SinglyLinkedList– tidskomplexitet

Skemaer – til sammenligning

SinglyLinkedList har ingen direkte index-adgang som array → alle index-based operationer er O(n).

Tilføjelse/fjernelse ved head eller givet node er O(1), hvilket er den største fordel.

Traversal er den dominerende tidskompleksitet for næsten alle andre operationer.

SinglyLinkedList

Læs et element[1]	første	sidste	midterste	i'te	tidligere[2]
	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Find element [3]	eksisterer *usorteret liste*	eksisterer *sorteret liste*	eksisterer ikke *usorteret liste*	eksisterer ikke *sorteret liste*	
	$O(n)$	$O(n)$	$O(n)$	$O(n)$	
Indsæt nyt element	i starten	i slutningen	i midten	efter node	før node
	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$
Fjern element	første	sidste	i'te	efter node	før node
	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$
Byt om på to elementer	første og sidste	første og i'te	sidste og i'te	i'te og j'te	nodes
	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

[1] At læse et element er som regel det samme som at skrive nyt indhold i et eksisterende element
[2] Hvis vi allerede har fat i ét element i en datastruktur, kan vi måske læse det ”næste” hurtigere end i+1'te
[3] Find et element med en bestemt værdi – alt efter om vi ved at listen er sorteret eller ej, og om elementet findes eller ej.