

# SinglyLinkedList- tidskomplexitet

## Skemaer – til sammenligning

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 <sup>1</sup>	første	sidste	midterste	i'te	næste <sup>2</sup>
Find element <sup>3</sup>	eksisterer <i>usorteret liste</i>	eksisterer <i>sorteret liste</i>	eksisterer ikke <i>usorteret liste</i>	eksisterer ikke <i>sorteret liste</i>	
Indsæt nyt element	i starten	i slutningen	i midten		
Fjern element	første	Sidste	i'te		
Byt om på to elementer	første og sidste	første og i'te	sidste og i'te	i'te og j'te	
	$O(n)$	$O(n)$	$O(n)$	$O(n)$	

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

<sup>2</sup> Hvis vi allerede har fat i ét element i en datastruktur, kan vi måske læse det ”næste” hurtigere end i+1'te

<sup>3</sup> Find et element med en bestemt værdi – alt efter om vi ved at listen er sorteret eller ej, og om elementet findes eller ej.