Day_1 Data Structure And Algorithms

Topics Covered:-

Fibonacci Numbers

Greatest Common Divisior

Big-O-Notation

Leetcode Problems:-

Problem: 509 https://leetcode.com/problems/fibonacci-number/

Problem: 1979 https://leetcode.com/problems/find-greatest-common-divisor-of-array/description/

Data Structure And Algorithms

Day-1. {1st Part}.

Date 21/1/21
Page

Fibonacci Numbers: $F_n = \begin{cases} 0, & n=0 \\ 1, & n=1 \\ F_{n-1} + F_{n-2} & n>1 \end{cases}$

↳ Rabbit Population:

⇒
```
if n ≤ 1:
    return n
else
    return FibRecurs(n-1) + FibRecurs(n-2)
```

Running Time:

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

Therefore $T(n) \ge F_n$

$T(100) \approx 1.77 \cdot 10^{21}$

Takes 56,000 years at 1GHz

* Why so slow:
This is calculating for ex F(n-3) several no. of times. From scratch.

⇒ (Better Algorithm)

create an array F[0...n]

F[0] ← 0

F[1] ← 1

for i from 2 to n:

    F[i] ← F[i-1] + F[i-2]

return F[n]

Running Time

$T_n = 2n+2$. So $T(100) = 202$

Greatest Common Divisor

o largest no. that divides (a,b)

ex (10,4) → 2 GCD's

Naive Algorithm ~~Naive GCD~~(a, b)

```
best ← 0
for d from 1 to a+b:
    if d/a and d/b:
        best ← d
return best
```

RunTime: approximately (a+b) many times

• very slow for 20 digit numb.

• Euclidean Algorithm.

⇒ (Better Algorithm).

```
IF b=0:
    return a
a' ← the remainder when a
     is divided by b
return EuclidGCD(b,a')
```

* Key Lemma:
Let a' be the remainder when a is divided by b then

gcd(a,b) = gcd(a',b) = gcd(b,a')

proof: $a = a' + bq$ for some q
d divides a and b if and only if it divides a' and b.

Computing Run Time:

→ Hard: . . Depends on fine details of program, computer works

Idea: Basic

All of these issues can multiply runtime by (large) constant

So measure runtime in a way that ignores constant multiples.

→ Consider asymptotic runtime.

Approximate Runtime

input size	n	$n \log n$	n^2	2^n
$n=20$	1 sec	1 sec	1 sec	1 sec
$n=50$	1 sec	1 sec	1 sec	13 day
$n=10^2$	1 sec	1 sec	1 sec	$4 \cdot 10^3$ year
$n=10^6$	1 sec	1 sec	17 min	
$n=10^9$	1 sec	30 sec	30 year	

$$\log n < \sqrt{n} < n < n \log n < n^2 < 2^n$$

Big-O notation:

Definition:

$f(n) = O(g(n))$ if there exist constant N and c so that for all $n \geq N$, $f(n) \leq c \cdot g(n)$

* To classify growth rate.
* Using Big-O loses important information about constant multiples.
* Big-O is only asymptotic

Common rules:

- Multiplicative constants can be omitted:
  $7n^3 = O(n^3)$, $\frac{n^3}{3} = O(n^3)$
- $n^a < n^b$ for $0 < a < b$:
  $n = O(n^2)$ . . $\sqrt{n} = O(n)$
- $n^a < n^b$ $(a > 0, b > 1)$:
  $n^5 = O(\sqrt{2}^n)$, $n^{100} = O(1.1^n)$
- $(\log n)^a < n^b$ $(a, b > 0)$:
  $(\log n)^3 = O(\sqrt{n})$, $n \log n = O(n^2)$
- Smaller terms can be omitted
  $n^2 + n = O(n^2)$, $2^n + n^9 = O(2^n)$

Ex Big-O in Practice

Operation	Runtime
Create an array F[0...n]	O(n)
F[0] ← 0	O(1)
F[1] ← 1	O(1)
for i from 2 to n:	loop O(n) times
F[i] ← F[i-1] + F[i-2]	O(n)
return F[n]	O(1)

Total: $O(n) + O(1) + O(1) + O(n) \cdot O(n) + O(1) = O(n^2)$

Practice Question

* Order the given function by increasing growth rate

→ $F(1) = n^3$ $F(2) = n^{0.3}$ $F(3) = n$ $F(4) = \sqrt{n}$ $F(5) = n^{1.5}$ $F(6) = n^2$

Ans These function are all polynomial function of n
For polynomial function n^a the growth rate increases as the exponent a increases

∴ order: $n^{0.3} < n^{0.5} < n < n^{1.5} < n^2 < n^3$

Q2 $F(1) = 3^n$ $F(2) = n \log_2 n$ $F(3) = \log_4 n$ $F(4) = n$
$F(5) = n^{2.321}$ $F(6) = n^{0.5}$ $F(7) = 4^n$

Ans $F(1) = 3^n$: Exponential growth
$F(2) = n \log_2 n$: slightly faster than linear growth
$F(3) = \log_4 n$: logarithmic growth
$F(4) = n$: linear growth
$F(5) = n^{2.321}$: polynomial growth with exponent 2.321
$F(6) = n^2$: Quadratic growth
$F(7) = n^{0.5}$: growth with the square root of n:
$F(8) = 4^n$: Exponential growth faster than 3^n

order
$\log_4 n < n^{0.5} < n < n \log_2 n < n^2 < n^{2.32} < 3^n < 4^n$