- Measures how runtime scales with input size

O(1) ⇒ Constant time

- Accessing an array by index
- Access a hash table

O(Log n) ⇒ Logarithmic time
⇒ Runtime increases slowly as input grows

- Binary search on sorted array
- Performing operations on a balanced binary tree

$O(n)$ ⇒ Linear time

⇒ Runtime grows with input as each element is touched once

- Finding min, max elements in an unsorted array
- Checking if an element exists in an unsorted array

$O(n \log n)$ ⇒ Linearithmic time

⇒ Runtime of most efficient sorting algorithms (merge sort, quick sort, heap)

$O(x^2)$ ⇒ Quadratic time

- Bubble sort
- Nested loops

$O(n^3)$ ⇒ Cubic time

- Naive matrix multiplication
- Triple nested loops

$O(2^n)$ ⇒ Exponential time

- Recursive algorithms

$O(n!)$ ⇒ Factorial time

- Permutation-generation problems
  ↳ Very inefficient for trivial input size

Important to note that big-O is not everything ⇒ its important to consider how algorithms work in the memory / rest of the system

Array Access

Although accessing a 2D array is $O(n^2)$, accessing by rows is faster than accessing by collumns!

Reading row-wise maximises sequential access ⇒ More cache friendly

Linked list vs Array

- Both are linear time
- Accessing an array is faster
  - ↳ Array elements are closer together
  - ↳ Linked-list nodes are often scattered in memory