

Projekt 3 - Operacje na drzewie BST

Aleksander Dygon - 151856, Wiktor Ząbek - 151947

Wprowadzenie

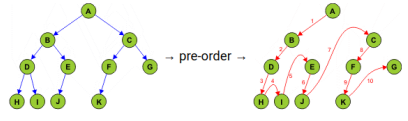
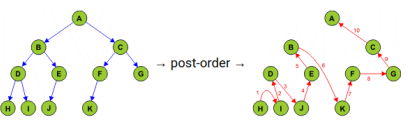
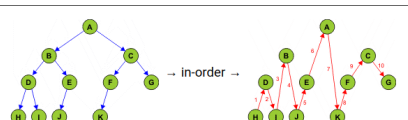
Celem tego projektu jest implementacja kodu poszukiwań drzewa binarnego. Binarny Search Tree to dynamiczna struktura danych; dzieli się na korzeń, gałęzie i liście - każdy podział nazywamy węzłem. Węzły mają swoją wartość klucza, to na jego podstawie określone jest gdzie dany element ma trafić; przechowują także wskaźniki na swoich "synów" (lewego i prawego) oraz na swojego "ojca".

Węzły posiadają dwa poddrzewa:

- lewe, które zawiera jedynie elementy o mniejszym kluczu niż klucz węzła
- prawe, w którym znajdują się elementy o kluczach nie mniejszych niż klucz węzła



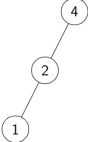
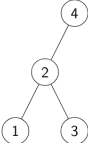
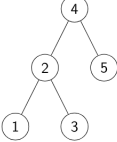
Drzewa binarne mają trzy podstawowe sposoby przeglądania:

- preorder (wzdłużne)
- postorder (poprzeczne)
- inorder (poprzeczne)

	PreOrder: A B D H I E J C F K G
	PostOrder: H I D J E B K F G C A
	InOrder: H D I B J E A K F C G

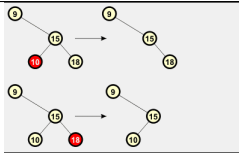
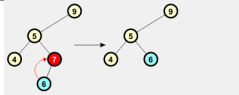
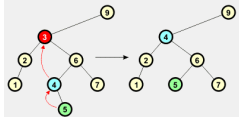
Wizualizacja operacji

Dodawanie

Stan	Opis
	Dodanie elementu o kluczu 4 jako korzenia drzewa.
	Dodanie elementu o kluczu 2. Ponieważ 2 jest mniejsze niż 4, element zostaje umieszczony w lewym poddrzewie korzenia.
	Dodanie elementu o kluczu 1. Ponieważ 1 jest mniejsze niż 4 oraz mniejsze niż 2, element zostaje umieszczony w lewym poddrzewie węzła 2.
	Dodanie elementu o kluczu 3. Ponieważ 3 jest mniejsze niż 4, ale większe lub równe 2, element zostaje umieszczony w lewym poddrzewie korzenia i w prawym poddrzewie węzła 2.
	Dodanie elementu o kluczu 5. Ponieważ 5 jest większe lub równe 4, element zostaje umieszczony w prawym poddrzewie korzenia.

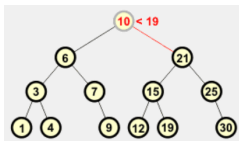
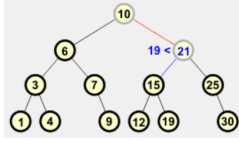
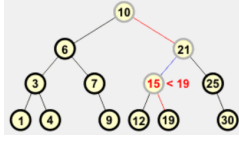
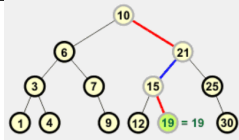
Usuwanie

Węzły usuwamy z drzewa BST, tak aby została zachowana hierarchia powiązań węzłów. Musimy rozpatrzyć kilka przypadków.

