Stack- tidskomplexitet

Stack udfra singlyLinkedList med size og prev

En **Stack** (stak) er en lineær datastruktur, der følger princippet **LIFO** (*Last In, First Out*). Det betyder, at det sidste element, der indsættes, er det første, der fjernes.

Fordele (Pros):

- Effektiv indsættelse og fjernelse på første element: O(1) – Elementer tilføjes og fjernes fra toppen uden at flytte andre elementer.
- Simpel struktur: Let at implementere og forstå.
- Bruges i mange algoritmer: Fx funktionskald (call stack), backtracking og parsing.

Ulemper (Cons):

- Ingen tilfældig adgang: O(n) – Kun det øverste element kan tilgås direkte.
- Begrænset fleksibilitet: Kan kun indsætte og fjerne fra toppen.
- Kan løbe fuld: Hvis implementeret med et fast array, kræver manuel håndtering af kapacitet.

SinglyLinkedList

Læs et element[1]	første	sidste	midterste	i'te	tidligere[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
	$O(1)$	NA	NA	NA	NA
Fjern element	første	Sidste	i'te	Efter node	Før node
	$O(1)$	NA	NA	NA	NA
Byt om på to elementer	første og sidste	første og i'te	sidste og i'te	i'te og j'te	Nodes
	NA	NA	NA	NA	NA

[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.