Queue- tidskomplexitet

Queue udfra singlyLinkedList med size og tail

En **Queue** (kø) er en lineær datastruktur, der følger princippet **FIFO** (*First In, First Out*).

Fordele (Pros):

- Effektiv indsættelse og fjernelse: O(1) – Elementer tilføjes bagest (*enqueue*) og fjernes forrest (*dequeue*) uden behov for flytning.
- Forudsigelig rækkefølge: Bevarer rækkefølgen af elementer, hvilket gør den ideel til opgavekøer og buffere.
- Kan implementeres med både arrays og linked lists: Giver fleksibilitet i design.

Ulemper (Cons):

- Ingen tilfældig adgang: O(n) – Du kan ikke springe direkte til et element i midten.
- Begrænset funktionalitet: Kun adgang til forreste og bagerste element.
- Statisk størrelse i array-baserede køer: Hvis ikke dynamisk implementeret, kan den løbe fuld.

SinglyLinkedList

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