Stan	Opis
	Usuwany węzeł jest liściem, tzn. nie posiada synów. W takim przypadku po prostu odłączamy go od drzewa i usuwamy.
	Usuwany węzeł posiada tylko jednego syna. Węzeł zastępujemy jego synem, po czym węzeł usuwamy z pamięci.
	Najbardziej skomplikowany jest przypadek trzeci, gdy usuwany węzeł posiada dwóch synów. W takim przypadku znajdujemy węzeł będący następnikiem usuwanego węzła. Przenosimy dane i klucz z następnika do usuwanego węzła, po czym następnik usuwamy z drzewa – do tej operacji można rekurencyjnie wykorzystać tę samą procedurę lub zastąpić następnik przez jego prawego syna (następnik nigdy nie posiada lewego syna). Jako wariant można również zastępować usuwany węzeł jego poprzednikiem.

Przeszukiwanie

Drzewa BST pozwalają wyszukiwać zawarte w nich elementy z klasą złożoności obliczeniowej $O(\log n)$, gdzie n oznacza liczbę węzłów. Prześledźmy sposób wyszukiwania węzła o kluczu 19 w drzewie BST:

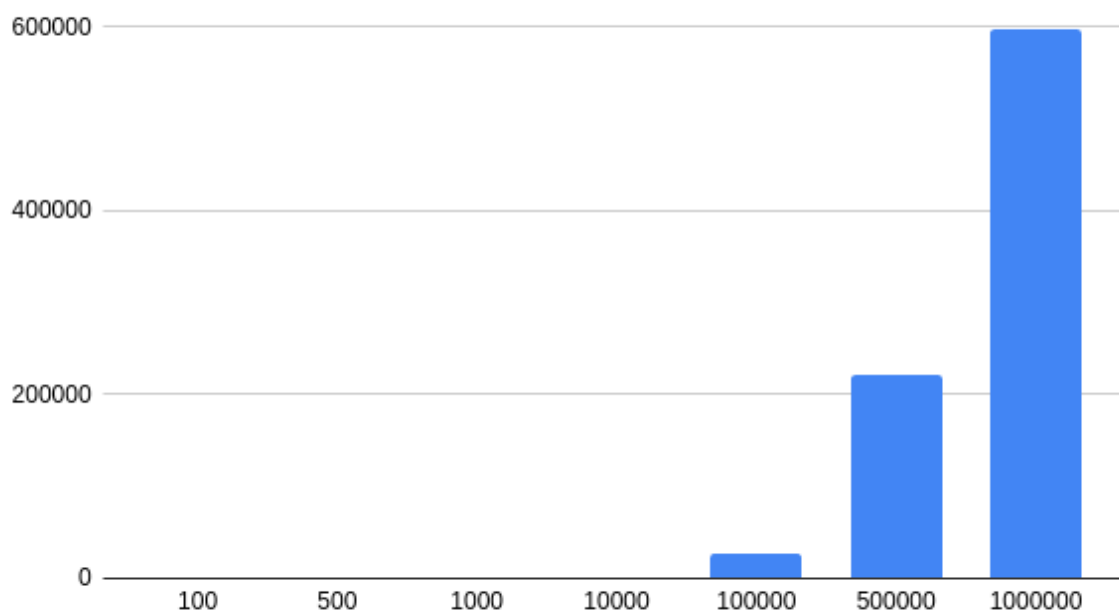
Stan	Opis
	Wyszukiwanie rozpoczynamy od korzenia drzewa. Porównujemy wartość węzła z wartością poszukiwaną. Ponieważ jest ona większa od wartości korzenia, idziemy wzdłuż prawej krawędzi do prawego syna (jeśli węzeł nie miałby prawego syna, to oznaczałoby to, że poszukiwanej wartości nie ma w drzewie BST).
	W węźle 21 ponownie dokonujemy porównania. Ponieważ poszukiwany węzeł jest mniejszy od 21, wybieramy gałąź lewą i przechodzimy do lewego syna 15.
	Porównujemy węzeł 15 z poszukiwanym 19. Ponieważ 19 jest większe, idziemy prawą krawędzią do prawego syna 19.
	Porównujemy węzeł 19 z poszukiwanym. Są równe. Wyszukiwanie zakończone.

Złożoność operacji

Podane testy zostały wygenerowane dla losowo wybranych danych z przedziału $[0, \text{SIZE}]$. Czasy przedstawione na wykresach są wyrażone w milisekundach.

