Asymptotic Analysis

- Runtime Analysis and Big Oh - $O(\cdot)$
- Complexity Classes and Curse of Exponential Time
- $\Omega(\cdot)$, $\Theta(\cdot)$, $o(\cdot)$, $\omega(\cdot)$ - Relational properties

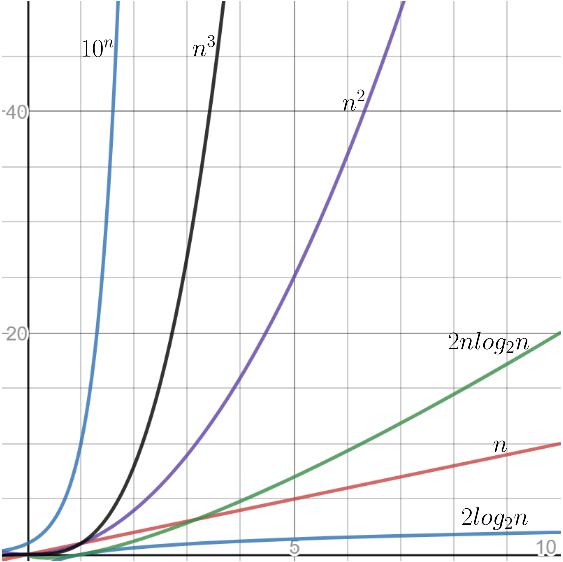
Imdad ullah Khan

Asymptotic-Complexity Classes

Class Name	**Class Symbol**	**Example**
Constant	$O(1)$	Comparison of two integers
Logarithmic	$O(log(n))$	Binary Search, Exponentiation
Linear	$O(n)$	Linear Search
Log-Linear	$On(log(n))$	Merge Sort
Quadratic	$O(n^2)$	Integer multiplications
Cubic	$O(n^3)$	Matrix multiplication
Polynomial	$O(n^a)$, $a \in \mathbb{R}$	
Exponential	$O(a^n)$, $a \in \mathbb{R}$	Print all subsets
Factorial	$O(n!)$	Print all permutations

$$n! \gg 2^n \gg n^3 \gg n^2 \gg nlogn \gg n \gg logn \gg 1$$

Growth Rates of Functions



Find F_n: The curse of Exponential time

Fibonacci Sequence

$$0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$$

Find F_n: The curse of Exponential time

Fibonacci Sequence

$$0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$$

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n > 2 \end{cases}$$

Find F_n: The curse of Exponential time

Implementation of the recursive definition of F_n

```
function FIB1(n)
   if n = 0 then
      return 0
   else if n = 1 then
      return 1
   else
      return FIB1(n − 1) + FIB1(n − 2)
```

Find F_n: The curse of Exponential time

Implementation of the recursive definition of F_n

---

function $\text{FIB1}(n)$
   if $n = 0$ **then**
      return 0
   else if $n = 1$ **then**
      return 1
   else
      return $\text{FIB1}(n-1) + \text{FIB1}(n-2)$

- Is it correct?
- How much time it takes to compute F_n?
- Can we do better?

Find F_n: The curse of Exponential time

Let $T(n)$ be the number of ops (comparisons and additions) on input n

Find F_n: The curse of Exponential time

Let $T(n)$ be the number of ops (comparisons and additions) on input n

```
function FIB1(n)
    if n = 0 then
        return 0
    else if n = 1 then
        return 1
    else
        return FIB1(n − 1) + FIB1(n − 2)
```

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2 & \text{if } n = 1 \\ T(n-1) + T(n-2) + 3 & \text{if } n > 2 \end{cases}$$

Find F_n: The curse of Exponential time

Let $T(n)$ be the number of ops (comparisons and additions) on input n

```
function FIB1(n)
   if n = 0 then
      return 0
   else if n = 1 then
      return 1
   else
      return FIB1(n − 1) + FIB1(n − 2)
```

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2 & \text{if } n = 1 \\ T(n-1) + T(n-2) + 3 & \text{if } n > 2 \end{cases}$$

