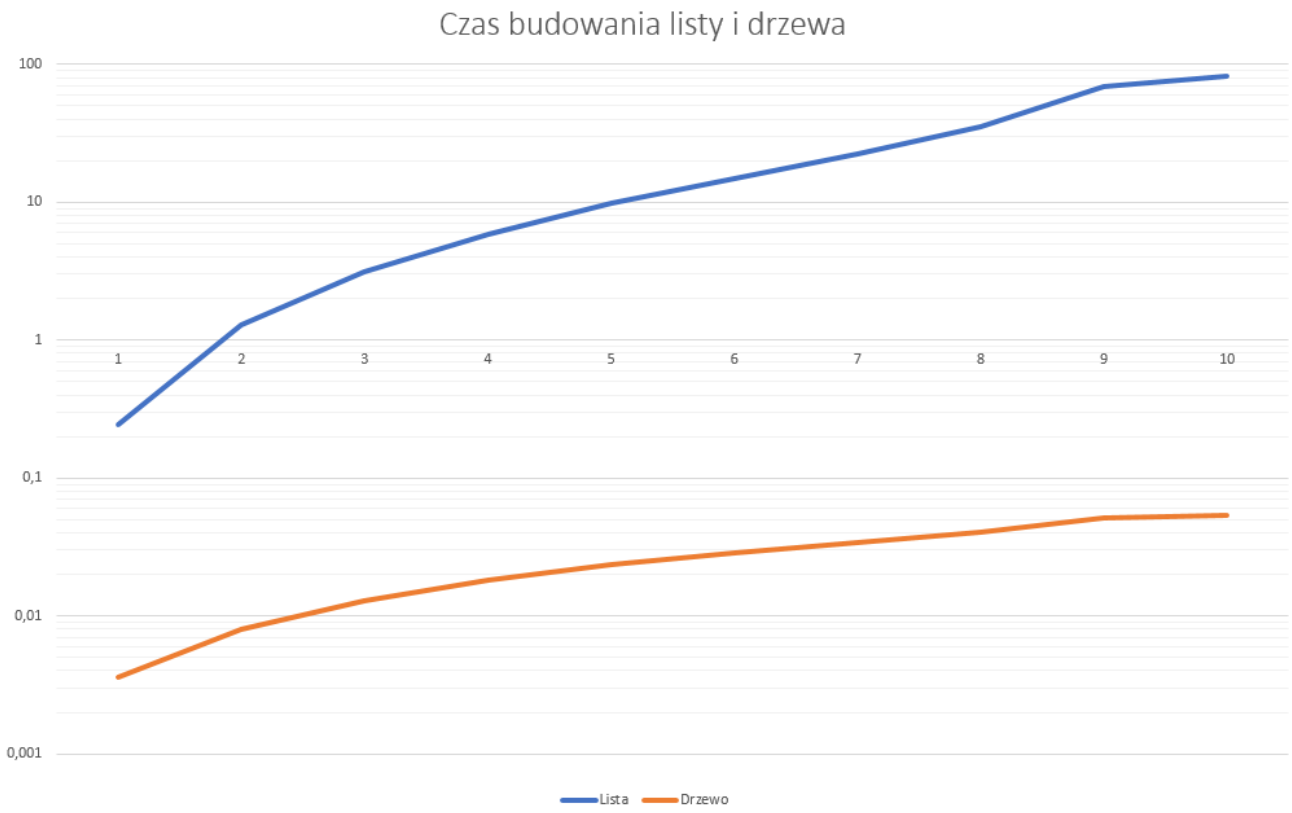
Zadanie 2. – złożone struktury danych

Grupa I1, Jan Techner (132332), Sebastian Maciejewski (132275)

Wszystkie wykresy są zrealizowane w podobny sposób: na osi oY oznaczono czas działania algorytmów w sekundach, zaś na osi oX przedstawiono ilość danych (10 - 15 kroków po 10 000 liczb). Wszystkie pomiary wykonano dla przynajmniej 10 próbek danych (czas ich działania jest średnią czasów dla poszczególnych próbek). Na niektórych wykresach konieczne było zastosowanie skali logarytmicznej, aby zaprezentować na jednym wykresie dane ze stosunkowo szerokiego przedziału.