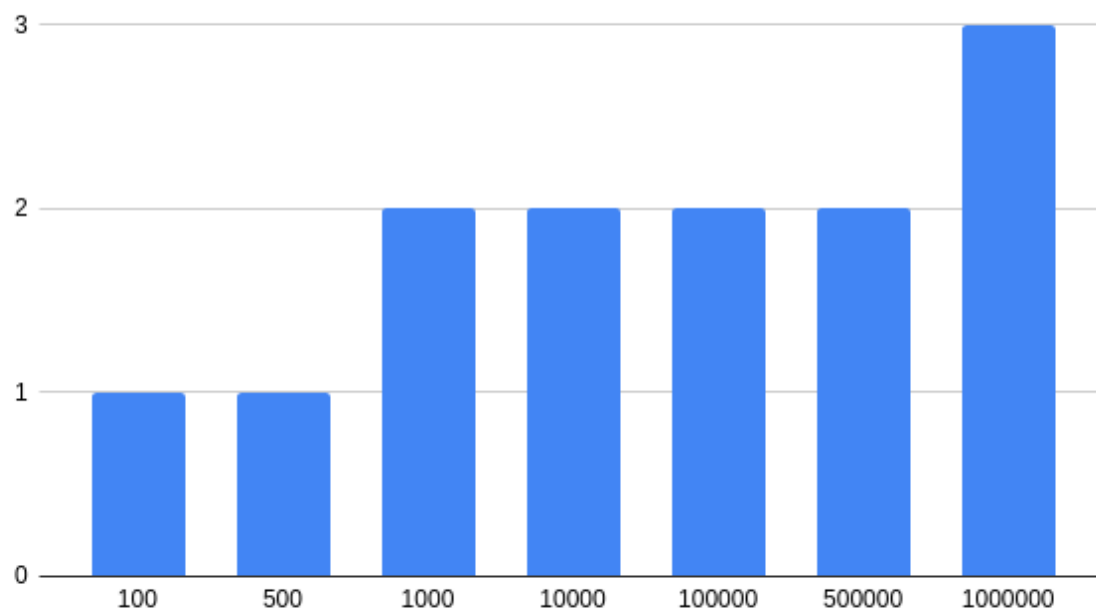
Dodawanie N elementów

Add elements



Dodawanie $N + 1$ elementu

add $n + 1$ element



Usuwanie

Delete lower element

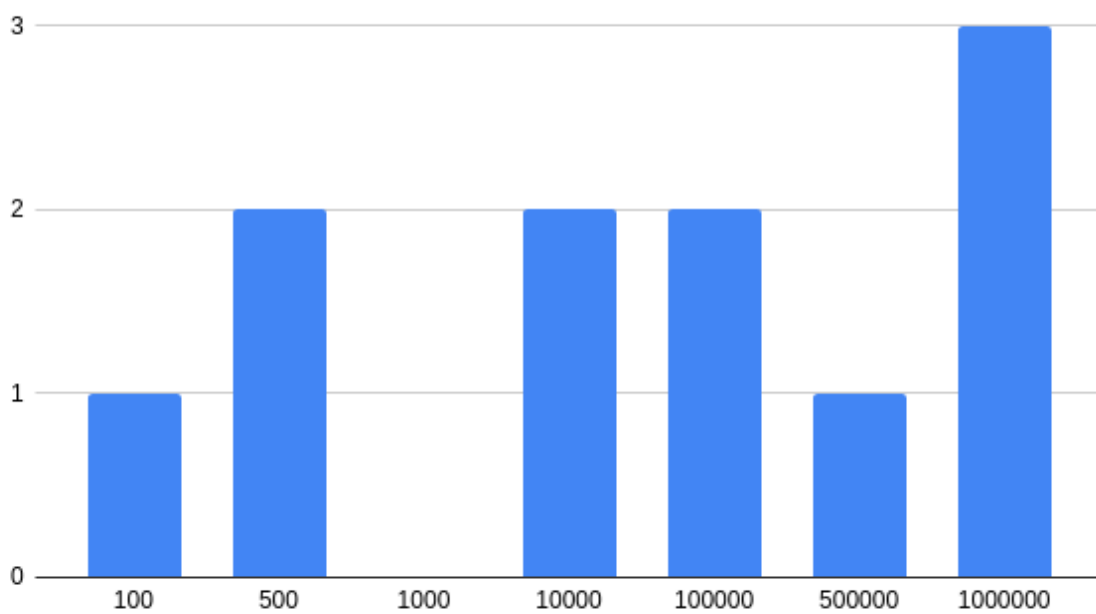


Figure 1: "Lower" oznacza element znajdujący się w dolnych 10 procent wysokości drzewa.

