Divide & Conquer

⇒ The divide-and-conquer method is a very powerful strategy for designing asymptotically efficient algorithms.

In divide-and-conquer strategy we solve a given problem recursively. If the problem is small enough (base case) we can solve it directly without using recursion. Otherwise for the recursive case we have to perform following three steps ⇒

- Divide ⇒ At first we divide the problem into one or more subproblems that are smaller instances of the main problem.
- Conquer ⇒ Then we have to conquer the subproblems by solving them recursively.
- Combine ⇒ At last we combine the solutions of the subproblems to get the ultimate solution of the original problem.

Recurrences.

⇒ A recurrence is an equation that describes a function in terms of its value on other.

Recurrences go hand in hand with the divide-and-conquer method as they give us a natural way to characterize the running times of recursive algorithms mathematically.

⇒ Following are 3 methods for solving recurrences

i) Substitution method

ii) Recursion-tree method

iii) Master method

- Here we'll use master method as it is the easiest method, It provides bounds for recurrences of the form

$$T(n) = aT(n/b) + f(n)$$

- Master theorem is a formula for solving recurrence relations of the form

$T(n) = aT(n/b) + f(n)$, * $a \geq 1$, $b > 1$ constants

$f(n) \rightarrow$ asymptotically positive function (for a sufficiently large value of n, we've $f(n) > 0$)

where,

n = size of input.

a = no. of sub-problems in the recursion

n/b = size of each subproblems. (assumed to have same size)

$f(n)$ = cost of the work done outside of the recursive call, including the cost of dividing the problem and the cost of merging the solutions.

- This concept is used to compute the time complexity in a simple & quick way. [mainly for d.&c. algo.]

If $a \geq 1$ & $b > 1$ then we assume the general form of

$f(n) = O(n^k \log^p n)$ [$p \in R$, $k \geq 0$]

- from this we'll find 2 values.

(i) a

(ii) b^k.

- Based on these 2 values there are 3 cases.

Now, Therefore, our recurrence relation

$\Rightarrow T(n) = aT(n/b) + O(n^k \log^p n)$

◈ c-1 ⇒ if $a > b^k$, then $T(n) = O(n^{\log_b a})$

◈ c-2 ⇒ if $a = b^k$

ⓐ if $p > -1$, $T(n) = O(n^{\log_b a} \log^{p+1} n)$

ⓑ if $p = -1$, $T(n) = O(n^{\log_b a} \cdot \log\log n)$

ⓒ if $p < -1$, $T(n) = O(n^{\log_b a})$

◈ c-3 if $a < b^k$

ⓐ if $p \geq 0$, $T(n) = O(n^k \log^p n)$ | ⓑ if $p < 0$, $T(n) = O(n^k)$

Matrix Multiplication

⇒ Let, we have 2 square matrices of order $n \times n$

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = C \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

* If the column of 1st matrix (m) = row of the 2nd matrix (n) then we can multiply those 2 and get the resultant matrix. $(n \times m)$.

• Naive Method/approach

```
Matrix_Multiplication (A, B, n)

  for i ← 1 to n
  do
    for j ← 1 to n
    do
      Cij ← 0
      for k ← 1 to n
      do
        Cij ← Cij + (Aik * Bkj)
      end for
    end for
  end for
  return C
```

* The innermost calculation $C_{ij} \leftarrow C_{ij} + (A_{ik} * B_{kj})$ runs in constant time, i.e. in $O(1)$
This statement is enclosed within 3 for loops which are running for n times.
Therefore, the running time complexity

$$T(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} O(1)$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{n} O(n) = \sum_{i=1}^{n} O(n^2)$$

$$= O(n^3).$$

◆ Divide & Conquer method/approach

⇒ Here we consider the smallest problem as 2×2 matrix. Greater than this will be treated as larger problem and we'll divide that into smaller sub-problems of 2×2 matrices.

⇒ We assume the dimentions as power of 2. If the matrices are not of type $2^n \times 2^n$, we'll fill the missing rows and columns with zeros.

Now,

$c_{11} = a_{11} * b_{11} + a_{12} * b_{21}$

$c_{12} = a_{11} * b_{12} + a_{12} * b_{22}$

$c_{21} = a_{21} * b_{11} + a_{22} * b_{21}$

$c_{22} = a_{21} * b_{12} + a_{22} * b_{22}$

$A = [a_{11}]$. $B = [b_{11}]$

$C = [a_{11} * b_{11}]$.

If matrices are of order 1X1. then only 1 multiplication is required.

