Divide-and-Conquer Algorithms

Suppose you wanted to calculate a^n without using your programming languages built in exponential function (where a and n are integers greater than 0).

How could you write a recursive function to do this?

Algorithm 1 int pow(int a, int n) $\therefore n \geq 1$

```
//Base case:
if n == 1 then
    //a^1 is a
    return a;
end if
//Recursive case:
return a*pow(a,n − 1);
```

$\underbrace{a*pow(a,n-1)}$

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

// Not divide & conquer
// cant use master theorem

Analysis of runtime:

$\Rightarrow T(n) = \underbrace{T(n-1)}_{\text{// run time dependent on } (n-1)} + \underbrace{1}_{\text{multiplication } (a)}$

(Base case)
$T(1) = 1$ step

$\Rightarrow T(n) = \boxed{T(n-1)} + 1$ // To find $T(n)$, we need to find $T(n-1), \ldots, T(n-2), \ldots$

$= \boxed{T(n-2) + 1} + 1$ // notice pattern

$(n-2)$ +1 +1
$(n-3)$ +1 +1 +1

$= \boxed{T(n-3) + 1} + 1 + 1$

$\vdots$

$= T(n-i) + i$ // This goes until base case reached

$\vdots$

$= T(1) + \underline{n-1}$ // $n - i = 1$?

$\Rightarrow n = i + 1$

$\Rightarrow i = n - 1$ // This tells us we need to do $(n-1)$ steps to reach base case

$= n$ steps

$= \Theta(n)$

// can we do better?

Can we do better than this? We can use divide & conquer to do better

Observe the following facts: Will we have fewer multiplications using this method?

if n is even: $a^n = a^{n/2+n/2} = a^{n/2}a^{n/2} = (a^{n/2})^2$

if n is odd: $a^n = a^{\lfloor n/2 \rfloor + \lfloor n/2 \rfloor + 1} = a^{\lfloor n/2 \rfloor} a^{\lfloor n/2 \rfloor} a = (a^{\lfloor n/2 \rfloor})^2 a$

Algorithm 2 int pow(int a, int n)

//recursive squaring

//Base case:
if $n == 1$ **then**
  //a^1 is a
  return a;
end if
//Recursive case:
temp=pow(a,$\lfloor n/2 \rfloor$);
temp=temp∗temp; $= (\lfloor n/2 \rfloor)^2$
if n is odd **then**
  temp=temp∗a; $= (\lfloor n/2 \rfloor)^2 \cdot a$
end if
return temp;

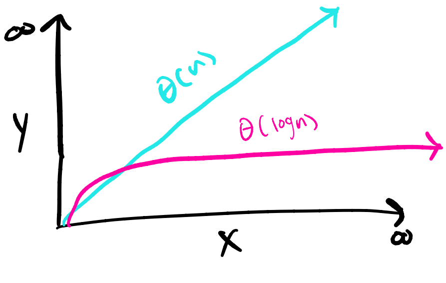
//notice 1 recursive call

Analysis of runtime:

$\Rightarrow T(n) = 1 \cdot T(n/2) + 2$ multiplications (worst case)

// using master theorem, we get case 2

$\Rightarrow 2 \in \Theta(1) \Rightarrow T(n) = \Theta(\log n)$, which is better than $\Theta(n)$



Matrix multiplication ex

AB

$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, $B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$

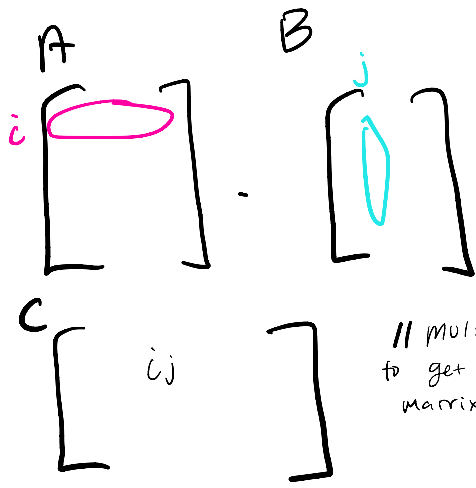
$1(5)+2(7) = 19$
$1(6)+2(8) = 22$
$3(5)+4(7) = 43$
$3(6)+4(8) = 50$

$AB = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$

Matrix multiplication

(1) **input**: $A = [a_{ij}^{\text{row column}}], B = [b_{ij}]$
(2) **output**: $C = [c_{ij}] = A \cdot B$

(where $1 \leq i, j \leq n$)



// multi entire row x column to get single # of new matrix

$$c_{ij} = \sum_{k=1}^{n} a_{ik}^{\text{row column}} \cdot b_{kj}^{\text{row column}}$$

// k walks through columns of A
& k walks through rows of B

Standard algorithm:

```
for (i = 1; i <= n; ++i) {            O(n)
    for (j = 1; j <= n; ++j) {        O(n)
        for (k = 1; k <= n; ++k) {    O(n)
            C[i][j] += a[i][k] · b[k][j];
        }
    }
}
```

Runtime: $O(n^3)$; Naive matrix multi.

Could we perform matrix multiplication faster using a divide and conquer algorithm?

Key idea: $n \times n$ matrix $= 2 \times 2$ matrix of $(n/2) \times (n/2)$ submatrices.

$$\underbrace{\begin{bmatrix} r & s \\ T & U \end{bmatrix}}_{C} = \underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}_{B}$$

// is there a relationship between these quadrants? Yes

$\Rightarrow r = \underbrace{a \cdot e}_{\text{matrix multiplication}} + b \cdot g \qquad T = c \cdot e + d \cdot g$

$\Rightarrow s = a \cdot f + b \cdot h \qquad U = c \cdot f + d \cdot h$

Analysis of runtime:

8 recursive calls $\Rightarrow$ e/a matrix requires 2 calls, and we have 4 matrices $\Rightarrow 2 \cdot 4 = 8$

$$\Rightarrow T(n) = 8\,T(n/2) + n^2$$

$\nearrow$ cost of matrix addition //technically its $(n/2)^2$

// we can now use master Theorem

Strassen's idea

Multiply 2×2 matrices with only 7 recursive multiplications:

Strassen's algorithm

(1) **Divide:** Partition A and B into $(n/2) \times (n/2)$ submatrices. Form P-terms to be multiplied using $+$ and $-$.

(2) **Conquer:** Perform 7 multiplications of $(n/2) \times (n/2)$ submatrices recursively.

(3) **Combine:** Form C by using $+$ and $-$ on the $(n/2) \times (n/2)$ submatrices