Delete middle element

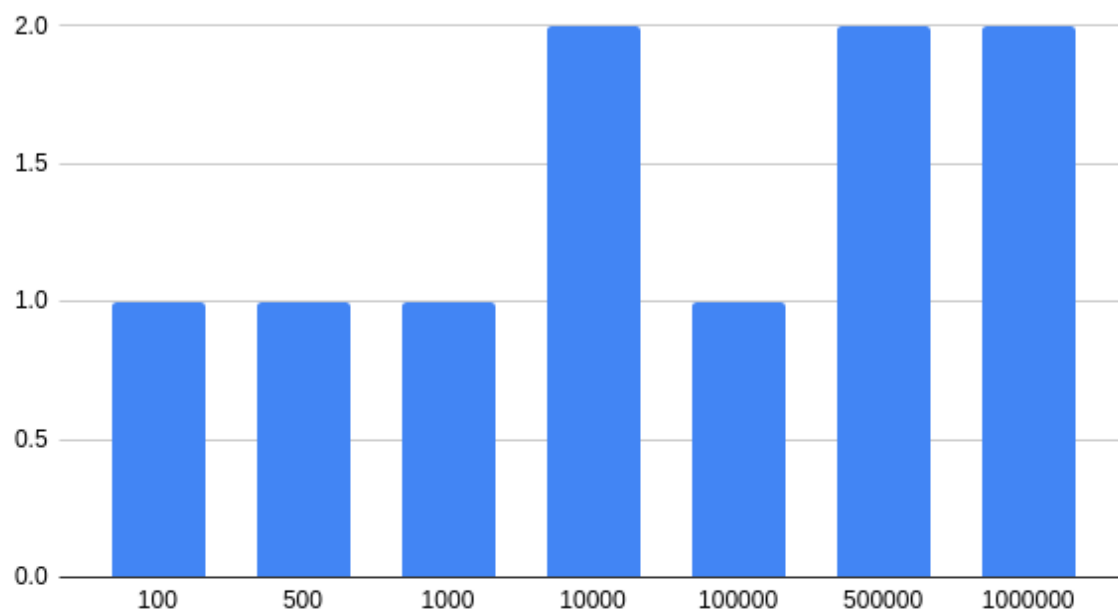
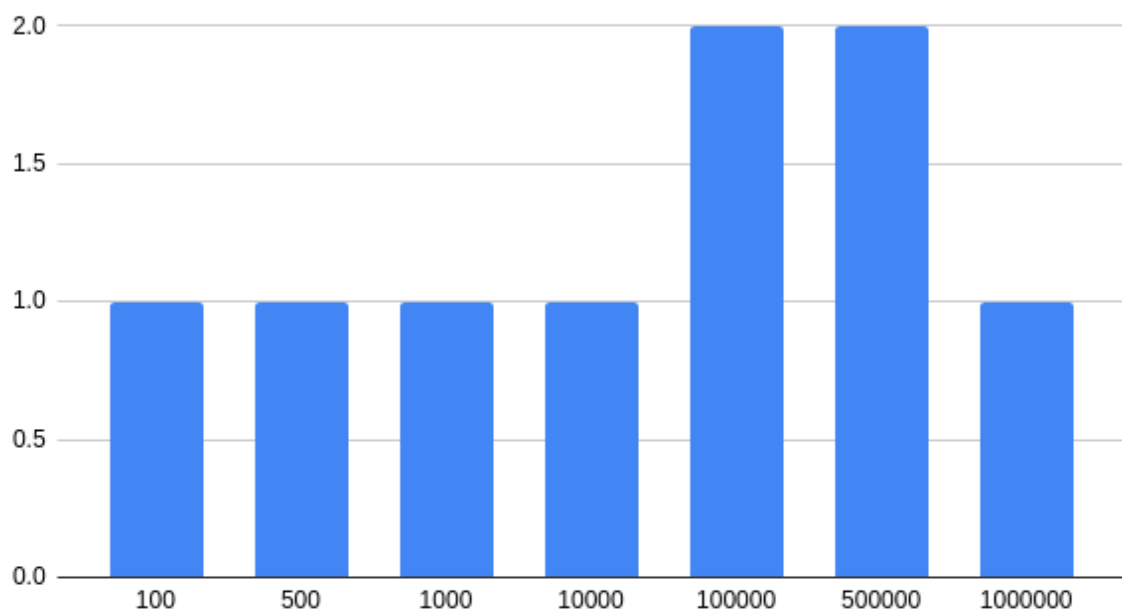


Figure 2: "Middle" oznacza element znajdujący się w zakresie od 10 procent do 40 procent wysokości drzewa.

