Module 14

Algorithmic Complexity

Motivation: How efficient an algorithm is ?

How much work an algorithm requires in order to solve a problem?

Cost of a Problem: Cost C of a problem is the no. of computations required to solve a problem.

Size of Problem: The size of a problem 'n' describes how many items are involved in the problem.
(Ex: # no pixels in the image, no. of items / list)

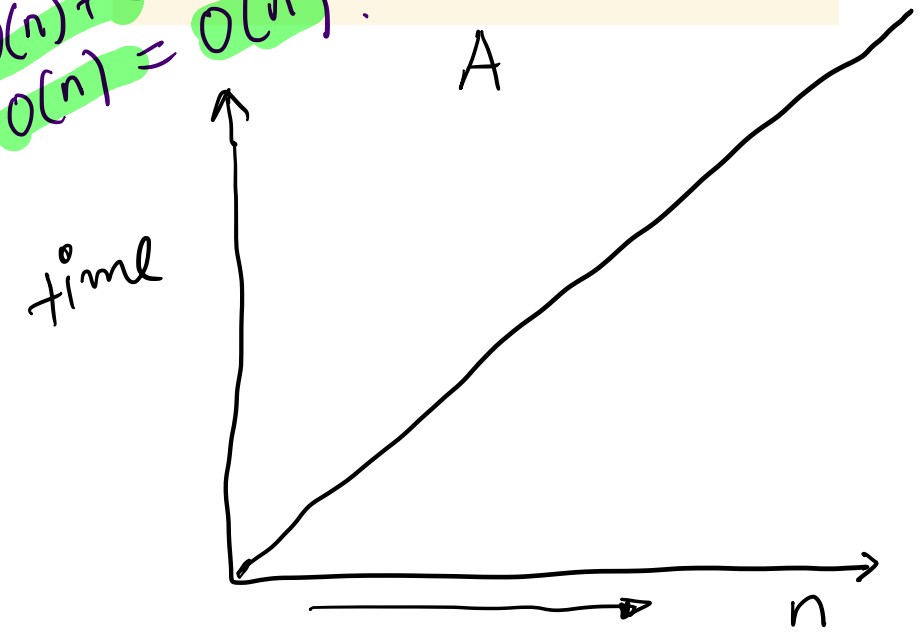
$$C = f(n)$$

Ex 1: 1+2+3+…+n= n(n+1)/2

Source: https://rithmschool.github.io/function-timer-demo/

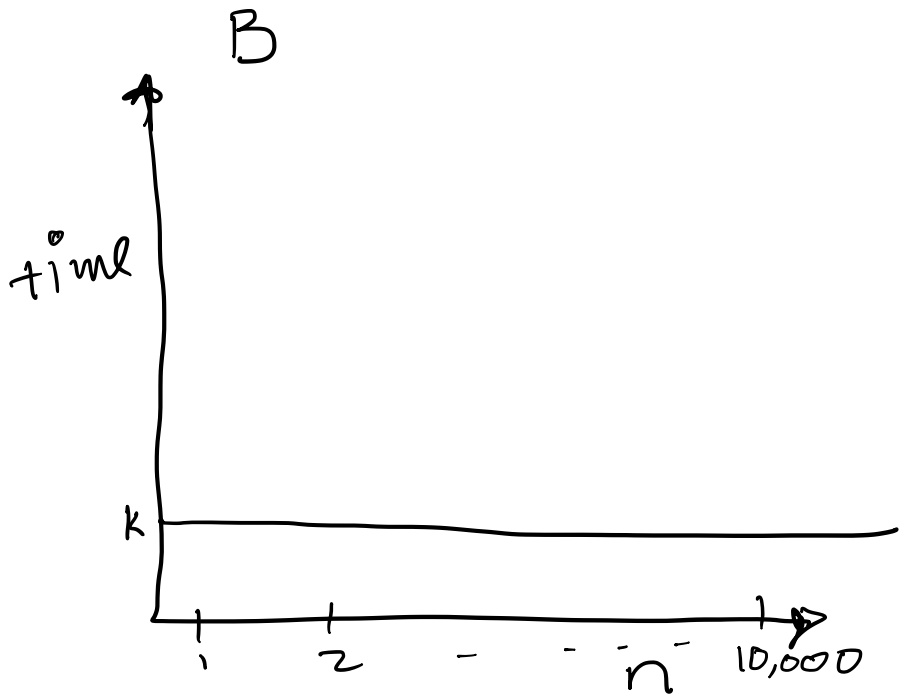
```
function addUpToFirst(n) {
  var total = 0;
  for (var i = 0; i <= n; i++) {
    total += i;
  }
  return total;
}
```

O(n)+O(n) = O(n)




```
function addUpToSecond(n) {
  return n * (n + 1) / 2;
}
```

O(k) or O(1)



Issues with timer.

* Different machines will record different time.
* Some machines may not precisely measure time.

* We want our code to perform BEST when $n \to \infty$ [n is very very large]

Use: Ordered Notation (O-Notation)

Goal: to approximate computation cost to the Input size n.

Algebric Foundation: Order (Degree) of Polynomial is the largest exponent.

$f(x) = 4x^4 + 5x^2 + x + 200$ $\quad O(x^4)$

$f(n) = n^3 + 6000$ $\quad O(n^3)$

$O(n^3) + O(1) = O(n^3)$

Big O Notation: addresses worst case cost.

The notation $f(n) = O(g(n))$ means there are positive # of c and m such that:

$$|f(n)| \leq c\, g(n) \quad \text{for all } n \geq m.$$

$5n^2 = O(n^2)$

$100n^3 + 10n^2 + 60 = O(n^3)$.

Ranking Order (Source: Desmos)

$O(k) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^k)$

$O(k)$ ↓ $O(1)$

$< O(k^n) < (n!)$

Ex 2: Algorithmic A and B do the same job. Algorithmic A is $O(n^2)$ and Algorithmic B is $O(n^3)$. Which is better?

$O(n^2) < O(n^3)$

Algorithm A runs faster which means A is BETTER.

Which Algorithm is BETTER?

1. run time / speed / faster ← time complexity.
2. less memory ← space complexity.
3. readable code / user freindly.

Ex 3: Determine the order notation of the following:

Source: https://rithmschool.github.io/function-timer-demo/

time vs Input.

```
function printAllPairs(n) {
  for (var i = 0; i < n; i++) {
    for (var j = 0; j < n; j++) {
      console.log(i, j);
    }
  }
}
```

n
n

nested loop.

$$O(n) \cdot O(n) = O(n^2)$$

Combining the orders:

a) $O(n^j) \times O(n^k) = O(n^{j+k})$

b) $O(n^j) + O(n^k) =$ largest of $O(n^j)$ or $O(n^k)$

c) $(O[n^j])^k = \underbrace{O(n^j) \times O(n^j) \cdots O(n^j)}_{k} = O(n^{jk})$