By definition, we have $T(n) > F_n$

Find F_n: The curse of Exponential time

Let $T(n)$ be the number of ops (comparisons and additions) on input n

```
function FIB1(n)
    if n = 0 then
        return 0
    else if n = 1 then
        return 1
    else
        return FIB1(n − 1) + FIB1(n − 2)
```

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2 & \text{if } n = 1 \\ T(n-1) + T(n-2) + 3 & \text{if } n > 2 \end{cases}$$

By definition, we have $T(n) > F_n$

The running time of $\text{FIB1}(n)$ grows as fast as F_n

Find F_n: The curse of Exponential time

Let $T(n)$ be the number of ops (comparisons and additions) on input n

```
function FIB1(n)
   if n = 0 then
      return 0
   else if n = 1 then
      return 1
   else
      return FIB1(n − 1) + FIB1(n − 2)
```

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2 & \text{if } n = 1 \\ T(n-1) + T(n-2) + 3 & \text{if } n > 2 \end{cases}$$

By definition, we have $T(n) > F_n$

The running time of $\text{FIB1}(n)$ grows as fast as F_n

$T(n) \geq 2^{.69n}$ ▷ **exponential** in n (prove by induction)

Find F_n: The curse of Exponential time

$$T(n) \geq 2^{.69n}$$

Find F_n: The curse of Exponential time

$$T(n) \geq 2^{.69n}$$

- For $n = 300$, computing F_{300} takes (much) more than 2^{150} ops
- On a $64\,THz$ computer (64×2^{40} operations per second)
- It needs $2^{104}s > 10^{27}h > 10^{23}$ years

Find F_n: The curse of Exponential time

$$T(n) \geq 2^{.69n}$$

- For $n = 300$, computing F_{300} takes (much) more than 2^{150} ops
- On a 64 *THz* computer (64×2^{40} operations per second)
- It needs $2^{104}s > 10^{27}h > 10^{23}$ years

Another perspective to see growth of exponential time

- Runtime of FIB1(n) is $\geq 2^{0.694n} \approx (1.6)^n$
  - it takes 1.6 times longer to compute F_{n+1} than F_n
- Moore's law $\implies$ computers get roughly 1.6 times faster each year

Find F_n: The curse of Exponential time

$$T(n) \geq 2^{.69n}$$

- For $n = 300$, computing F_{300} takes (much) more than 2^{150} ops
- On a $64\,THz$ computer (64×2^{40} operations per second)
- It needs $2^{104}s > 10^{27}h > 10^{23}$ years

Another perspective to see growth of exponential time

- Runtime of $\text{FIB1}(n)$ is $\geq 2^{0.694n} \approx (1.6)^n$
  - it takes 1.6 times longer to compute F_{n+1} than F_n
- Moore's law $\implies$ computers get roughly 1.6 times faster each year
- If we can compute F_{100} with this year's technology, next year we will manage F_{101}, the year after, F_{102}, ...

  ▷ one more Fibonacci number every year

Such is the curse of exponential time

Find F_n: The curse of Exponential time

$$T(n) \geq 2^{.69n}$$

- For $n = 300$, computing F_{300} takes (much) more than 2^{150} ops
- On a $64\,THz$ computer (64×2^{40} operations per second)
- It needs $2^{104}s > 10^{27}h > 10^{23}$ years

Another perspective to see growth of exponential time

- Runtime of FIB1(n) is $\geq 2^{0.694n} \approx (1.6)^n$
  - it takes 1.6 times longer to compute F_{n+1} than F_n
- Moore's law $\implies$ computers get roughly 1.6 times faster each year
- If we can compute F_{100} with this year's technology, next year we will manage F_{101}, the year after, F_{102}, ...

  ▷ one more Fibonacci number every year

Such is the curse of exponential time

How can we improve it?

Exponential vs Polynomial Growth rates

Sizes of problems that can be solved within 10^{12} operations on today's computer and next years computer with double speed

Complexity	Increase	Problem Size (today)	Problem Size (next year)
n	$n \to 2n$	10^{12}	2×10^{12}
n^2	$n \to \sqrt{2}n$	10^6	1.4×10^6
n^3	$n \to \sqrt[3]{2}n$	10^4	1.25×10^4
$2^{n/10}$	$n \to n + 10$	400	410
2^n	$n \to n + 1$	40	41