Delete Root



Przeszukiwanie

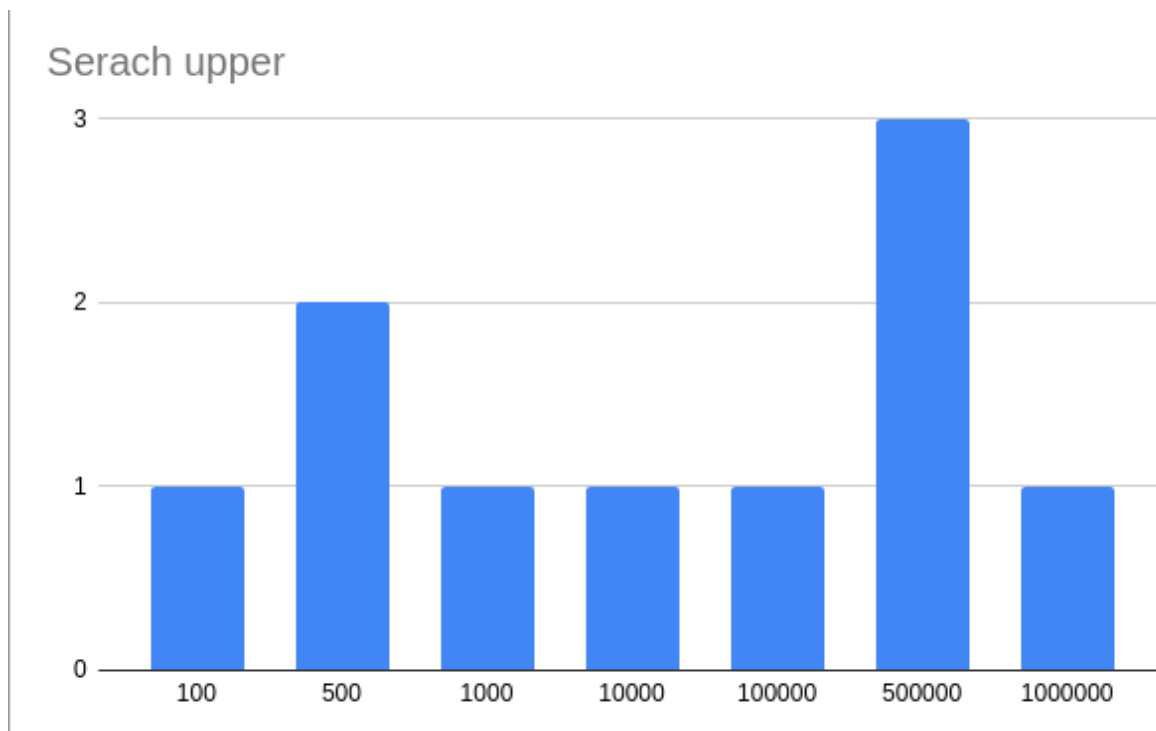


Figure 3: "Upper" oznacza element znajdujący się w zakresie od 40 procent do 100 procent wysokości drzewa.

