

Datastrukturer – tidskompleksitet

AVLTree (self balancing BST)

	Første	sidste	midterste	i'te	næste
Læs et element ¹	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
Find element ²	eksisterer		Eksisterer ikke		
	$O(\log n)$		$O(\log n)$		
Indsæt nyt element	Den rette plads	i slutningen	i midten	efter node	før node
	$O(\log n)^*$	n/a^*	n/a^*	n/a^*	n/a^*
Fjern element	første	Sidste	i'te	efter node(child)	før node(parent)
	$O(\log n)^{**}$	$O(\log n)^{**}$	$O(\log n)^{**}$	$O(\log n)$	$O(\log n)$
Byt om på to elementer	første og sidste	første og i'te	sidste og i'te	i'te og j'te	nodes
	n/a^{***}	n/a^{***}	n/a^{***}	n/a^{***}	n/a^{***}

* Vi bestemmer ikke selv hvor det skal være da det bliver indsat det "rigtige" sted ud fra comparator funktionen.

** $O(\log n)$ for at finde den node der skal fjernes, $O(1)$ for at fjerne en node

*** giver ikke mening at bytte om på to elementer da træet er sorteret

¹ At læse et element er som regel det samme som at skrive nyt indhold i et eksisterende element

² Find et element med en bestemt værdi – alt efter om vi ved at listen er sorteret eller ej, og om elementet findes eller ej.