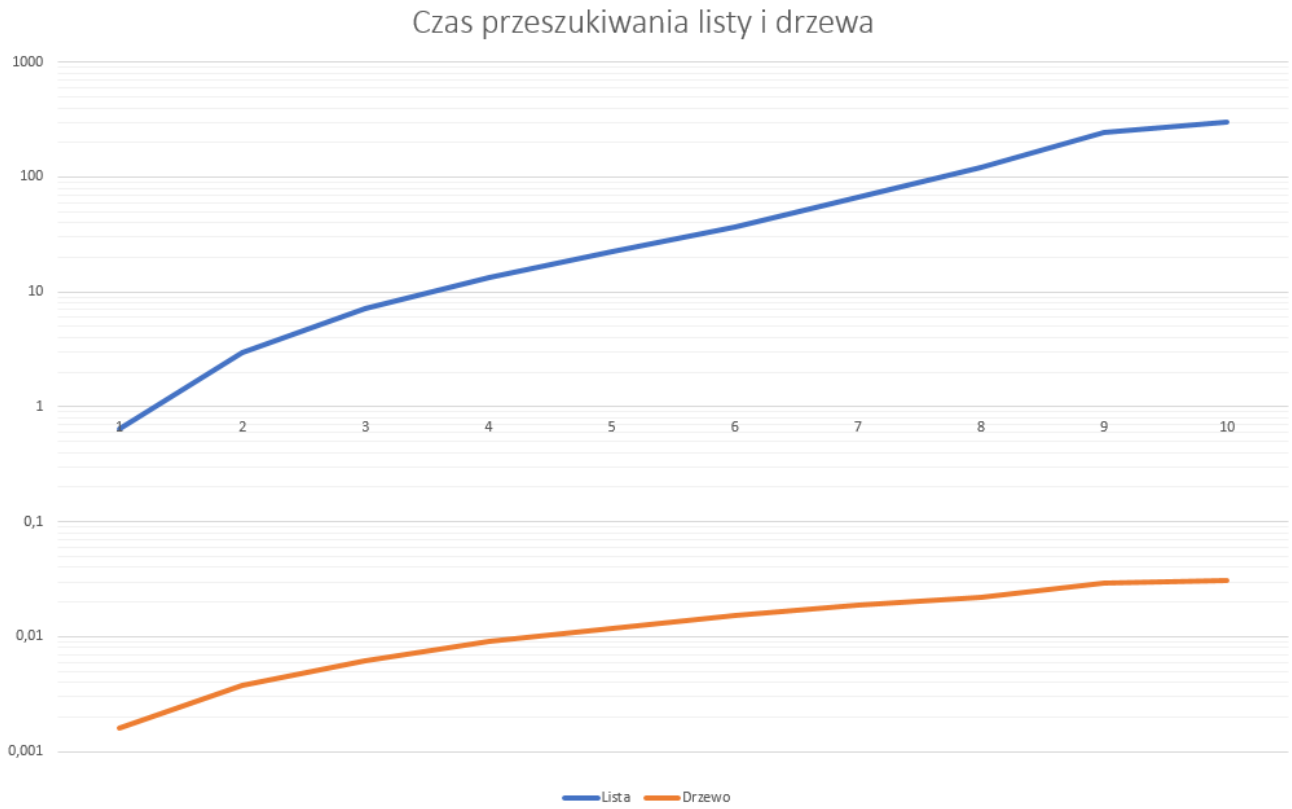
Zacznijmy od porównania drzewa przeszukiwań binarnych (BST) z listą jednokierunkową. Na początek spójrzmy na przedstawiony na wykresie czas tworzenia tych struktur.