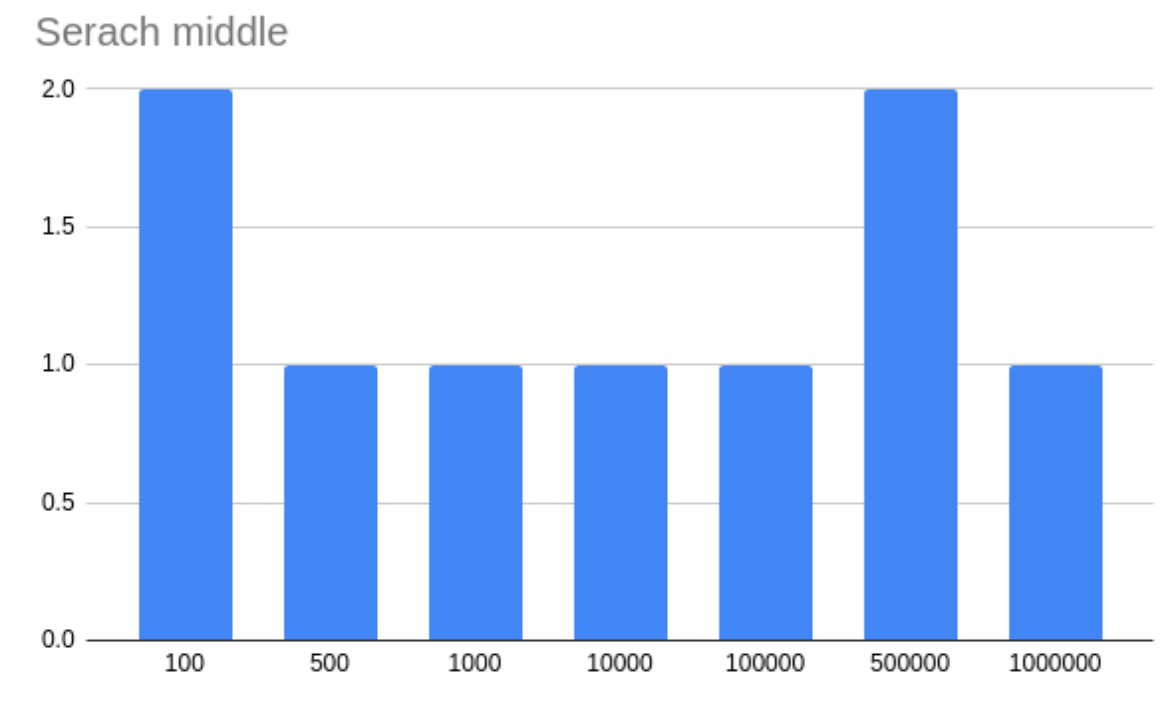


Figure 4: "Middle" oznacza element znajdujący się w zakresie od 10 procent do 40 procent wysokości drzewa.