If the matrices are of order 2X2 then we can calculate their multiplication by simply using these 4 formulas.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

A_{11} A_{12} A_{21} A_{22} B_{11} B_{12} B_{21} B_{22}

⇒ Algorithm MM(A, B, n).

```
if (n <= 2)
do,
    c11 = a11 * b11 + a12 * b21
    c12 = a11 * b12 + a12 * b22
    c21 = a21 * b11 + a22 * b21
    c22 = a21 * b12 + a22 * b22.
else.
do,
    mid → n/2.
    MM(A11, B11, n/2) + MM(A12, B21, n/2).
    MM(A11, B12, n/2) + MM(A12, B22, n/2).
    MM(A21, B11, n/2) + MM(A22, B21, n/2)
    MM(A21, B12, n/2) + MM(A22, B22, n/2).
```

- Recurrence Relation

⇒ $T(n) = \begin{cases} 1 & n \leq 2 \\ 8T(n/2) + n^2 & n > 2 \end{cases}$ By using master theorem.

Here $a = 8$, $b = 2$, $k = 2$, $p = 0$.

$a > b^k$. $[b^k = 4]$

$T(n) = O(n^{\log_b a}) = O(n^{\log_2 8}) = O(n^3)$.

Strassen's algorithm.

⇒ Divide matrix A and B in 4 sub-matrices of size $n/2 \times n/2$.

⇒ Calculate the 7 matrix multiplications recursively.

$$P = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$Q = B_{11}(A_{21} + A_{22})$$

$$R = A_{11}(B_{12} - B_{22})$$

$$S = A_{22}(B_{21} - B_{11})$$

$$T = B_{22}(A_{11} + A_{12})$$

$$U = (B_{11} + B_{12})(A_{21} - A_{11})$$

$$V = (B_{21} + B_{22})(B_{12} - A_{22})$$

⇒ Compute the sub-matrices of C.

$$C_{11} = P + S - T + V$$

$$C_{12} = R + T$$

$$C_{21} = Q + S$$

$$C_{22} = P + R - Q + U$$

⇒ Combine these sub-matrices into our new matrix C.

• Strassen's Matrix Multiplication

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

* To compute multiplication of 2 square matrices Strassen compute 7 multiplication and 18 addition & subtraction, instead of doing 8 multiplication to reduce time complexity.

• Formulas

$$P = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$Q = B_{11}(A_{21} + A_{22})$$
$$R = A_{11}(B_{12} - B_{22})$$
$$S = A_{22}(B_{21} - B_{11})$$
$$T = B_{22}(A_{11} + A_{12})$$
$$U = (B_{11} + B_{12})(A_{21} - A_{11})$$
$$V = (B_{21} + B_{22})(A_{12} - A_{22})$$

$$C_{11} = P + S - T + V$$
$$C_{12} = R + T$$
$$C_{21} = Q + S$$
$$C_{22} = P + R - Q + U$$

Example →

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

$A_{11} = 1$, $A_{12} = 3$, $A_{21} = 7$, $A_{22} = 5$

$B_{11} = 6$, $B_{12} = 7$, $B_{21} = 3$, $B_{22} = 8$.

$P = (A_{11} + A_{22})(B_{11} + B_{22}) = (1+5)(6+8) = 6 \times 14 = 84.$

$Q = B_{11}(A_{21} + A_{22}) = 6(7+5) = 6 \times 12 = 72$

$R = A_{11}(B_{12} - B_{22}) = 1(7-8) = 1 \times -1 = -1.$

$S = A_{22}(B_{21} - B_{11}) = 5(3-6) = 5 \times -3 = -15$

$T = B_{22}(A_{11} + A_{12}) = 8(1+3) = 8 \times 4 = 32.$

$U = (B_{11} + B_{12})(A_{21} - A_{11}) = (6+7)(7-1) = 13 \times 6 = 78.$

$V = (B_{21} + B_{22})(A_{12} - A_{22}) = (3+8)(3-5) = 11 \times -2 = -22$

$C_{11} = P + S - T + V$

$= 84 + (-15) - 32 + (-22)$

$= 84 - (15 + 32 + 22)$

$= 84 - 69.$

$= 15.$

$C_{12} = R + T = -1 + 32$

$= 31.$

$C_{21} = Q + S = 72 + (-15)$

$= 72 - 15$

$= 57$

$C_{22} = P + R - Q + U$

$= 84 + (-1) - 72 + 78.$

$= 84 + 78 - (1 + 72)$

$= 162 - 73.$

$= 89.$

Therefore, the calculated matrix $C = \begin{bmatrix} 15 & 31 \\ 57 & 89 \end{bmatrix}$

Recurrence Relation for Strassen's approach

$$T(n) = \begin{cases} 1 & n \leq 2 \\ 7T(n/2) + n^2 & n > 2 \end{cases}$$

Here, $a = 7$, $a > b^k$ $[b^k = 4]$

$b = 2$

$k = 2$

$p = 0.$

By applying master theorem.

$$T(n) = O(n^{\log_b^a}) = O(n^{\log_2^7}) = O(n^{2.81})$$

- It helps us to improve the time complexity a little bit.

