Datastruktur Array – performance

Array (traditionelt)

Et traditionelt array har altid en fast størrelse, og kan hverken blive større eller mindre, og giver derfor heller ikke mulig for at skubbe elementer ind i rækken, eller fjerne dem igen.

Samtidig har et array ikke nodes, så det giver ikke mening at tale om dem

*Et array i JavaScript er en helt anden konstruktion – mere sammenligneligt med et HashMap – så her vil langt de fleste operationer være O(1).*

# Array

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Læs et element[[1]](#footnote-1) | første | sidste | midterste | i'te |  |
| *O(1)* | *O(1)* | *O(1)* | *O(1)* |  |
| Find element [[2]](#footnote-2) | eksisterer  *usorteret* | eksisterer *sorteret \*\** | eksisterer ikke *usorteret* | eksisterer ikke *sorteret* |  |
| *O(i) \** | *O(log n)* | *O(n)* | *O(log n)* |  |
| Indsæt nyt element | i starten | i slutningen | i midten | efter node | før node |
| *na* | *na* | *na* |  |  |
| Fjern element | første | sidste | i'te | efter node | før node |
| *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 |
| *O(1)* | *O(1)* | *O(1)* | *O(1)* |  |

\*) at finde et element i et usorteret array tager præcis så lang tid som elementets position i arrayet – i princippet altså O(n)

\*\*) går ud fra at der anvendes binary search til at finde et element i et sorteret array

\*\*\*) ombytninger kræver altid en ekstra, midlertidig variabel – så space complexity er +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. 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)