SinglyLinkedList- tidskomplexitet

SingleLinkedList uden size og tail

En **SinglyLinkedList** består af noder, hvor hver node peger på den næste. Den har ikke direkte adgang via indeks, men gør det let at indsætte og fjerne elementer i starten af listen.

Pros:

- Indsættelse/fjernelse i starten: O(1) – Hurtigt, da kun referencen til første node skal ændres.
- Dynamisk størrelse: Listen vokser og reduceres efter behov, uden at kopiere elementer.
- Effektiv ved mange ændringer i starten: Ingen behov for at flytte eksisterende elementer.

Cons:

- Adgang via indeks: O(n) – Skal traversere listen fra starten for at finde et element.
- Indsættelse/fjernelse i midten eller slutningen: O(n) – Skal finde den rigtige position først.
- Ekstra hukommelse: Hver node kræver en ekstra reference (peger på næste node).

SinglyLinkedList

Læs et element[1]	første	sidste	midterste	i'te	næste[2]
	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$
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
	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(1)/O(n)$

***Nodes** Temp = node1.data node1.data=node2.data node2.data = temp

[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.