Widać tutaj na pierwszy rzut oka dużą różnicę w czasie działania dla algorytmu budowania drzewa i algorytmu budowania i sortowania listy. Tak duża rozbieżność wynika z konieczności przeprowadzenia dużej ilości porównań elementów (co jest czasochłonną operacją) podczas sortowania listy. Zastosowane przez nas sortowanie podczas budowania listy to de facto sortowanie przez wstawianie, stąd nie dziwi kształt krzywej przedstawiającej czas budowania listy – przypomina ona wykres sortowania przez wstawianie. Stąd można wywnioskować, że złożoność obliczeniowa budowania z sortowaniem listy jednokierunkowej wynosi $O(n^2)$, mimo, że złożoność obliczeniowa samego dodawania elementów do listy jest liniowa. Natomiast w przypadku drzewa BST złożoność obliczeniowa jest znacznie mniejsza, co wynika z faktu, że podczas wstawiania elementu do drzewa (zakładając, że jest to średni przypadek i drzewo jest stosunkowo wyważone, co jest zgodne z prawdą przy danych losowych) ilość porównań i „głębokość” na jaką trzeba zejść, żeby umiejscowić element

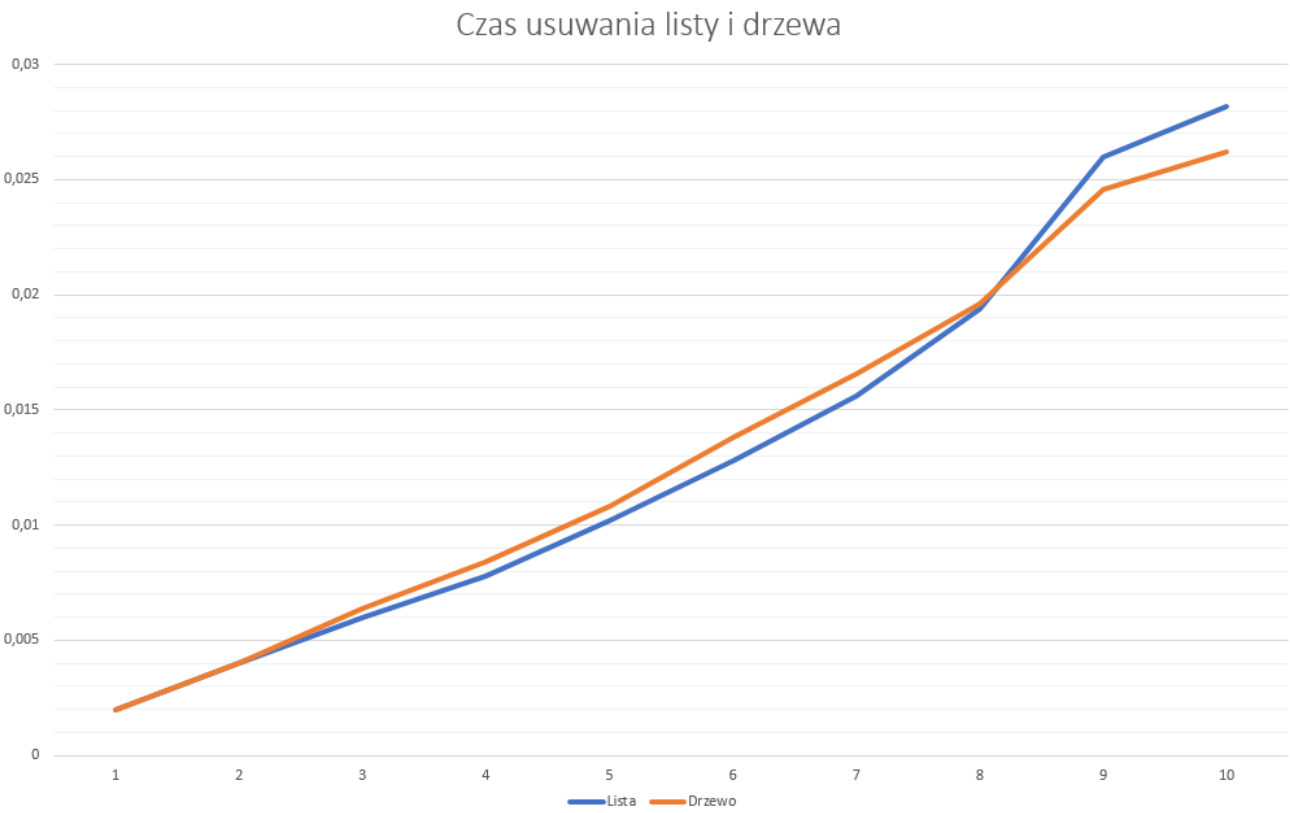
na swoim miejscu jest równa lub zbliżona do logarytmu przy podstawie 2 z ilości dotychczas wstawionych elementów. Dlatego też złożoność obliczeniowa zbudowania całego drzewa składającego się z n elementów wynosi $O(n\ log_2 n)$, co widać na wykresie.