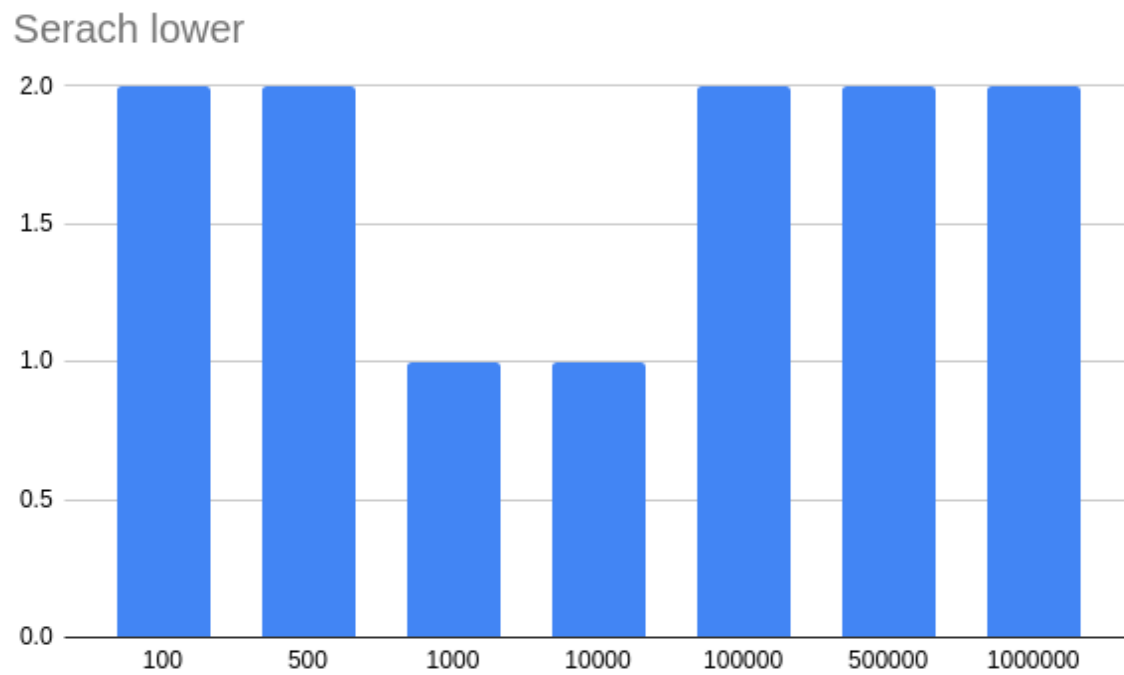
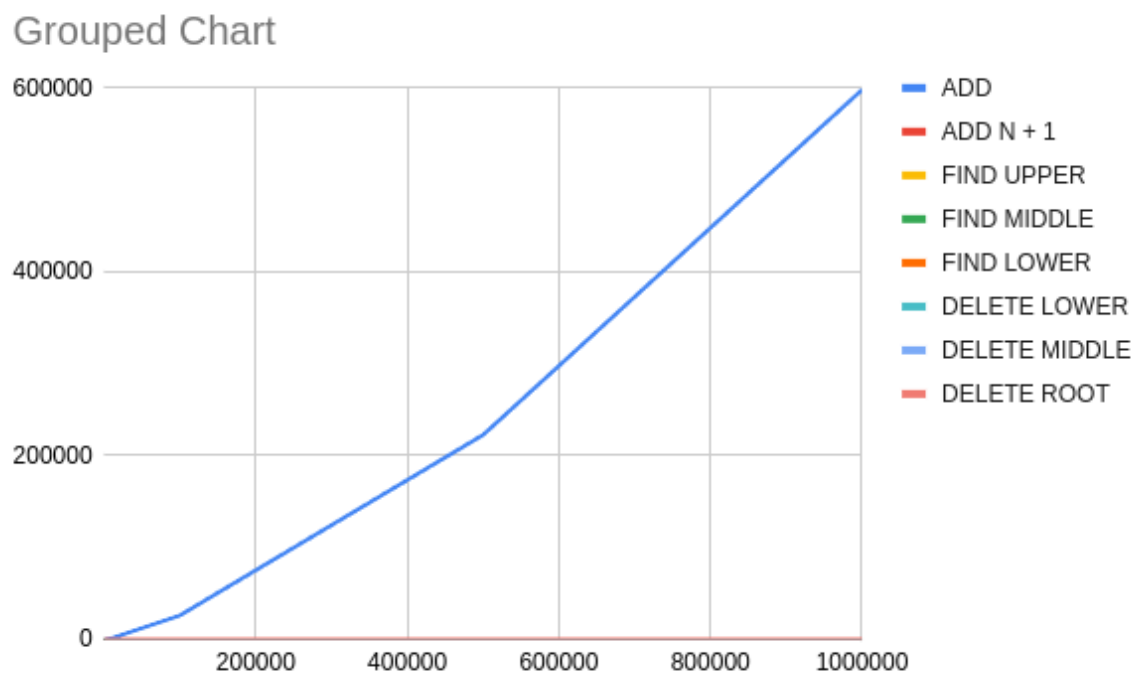


Figure 5: "Lower" oznacza element znajdujący się w dolnych 10 procent wysokości drzewa.

Wykres zbiorczy



Wyniki przeprowadzonych testów

[ms]	100	500	1000	10000	100000	500000	1000000
ADD	7	44	91	1284	26117	222522	598058
ADD N + 1	1	1	2	2	2	2	3
FIND UPPER	1	2	1	1	1	3	1
FIND MIDDLE	2	1	1	1	1	2	1
FIND LOWER	2	2	1	1	2	2	2
DELETE LOWER	1	2	0	2	2	1	3
DELETE MIDDLE	1	1	1	2	1	2	2
DELETE ROOT	1	1	1	1	2	2	1

Podsumowanie

Na podstawie przeprowadzonych testów oraz ich wyników widzimy wydajność drzewa BST po przeprowadzeniu konkretnych operacji względem tego ile jest zawartych w nim elementów. Wyniki testów wydajnościowych potwierdzają, że drzewo BST jest skuteczną strukturą danych do przechowywania i manipulowania zbiorem danych. Czas operacji dodawania wielu elementów gwałtownie wzrasta, w zależności od ich liczby. W przypadku dodawania, wyszukiwania czy też usuwania pojedynczego elementu czas ten nie przekracza 3[ms] co pokazuje jak bardzo drzewo BST jest efektywne w tego typu zabiegach.