- How do we multiply two integers x, y ?

→ From partial products - multiply each digit of y separately by x.

→ Add up all partial products.

→ Works the same in any base - e.g. binary

→ To multiply two n-bit numbers
  → n partial products
  → Adding each partial product to cumulative sum is O(n)
  → Overall $O(n^2)$

```
  12
 x13
----
  36
+12
----
 156
```

```
  1100
 x1101
------
  1100
 0000
1100
1100
------
10011100
```

Algorithm of integer multiplication using divide and conquer

Multiply_D.C. (x, y, n).

if $n < 2$

return $x \cdot y$.

else.

$m = n/2$

$(x_1, x_0) = (x/2^m, x \bmod 2^m)$

$(y_1, y_0) = (y/2^m, y \bmod 2^m)$

$x_1 y_1 \leftarrow$ Multiply_D.C. (x_1, y_1, m)

$x_1 y_0 \leftarrow$ Multiply_D.C. (x_1, y_0, m)

$x_0 y_1 \leftarrow$ Multiply_D.C. (x_0, y_1, m)

$x_0 y_0 \leftarrow$ Multiply_D.C. (x_0, y_0, m)

return $(x_1 y_1) \cdot 2^n + (x_1 y_0 + x_0 y_1) 2^{n/2} + x_0 y_0$.

If there is a large number. Then we apply divide & Conquer method by which we can break the large problem into subproblems.

Let, two numbers x and Y

→ Rewrite XY as

$(X_1 \cdot 2^{n/2} + X_0)(Y_1 \cdot 2^{n/2} + Y_0)$

→ Regroup as,

$X_1 Y_1 \cdot 2^n + (X_1 Y_0 + X_0 Y_1) \cdot 2^{n/2} + X_0 Y_0$

So, from this the recurrence relation is -

$$T(n) = \begin{cases} 1 & n < 2 \\ 4T(n/2) + n & n \geq 2 \end{cases}$$

So, By the Master theorem -

$$T(n) = 4T(n/2) + n$$

$a = 4$

$b = 2$

$k = 1$

$p = 0$

$a > b^k \quad [b^k = 2]$

$T(n) = O(n^{\log_b a})$

$= O(n^{\log_2 k})$

$= O(n^2)$

By applying Divide & Conquer method also we achieve the complexity $O(n^2)$

So Divide & Conquer has not helped! we can say that, in both of the method naive and Divide & Conquer, the time complexity remains same but in the case of large integer values we can multiply it by dividing it in subproblems So, for that kind of problems we can apply Divide & Conquer method.

• Multiplication using Divide & Conquer approach

$x = 7768. \rightarrow x_1 = 77 \quad x_0 = 68.$

$y = 9276. \rightarrow y_1 = 92 \quad y_0 = 76.$

$\Rightarrow x = (77 \times 10^2) + 68.$

$y = (92 \times 10^2) + 76.$

Now $\rightarrow x \times y$

$= \{(77 \times 10^2) + 68\}\{(92 \times 10^2) + 76\}$

$= \underbrace{(77 \times 92)}_{1} 10^4 + \{\underbrace{(77 \times 76)}_{2} + \underbrace{(68 \times 92)}_{3}\} 10^2 + \underbrace{(68 \times 76)}_{4}.$

$= 7084 \times 10^4 + \{5852 + 6256\} 10^2 + 5168.$

$= 70840000 + 1210800 + 5168.$

$= 72055968.$

7768
X 9276
46608
54376X
15536X
69912X
72055968

using naive approach

• $x = 1111. \rightarrow x_1 = 11, \; x_0 = 11$

$y = 1001. \rightarrow y_1 = 10, \; y_0 = 01.$

$\Rightarrow x = (11 \times 2^2) + 11.$

$\Rightarrow y = (10 \times 2^2) + 01.$

Here, 2^x means 2^x no. of bit Shifting

Now $x \times y$

$= \{(11 \times 2^2) + 11\}\{(10 \times 2^2) + 01\}$

$= (11 \times 10) 2^4 + \{(11 \times 01) + (11 \times 10)\} 2^2 + (11 \times 01)$

$= 110 \times 2^4 + \{011 + 110\} 2^2 + 011.$

$= 1100000 + 100100 + 011$

$= 10000111 \Rightarrow$ using Divide & Conquer approach

1111
X 1001
1111
0000X
0000X
1111X
10000111

using naive approach

rough: 11 × 10 = 00 + 11X + 110 ; 11 × 01 = 11 + 00X = 011

1100000
100100
+ 011
10000111

011 + 110 = 1001

★ Karatsuba's Algorithm for fast multiplication using Divide and Conquer algorithm.

