Datastrukturer – performance

Skemaer – til sammenligning

Udfyld et skema som det herunder med Big O estimater (time complexity) for de datastrukturer du lærer. Skriv også noter til dig selv om nogle af de antagelser du gør dig (for eksempel tager det *O(1)* at fjerne det sidste element i en arraylist, hvis arrayet ikke kopieres, men *O(n)* hvis det gør …)

Det er op til dig selv om du laver ét nyt dokument for hver struktur, eller laver nye sider i et samlet dokument.

Bemærk at skemaerne altid anvender n som det samlede antal af elementer i strukturen, og i som et enkelt, vilkårligt, element.

Data i datastrukturen omtales altid som elementer, mens de ”databærende enheder” kaldes nodes. Et array og en array list har for eksempel ikke nodes, men en linked list eller et tree har altid, og nogle operationer er markant hurtigere hvis man allerede har adgang til en nabo-node.

# Datastrukturnavn

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Læs et element[[1]](#footnote-1) | første | sidste | midterste | i'te |  |
|  |  |  |  |  |
| Find element [[2]](#footnote-2) | eksisterer  *usorteret* | eksisterer *sorteret* | eksisterer ikke *usorteret* | eksisterer ikke *sorteret* |  |
|  |  |  |  |  |
| Indsæt nyt element | i starten | i slutningen | i midten | efter node | før node |
|  |  |  |  |  |
| Fjern element | første | sidste | i'te | efter node | før node |
|  |  |  |  |  |
| Byt om på to elementer | første og sidste | første og i’te | sidste og i’te | i’te og j’te | nodes |
|  |  |  |  |  |

1. At læse et element er som regel det samme som at skrive nyt indhold i et eksisterende element [↑](#footnote-ref-1)
2. Find et element med en bestemt værdi – alt efter om vi ved at listen er sorteret eller ej, og om elementet findes eller ej. [↑](#footnote-ref-2)