Zajmijmy się teraz czasem przeszukiwania drzew i list. Zależności między złożonością czasową tych operacji przedstawione są na poniższym wykresie.



Gdy spojrzy się na ten wykres, od razu można wywnioskować, że przeszukiwanie listy jednokierunkowej jest bardzo złożoną operacją – wyszukanie kolejno wszystkich unikalnych wartości 100 000 elementów listy zajmuje ponad 5 minut. Dla porównania – wyszukanie wszystkich elementów drzewa zajmuje tylko 0,03 sekundy. Można się zastanowić skąd bierze się olbrzymia różnica 4 rzędów wielkości (!) między czasem działania BST a listy. Odpowiedź jest prosta – przeszukiwanie listy jednokierunkowej polega na przebieganiu listy od początku dla każdego z wyszukiwanych elementów, co jest bardzo kosztowne czasowo, zaś przeszukiwanie drzewa w najgorszym przypadku polega na przeszukaniu wszystkich elementów, podobnie jak w przypadku listy, ale na ogół polega na przejściu przez znacznie mniejszą ilość elementów. Ma to swoje odzwierciedlenie w złożoności obliczeniowej operacji wyszukiwania pojedynczego elementu – w przypadku listy jest to zawsze złożoność O(n), podobnie jak pesymistyczna złożoność przeszukiwania drzewa (w przypadku, w którym każdy węzeł w drzewie ma tylko jeden następnik), ale dla optymistycznego przypadku złożonością operacji przeszukania BST jest O(log(n)). Biorąc pod uwagę fakt, że wykonywaliśmy aż 10 pomiarów dla każdej ilości danych, możemy stwierdzić, że obie operacje przeszukiwania działały na danych będących przybliżeniem średniego przypadku, co pozwala stwierdzić o wiarygodności danych przedstawianych na wykresie.

Znając już różnice pomiędzy operacjami budowania i przeszukiwania drzew, możemy zająć się operacją usuwania obu struktur danych (polegającą na usuwaniu pierwszego elementu listy i usuwaniu elementów drzewa w porządku wstecznym. Oto wykres przedstawiający te zależności.



Przedstawiony wykres znacznie różni się od dwóch poprzednich, głównie z powodu małej różnicy między seriami danych (brak konieczności stosowania skali logarytmicznej). Widać na nim, że czas usuwania drzewa i listy jest prawie identyczny, co w zasadzie nie powinno nas dziwić – procedura usuwania pierwszego elementu listy polega na przypisaniu drugiemu elementowi listy pozycji „głowy" i usunięciu pierwszego elementu, zaś procedura usuwania drzewa w porządku wstecznym sprowadza się do usuwania wskaźników na następniki węzłów przedostatniego poziomu i usuwaniu tych następników. Obie procedury polegają w zasadzie na tych samych operacjach (w przypadku drzewa dokonuje się usunięcia dwóch elementów na każdy węzeł, stad minimalna różnica na korzyść listy do pewnego momentu), zatem taki kształt krzywych na wykresie nie powinien na dziwić. Te różnice stają się szczególnie mało istotne, gdy zwrócimy uwagę na bardzo małe jednostki na osi oY – usunięcie struktury o długości 100 000 elementów zajmuje poniżej 0,03 sekundy.

Po porównaniu złożoności czasowej operacji budowania, przeszukiwania i usuwania listy jednokierunkowej i drzewa przeszukiwań binarnych, możemy wysnuć wnioski dotyczące zalet i wad tych struktur.

