Array – tidskomplexitet

Skemaer – til sammenligning

Arrayes er immutable (fast længde), fordi den allokerer en fast mængde i computerens hukommelse.

Det fungere ret godt, hvis man kender størrelse på det man skal bruge, men er ret rigid, hvis man finder ud af der skal mere i.

# Array

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Læs et element[[1]](#footnote-1) | første | sidste | midterste | i'te | næste[[2]](#footnote-2) |
| *O(1)* | *O(1)* | *O(1)* | *O(1)* | O(n)  O(1)-hvis vi kender index |
| Find element [[3]](#footnote-3) | eksisterer  *usorteret liste* | eksisterer *sorteret liste* | eksisterer ikke *usorteret liste* | eksisterer ikke *sorteret liste* |  |
| *O(n)* | *O(log n)* | *O(n)* | *O(log n)* |  |
| Indsæt nyt element | i starten | i slutningen | i midten |  |  |
| *n/a* | *n/a* | *n/a* |  |  |
| Fjern element | første | sidste | i'te |  |  |
| *n/a* | *n/a* | *n/a* |  |  |
| Byt om på to elementer | første og sidste | første og i’te | sidste og i’te | i’te og j’te |  |
| *O(1)* | *O(1)* | *O(1)* | *O(1)* |  |

1. At læse et element er som regel det samme som at skrive nyt indhold i et eksisterende element [↑](#footnote-ref-1)
2. Hvis vi allerede har fat i ét element i en datastruktur, kan vi måske læse det ”næste” hurtigere end i+1’te [↑](#footnote-ref-2)
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. [↑](#footnote-ref-3)