The basic principle of karatsuba's algorithm is divide and conquer, using a formula that allows one to compute the product of two large numbers x and y using three multiplications of smaller numbers, each with about half as many digits as x or y, plus some additions and digit shifts.

Karatsuba observed that xy can be computed in only ~~three~~ three multiplications, at the cost of a few extra additions.

The algorithm is as follows:-

```
Fast_Multiply(x, y, n)
if n=1
    return x.y        } → best case
else
    m = n/2           → otherwise split
(x1, x0) = (x/2^m, x mod 2^m)
(y1, y0) = (y/2^m, y mod 2^m)
(a, b) = (x1 - x0, y1 - y0)
p = Fast_Multiply(x1, y1, m)
q = Fast_Multiply(x0, y0, m)
r = Fast_Multiply(a, b, m)
return p.2^n + (p + q - r).2^(n/2) + q
```

Hence the formula can be written as

$$xy = \underbrace{x_1y_1}_{p} \cdot 2^n + \left(\underbrace{x_1y_1}_{p} + \underbrace{x_0y_0}_{q} - \underbrace{(x_1 - x_0)(y_1 - y_0)}_{r}\right) \cdot 2^{n/2} + \underbrace{x_0y_0}_{q}$$

With the above trick, the recurrence becomes

$$T(n) = 3T(n/2) + O(n)$$

To handle the different length case, we append 0's in the beginning. To handle odd length, we put floor(n/2) bits in left half and ceil(n/2) bits in right half.

The recurrence can be solved using master theorem as follows:

$$T(n) = aT(n/b) + O(n^k \log^p n)$$

where

$a \geq 1$, $b > 1$, $k \geq 0$ & p is a real number.

So here case 1 occurs i.e

if $a > b^k$, then $T(n) = O(n^{\log_b a})$

as $(3 > 2^1)$

So in our recurrence,

$$T(n) = 3T(n/2) + n$$

Here, $a = 3$

$b = 2$

$k = 1$

$p = 0$

$$T(n) = O(n^{\log_2 3})$$

$$T(n) \simeq O(n^{1.59})$$

Hence the above algorithm reduces the time complexity of integer multiplication from $O(n^2)$ to $O(n^{1.59})$.

• Using Karatsuba approach.

1) $x = 7768 \rightarrow x_1 = 77 \quad x_0 = 68$

$y = 9276. \rightarrow y_1 = 92 \quad y_0 = 76.$

$\rightarrow x = (77 \times 10^2) + 68.$

$y = (92 \times 10^2) + 76.$

Now $x \times y$

$= \{(77 \times 10^2) + 68\}\{(92 \times 10^2) + 76\}.$

$= (77 \times 92)10^4 + [(77 \times 92) + (68 \times 76) - \{(77-68)(92-76)\}]10^2 + (68 \times 76)$

$= 7084 \times 10^4 + [7084 + 5168 - \{9 \times 16\}]10^2 + 5168.$

$= 70840000 + [12252 - 144]10^2 + 5168$

$= 70840000 + 1210800 + 5168.$

$= 72055968$

---

2) $x = 1111 \Rightarrow x_1 = 11, \; x_0 = 11$

$y = 1001. \Rightarrow y_1 = 10, \; y_0 = 01.$

$x = (11 \times 2^2) + 11.$

$y = (10 \times 2^2) + 01.$

Now $x \times y.$

$= \{(11 \times 2^2) + 11\}\{(10 \times 2^2) + 01\}$

$= (11 \times 10)2^4 + [(11 \times 10) + (11 \times 01) - \{(11-11)(10-01)\}]2^2 + (11 \times 01)$

$= 110 \times 2^4 + [110 + 011 - \{00 \times 01\}]2^2 + 011.$

$= 1100000 + [1001 - 0]2^2 + 011$

$= 1100000 + 100100 + 011.$

$= 10000111.$

110
+011
1001
1100000.
100100
+ 011
10000111