6 Decrease and Conquer

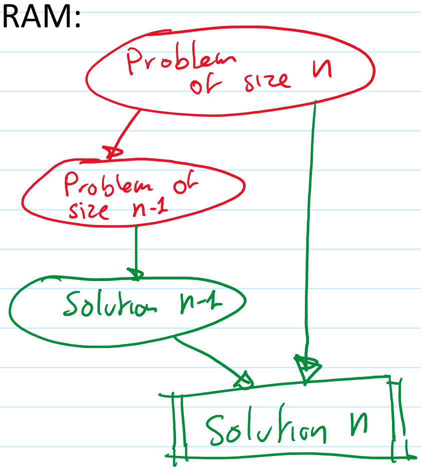
Thursday, March 20, 2025 1:20 PM

Algorithm Design Technique.

- Exploit the relationship between a large problem and a smaller instance of the problem

Variants → Decrease by 1
→ Decrease by constant c
→ Decrease by a variable size.

- DIAGRAM:



- Decrease by 1

E.G. Power

$$a^n = \underbrace{a \cdot a \cdot a \cdot \ldots \cdot a}_{n\text{-times.}}$$

How a^n related to a^m $m < n$?

- $a^n = a^{n-1} \cdot a$
- $a^0 = 1$

```
FUNCTION pow(a, n)
  IF n = 0
     RETURN 1
  ELSE
     RETURN a * pow(a, n-1)
```

← top-down approach.

a^4	John
a^3	emily
a^2	chisha
a^1	josh
a^0	david

What about a "bottom-up" approach.
- Start at a^0 and build up to a^n
- unroll the recursion.

- Start at a^0 and 'build up' to a^n
- unroll the recursion.

```
FUNCTION pow(a, n)
    k ← 0
    r ← 1            // a^0
    WHILE k ≤ n DO
        k ← k+1
        r ← r * a    // r = a^k
    RETURN r  // r = a^k ∧ k = n → r = a^n
```

Analysis:

basic operation: $*$

$M(n) = n \in \Theta(n)$ linear

- Decrease by a constant factor ②

E.G. Power.

How a^n relates a^m $m < n$?

a^n related $a^{n/2}$?

$$\begin{cases} a^n = a^{\lfloor n/2 \rfloor} \cdot a^{\lfloor n/2 \rfloor} & \text{when } n \text{ is even} \\ a^n = a^{\lfloor n/2 \rfloor} \cdot a^{\lfloor n/2 \rfloor} \cdot a & \text{when } n \text{ is odd} \end{cases}$$

$a^0 = 1$

e.g $a^6 = a^3 \cdot a^3$

$a^7 = a^3 \cdot a^3 \cdot a$

```
FUNCTION pow(a, n)
  IF n = 0 THEN
      RETURN 1
  ELSE
      b ← pow(a, ⌊n/2⌋)
      IF n is even THEN
          RETURN b * b
      ELSE
          RETURN b * b * a
```

Analysis:

basic-operation $*$

Using the function call tree:

$$\left.\begin{array}{ll} pow(a, n) & 2 \\ pow(a, \lfloor \frac{n}{2} \rfloor) & 2 \\ pow(a, \lfloor n/4 \rfloor) & 2 \end{array}\right\} (\log_2 n) \cdot 2 = 2 \cdot \log_2 n \in \Theta(\log n)$$

$M(n)$

logarithmic.

$\text{pow}(a, \lfloor n/2 \rfloor)$ 2

$\downarrow$

$\text{pow}(a, \lfloor n/4 \rfloor)$ 2

$\downarrow$

$\text{pow}(a, \lfloor n/8 \rfloor)$ 2

$\downarrow$

$\vdots$ $\vdots$

$\downarrow$

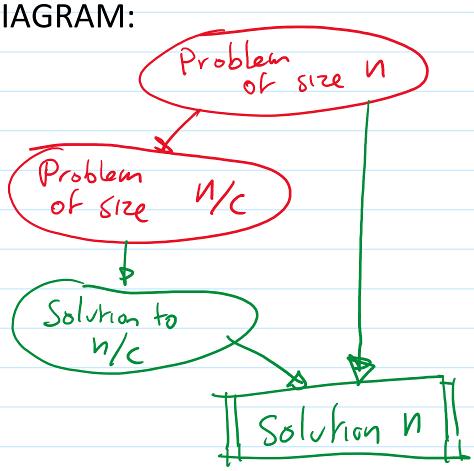
$\text{pow}(a, 0)$ 0

$$(\log_2 n) \cdot 2 = 2 \cdot \log_2 n \in \Theta(\log n)$$ logarithmic.

a^{100} 100 vs 14

a^{1000} 1000 vs 20

- DIAGRAM:



- Decrease by variable size:

e.g. Euclid's Algorithm

$a > b$ • $GCD(a,b) = GCD(b, a-b)$

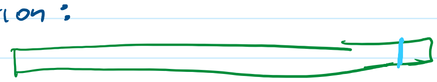
$= GCD(b, a \bmod b)$

• $GCD(a,a) = a$

```
FUNCTION gcd(a, b)      a >= b
    IF a = b
        RETURN a
    ELSE
        RETURN gcd(b, a mod b)
```

- Decrease by 1
  - Sorting

Intuition:



size n

Suppose somebody can solve a smaller problem size n

Suppose somebody can solve a smaller problem
i.e. sort an array of size $n-1$

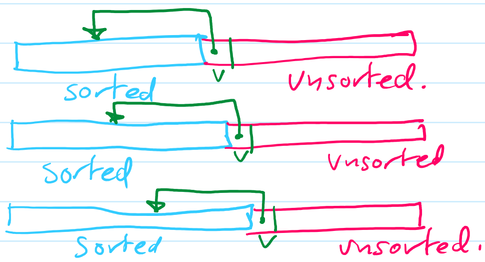
sorted

insert remaining element in correct place.

Simplest Case:

size 1

size 2: either swap or keep.




```
FUNCTION InsertionSort( A[0...n-1] )
  FOR i <- 1 TO n-1 DO        // Note: A[0..i-1] is already sorted.
    v <- A[i]                 // A[i]: element that we are inserting.
    j <- i-1                  // index j searches for a place to insert v.
    WHILE j >= 0 and A[j] > v DO
      A[j+1] <- A[j]
      j <- j-1
    A[j+1] <- v
```

Trace:

6	3	9	2	10	5	12

j i

3 6 9 2 10 5 12

j i

3 6 9 2 10 5 12

j i

2 3 6 9 10 5 12

j i

2 3 6 9 10 5 12

j i

2 3 5 6 9 10 12

j i

2 3 5 6 9 10 12

https://visualgo.net/en

Analysis:

basic operation $<$

Consider j: $1, 2, 3, 4 \ldots\ldots, n-1$

$$C_{worst}(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} \approx \frac{1}{2}n^2 \in \Theta(n^2)$$ Quadratic.

- Worst case scenario = array is sorted in reverse order
- Best case scenario = array is already sorted.

$$C_{best}(n) = \sum_{i=1}^{n-1} 1 = n-1 \in \Theta(n)$$ linear.

—o—o· EOF