1000!=1000*999*998........1

Multiplication of Two n-digit numbers

Z=x*y; 0(1)   is it?

int x[ ], y[], z[];

X[]=2345678912345678

y[]=2145678912245676

Z[]=mul(x*y)

School level algorithm for multplication

```
            2  3  4  5

            3  4  5 2
          ---------------
           ................. n muls + n-1 addition
        ...............x     "
     ------------------------
```

$O(n^2)$

Can we do it better?

How?

Yes, using divide and conquer approach

$O(n^{1.59})$

Multiplication of Two n-digit numbers

x: n-bit number

y: n-bit number

x= | x_L | x_R | $=2^{n/2}x_L + x_R$

y= | y_L | y_R | $=2^{n/2}y_L + y_R$

x=10110110, x_L=1011, x_R=0110

x=(1011) * 2^4 + 0110

$$xy=(2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R)$$

$$=2^n x_L y_L + 2^{n\backslash 2}(x_L y_R + x_R y_L) + x_R y_R$$

4 products, each one is product of n/2 bit numbers

The original problem is subdivided into four sub-problems and the some problems are of similar type and size of each sub-problem is half of the original problem

T(n): running time of multiplying two n-bit numbers

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

running time for combining the solutions of sub probs

where T(n/2)= running time for computing product of two n/2 bit numbers

Recurrence relations

How to solve this recurrence equation?

(1) Substitution method

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

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

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

$T(n)= 4[4T(n/4)+O(n/2)] +O(n)$

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

$=4^2T(n/2^2)+(2^2 -1) O(n)= 4^2[4T(n/8) +O(n/4)]+3O(n)$

$=4^3 T(n/2^3) +4^2 O(n/4) + 3O(n)$

$=4^3 T(n/2^3)+ (2^3 -1) O(n)$

....

....

$=4^k T(n/2^k)+ (2^k -1) O(n)$

$=4^{\log_2 n} T(1)+ (2^{\log_2 n} -1) O(n)$

$=4^{\log_2 n} + (2^{\log_2 n} -1) O(n)$ [T(1)=1]

$=n^{\log_2 4}+(n-1)O(n)$

$=n^2 + O(n^2)- O(n)$

$=O(n^2)$

Say at step k, n reduces to 1

$T(n/2^k)=T(1)$

$n/2^k=1$

$2^k=n$

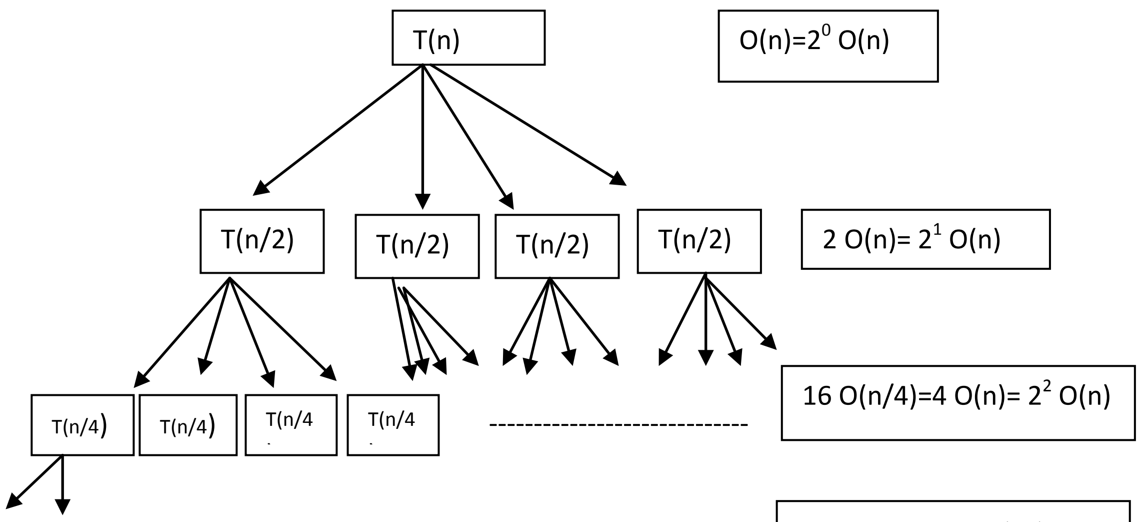
$k=\log_2 n$

No improvement over school level algorithm

How can

(2) Recursion Tree

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



At level x, ... $\boxed{2^x \times O(n)}$

Say, at level x, prob size reduces to 1, that is, $n/2^x = 1$

x=$log_2 n$ (depth of the tree)

At level 1, number of sub problems=4

At level 2, number of sub-problems=4^2

At level x, number of sub-problems of size 1 =4^x

Time spent at level x =$4^x. T(1)$

=$4^x .1$ (assume T[1]=1)

=$4^x . O(\frac{n}{2^x})$ [since 1=$\frac{n}{2^x}$]

=$\left(\frac{4}{2}\right)^x \times O(n) = 2^x \times O(n)$

Say, at level x, size of the sub problem reduces to 1

$$T(n) = \sum_{i=0}^{x} 2^i\, O(n)$$

$$T(n) = \sum_{i=0}^{\log_2 n} 2^i\, O(n)$$

$=O(2^{\log_2 n}\, O(n))$ [since the last term is the largest]

$= O(n.\, O(n))=O(n^2)$

Can we have an algorithm which is better than this?