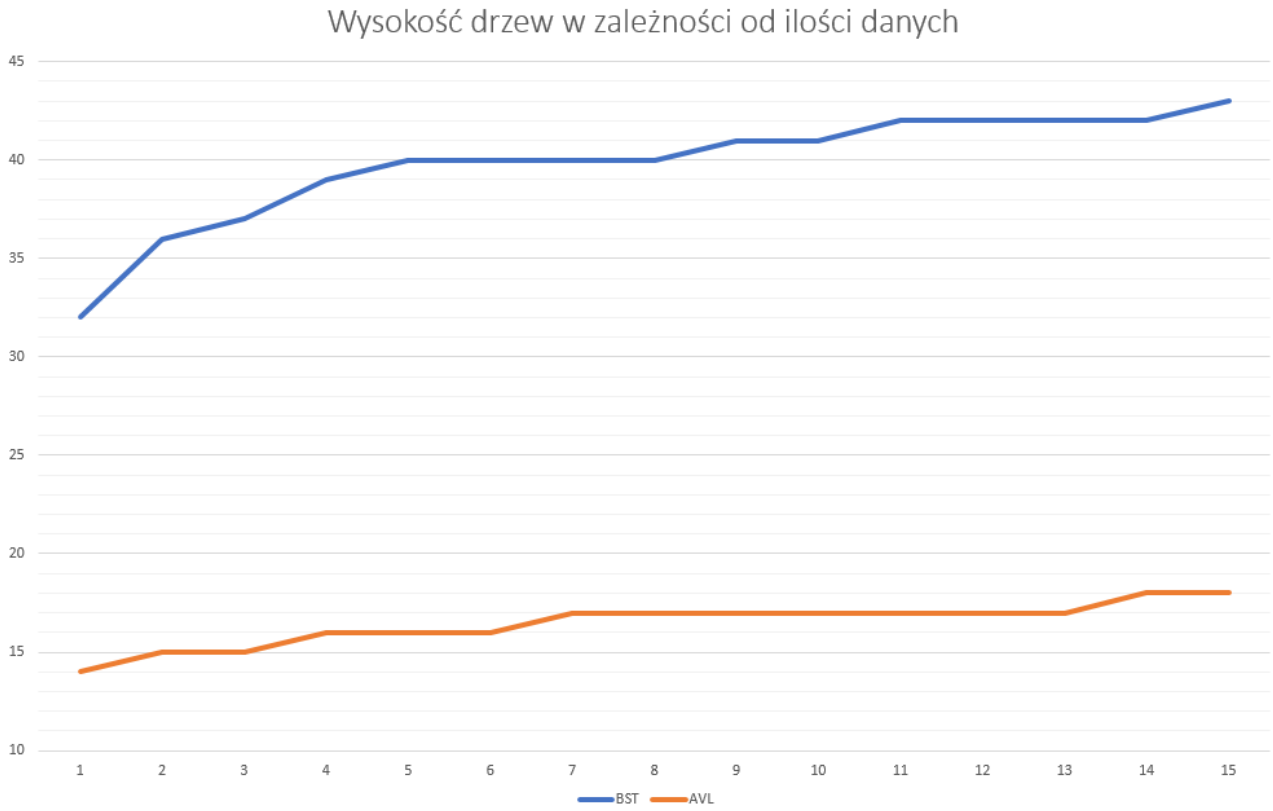
Zaletą BST jest głównie znacznie szybsze przeszukiwanie niż w przypadku listy oraz krótszy czas budowy. Drzewo jest też stosunkowo wygodne w obsłudze, dostęp do poszczególnych elementów jest łatwiejszy niż w przypadku listy (na ogół wymaga mniejszej liczby operacji).

Z kolei wadą tej struktury danych, w porównaniu do listy jednokierunkowej, jest jej bardziej złożona budowa i trudniejsza implementacja. Dodatkowo BST nie sprawdzi się podczas organizowania danych, których nie można w żaden sposób uporządkować, ponieważ z założenia budowa takiego drzewa opiera się na serii porównań.

Lista jednokierunkowa, choć wydaje się być strukturą pod każdym względem gorszą od drzewa, ma też swoje zalety, do których należy głównie łatwiejsza implementacja (szczególnie gdy mówimy o dołączaniu / wstawianiu elementu) i możliwość posiadania elementów, które nie są w żaden sposób uporządkowane. O jej wadach pisaliśmy już wcześniej (głównie podczas opisywania wykresów).

Na uwagę zasługuje też wygoda korzystania z porównywanych struktur – lista jest strukturą bardziej intuicyjną, łatwiejszą do zrozumienia, stąd też łatwiej przeprowadzać na niej operacje wstawiania i usuwania pojedynczego elementu (sprowadza się to do przekierowania wskaźnika na dołączany element w przypadku wstawiania i analogicznie w przypadku usuwania). Dla stosunkowo małej ilości danych, lista sprawdza się dobrze, jednak problemy związane z dużą złożonością obliczeniową operacji pojawiają się większych ilościach danych. Drzewo z kolei dobrze daje sobie radę z większymi ilościami danych, jednak nie jest tak intuicyjną i łatwą w zrozumieniu strukturą. Wszystkie operacje drzewa mają wyższy stopień skomplikowania niż operacje listy, co sprawia, że drzewo jest ogólnie trudniejsze do zaimplementowania.

Skoro dokonaliśmy już ogólnego porównania listy jednokierunkowej i drzewa przeszukiwań binarnych, możemy zająć się zagadnieniem wyważania drzewa. Poniższy wykres obrazuje różnice między wysokościami drzewa BST i wyważonego drzewa AVL, na osi oX standardowo ukazana jest ilość elementów (15 kroków po 10 000), zaś na osi oY przedstawiona jest wysokość tych drzew.



Jak widać z wykresu, wyważanie drzewa znacznie zmniejsza jego wysokość. Warto zauważyć, że wysokość zrównoważonego drzewa AVL odpowiada wartości logarytmu o podstawie 2 z ilości węzłów drzewa. Wynika to z algorytmu równoważenia drzewa, który polega na zastosowaniu metody połowienia binarnego. Redukuje on zbyt długie gałęzie, które mogą się utworzyć podczas tworzenia drzewa BST i sprawia, że długości dowolnych dwóch gałęzi różnią się maksymalnie o 1.

Głównym celem wykonywania operacji wyważania drzewa BST jest znaczne przyspieszenie działania operacji wyszukiwania. Niestety, budowa drzewa AVL jest kosztowna czasowo, dlatego jej zastosowanie ma sens głównie dla dużych zestawów danych statycznych, ponieważ zmiany w danych (dodawanie / usuwanie elementów) wymagają na ogół ponownego wyważania całego drzewa.

Operacje, których czas działania mierzyliśmy to przykłady działań na dynamicznej strukturze danych, ukazuje to znaczne różnice pomiędzy poszczególnymi dynamicznymi sposobami przechowywania danych. Zastanówmy się jednak nad ogólną różnicą pomiędzy dynamicznymi i statycznymi strukturami danych i zaletami jednych i drugich.

Struktury statyczne bazują na zadeklarowanej z góry ilości dostępnej pamięci, co znacznie utrudnia jej wykorzystywanie w przypadku zmieniających się zbiorów do których trzeba dodawać elementy lub je usuwać. Przykładem takiego sposobu przechowywania danych jest np. tablica, której główną zaletą jest łatwy dostęp do każdego elementu w stałym czasie. Dzięki ustalonemu od początku rozmiarowi i uporządkowaniu danych statycznych, można je w sposób szybszy przetwarzać.

Z kolei struktury dynamiczne, w odróżnieniu od statycznych, charakteryzują się zmienną ilością zajmowanej pamięci (do struktury można „dołożyć” danych). Takimi właśnie strukturami są analizowane przez nas drzewa przeszukiwań binarnych i listy – bazują one na systemie wskaźników do kolejnych elementów struktury, przez co istnieje możliwość dodania w dowolnym miejscu dowolnej ilości elementów. Tego typu struktury mają liczne zalety, poczynając od najbardziej oczywistych, takich jak nieograniczony rozmiar, możliwość zwalniania pamięci po usuwaniu elementów, a kończąc na różnych możliwościach usprawniania wyszukiwania elementów w tych strukturach (np. poprzez równoważenie drzew BST). Wadą dynamicznego podejścia do alokowania pamięci może być brak ustalonego miejsca przechowywania danego elementu (w odróżnieniu od np. tablicy) oraz większe trudności z implementacją niż w przypadku statystycznych struktur danych.