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

Here dividing the problem into 4 sub-problems

If we want to do better, we need to concentrate on how to reduce number of sub-problems?

1777-1855

C. F gauss

(a + bi) (c+di)=ac-bd +(bc +ad)i (4 different products leading to 4 sub-problems)

bc +ad= (a+b)(c+d)-ac-bd

(a + bi) (c+di)=ac-bd +{(a+b)(c+d)-ac-bd }i (3 products leading to 3 sub-problems)

1 product gain.

How this idea can be used in designing an efficient algorithm for multiplication!!

x: n-bit number

y: n-bit number

x= | x_L | x_R | $=2^{n/2}x_L + x_R$

y= | y_L | y_R | $=2^{n/2}y_L + y_R$

x=10110110, x_L=1011, x_R=0110

x=(1011) * 2^4 + 0110

xy=$(2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R)$

$=2^n x_L y_L + 2^{n\backslash 2}(x_L y_R + x_R y_L) + x_R\, y_R$

4 products, each one is product of n/2 bit numbers

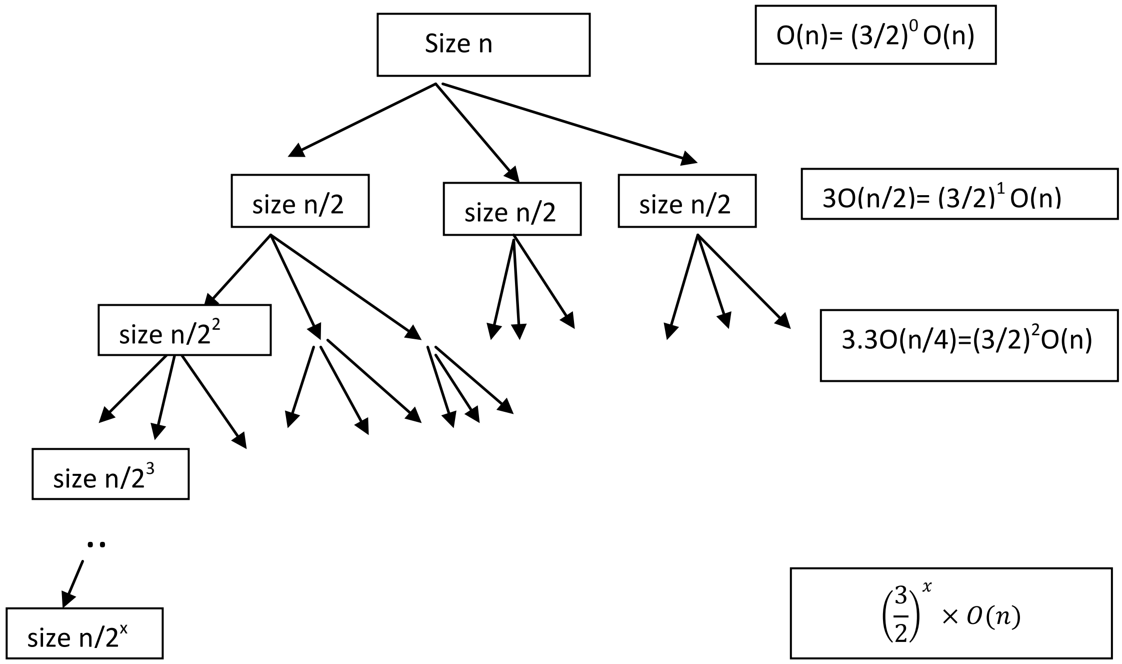
$x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R =$

P3-P1-P2

Function multiply(x,y) T(n)

Input: n-bit positive integers x and y
output: product
if(n==1): return xy
x_L,x_R=leftmost ceil(n/2), rightmost(floor(n/2))
y_L,y_R=leftmost ceil(n/2), rightmost(floor(n/2))
P1= multiply(x_L, y_L) T(n/2)
P2= multiply(x_R, y_R) T(n/2)
P3= multiply($x_L + x_R$, $y_L + y_R$) T(n/2)
return P1 *2^n + (P3-P1-P2)*$2^{n/2}$ +P2 O(n)

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



Say, at level x, prob size reduces to 1, that is, $n/2^x = 1$

$x = log_2 n$

At level 1, number of sub problems=3

At level 2, number of sub-problems=3^2

At level x, number of sub-problems of size 1 =3^x

Time spent at level x=$3^x .\, T(1)$

$=3^x .1$ (assume T[1]=1)

$=3^x . O(\frac{n}{2^x})$ [since $1=\frac{n}{2^x}$]

$$=\left(\frac{3}{2}\right)^x \times O(n)$$

$$T(n) = \sum_{i=0}^{log_2 n} [\left(\frac{3}{2}\right)^i \times O(n)]$$

$=O(\left(\frac{3}{2}\right)^{log_2 n} \times O(n))$ [since last term of

T(n) is the largest]

$=O(\frac{3^{log\,2n}}{2^{log\,2n}}$ x O(n))

$= O(3^{log_2 n}) = O(n^{log_2 3}) = O(n^{1.59})$

Mergesort Algorithm