c

Master Method.

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

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

Case 1: if $a > b^k$, then $T(n) = \theta(n^{\log_b^a})$

Case 2: if $a = b^k$

a) if $P > -1$ then $T(n) = \theta(n^{\log_b^a} \log^{p+1} n)$

b) if $P = -1$ then $T(n) = \theta(n^{\log_b^a} \log\log n)$

c) if $P < -1$ then $T(n) = \theta(n^{\log_b^a})$

Case 3: if $a < b^k$

a) if $P \geq 0$, then $T(n) = \theta(n^k \log^p n)$

b) if $P < 0$, then $T(n) = O(n^k)$.

---

① $T(n) = 3T(n/2) + n^2$

here $a = 3$, $b = 2$, $K = 2$, $P = 0$.

$a \quad b^k$

$3 \quad 2^2$

$3 < 4$

$a < b^k \rightarrow$ Case 3.

$T(n) = \theta(n^k \log^p n) = \theta(n^2 \log^0 n)$

$= \theta(n^2)$.

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

$a = 4,\ b = 2,\ k = 2,\ p = 0$

$a = 4,\ b^k = 2^2 = 4$

$a = b \rightarrow$ case 2. $p > -1$ case 2a.

$$T(n) = \theta(n^{\log_b a} \log^{p+1} n).$$

$$= \theta(n^{\log_2 4} \log^{0+1} n) = \theta(n^{\log_2 2^2} \log n)$$

$$= \theta(n^2 \log n).$$

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

$a = 1,\ b = 2,\ k = 2,\ p = 0$

$a = 1,\ b^k = 2^2 = 4$

$a < b^k \rightarrow$ case 3 "" $p = 0 \rightarrow$ case 3.a.

$$T(n) = \theta(n^k \log^p n)$$

$$= \theta(n^2 \log^0 n) = \theta(n^2).$$

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

→ here 'a' have to be a constant for masters theorem, Since 'a' is not constant therefore we cannot apply masters theorem.

(5) $T(n) = 16T(n/4) + n.$

$a = 16,\ b = 4,\ k = 1,\ p = 0.$

$a = 16,\ b^k = 4.$

$a > b^k$. case 1, $p = 0$

$$T(n) = \theta(n^{\log_b a}) = \theta(n^{\log_4 16}) = \theta(n^{\log_4 4^2})$$

$$= \theta(n^2)$$

6. $T(n) = 2T(n/2) + n\lg n$

$a=2,\ b=2,\ K=1,\ p=1.$

$a=2,\ b^k=2.$

$a=b$, $\underline{\text{Case 2}}$, $p=1 \Rightarrow$ case 2(a).

$T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$

$= \Theta(n^{\log_2 2} \log^{1+1} n) = \Theta(n^1 \log^2 n).$

$= \Theta(n \log^2 n).$

7. $T(n) = 2T(n/2) + n/\log n.$

$= 2T(n/2) + n \log^{-1} n$

$a=2,\ b=2,\ K=1,\ p=-1.$

$a=2,\ b^K=2.$

$a=b$. $\underline{\text{Case 2}} \Rightarrow$ $p=-1 \Rightarrow$ case 2(b).

$T(n) = \Theta(n^{\log_a b} \log\log n).$

$= \Theta(n^{\log_2 2} \log\log n)$

$= \Theta(n \log\log n).$

8. $T(n) = 2T(n/4) + n^{0.51}$

$a=2,\ b=4,\ K=0.51,\ p=0.$

$a=2,\ b^k = 4^{0.51} \simeq 2.028.$

$a < b^k$ $\underline{\text{Case 3}} \Rightarrow p=0 \Rightarrow \underline{\text{Case 3(a)}}.$

$T(n) = \Theta(n^k \log^p n) = \Theta(n^{0.51} \log^0 n)$

$= \Theta(n^{0.51}).$

9. $T(n) = 0.5\,T(n/2) + 1/n$

$a = 0.5$

But according to the formula $a \geq 1$.

Therefore we can not apply Masters theorem here as $a < 1$.

10. $T(n) = 6\,T(n/3) + n^2 \log n$

$a = 6$, $b = 3$, $k = 2$ & $p = 1$.

$a = 6$, $b^k = 3^2 = 9$.

$a < b^k$. Case 3 $\Rightarrow p = 1 \Rightarrow$ Case 3(a)

$$T(n) = \theta(n^k \log^p n)$$

$$= \theta(n^2 \log^1 n) = \theta(n^2 \log n).$$

11. $T(n) = 4T(n/2) + \log n$

$a = 4$, $b = 2$, $k = 0$, $p = 1$

$a = 4$, $b^k = 2^0 = 1$.

$a > b^k$ Case 1

$$T(n) = \theta(n^{\log_b a}) = \theta(n^{\log_2 4}) = \theta(n^{\log_2 2^2})$$

$$= \theta(n^2)$$

12. $T(n) = 64\,T(n/8) - n^2 \log n$
13. $T(n) = 7T(n/3) + n^2$
14. $T(n) = 4T(n/2) + \log n$
15. $T(n) = \sqrt{2}\,T(n/2) + \log n$
16. $T(n) = 2T(n/2) + \sqrt{n}$
17. $T(n) = 3T(n/2) + n$
18. $T(n) = 3T(n/3) + \sqrt{n}$
19. $T(n) = 4T(n/2) + Cn$
20. $T(n) = 3T(n/4) + n \log n$.

Integer Multiplication

The problem of multiplying "big integers" i.e. integers represented by a large number of bits that cannot be handled directly by the arithmatic unit of a single processor.

⇒ Multiplying big integers has applications to data security, where big integers are used in encryption scheme.

for given two big integers I & J. represented with 'n' bits each we can compute $I+J$ & $I-J$ in $O(n)$ time.

However computing $I*J$ using the common grade high school algorithm requires however $O(n^2)$ time.

The goal of this lecture is to come up with a mechanism to reduce the complexity for $I*J$. by using divide & conquer technique.

Let us assume that n is a power of 2.
we can therefore divide the bit representation of I & J in half. with one half representing the higher order bits & the other half representing lower-order bits. In particular If we split I into I_h & I_l & J into J_h & J_l, then

$$I = I_h 2^{n/2} + I_l \quad \text{------} \quad (i)$$

$$J = J_h 2^{n/2} + J_l \quad \text{------} \quad (ii)$$

Here observe that Multiplying a binary number I by a power of 2, 2^k is trivial. It simply involves shifting left (i.e. in the higher order direction). the number I by K bit position. Multiplying an integer by 2^k takes $O(k)$ time.

Using (I) & (II) we can write

$$I * J = (I_h 2^{n/2} + I_l) * (J_h 2^{n/2} + J_l).$$

$$= I_h J_h 2^n + I_l J_h 2^{n/2} + I_h J_l 2^{n/2} + I_l J_l \quad \text{---- (III)}$$

Thus we can compute $I*J$ by applying divide & conquer algorithm that divides the bit representation of I & J in half. The divide & conquer algorithm has a running time that can be characterize by the following recurrence (for $n \geq 2$).

$$T(n) = 4T(n/2) + Cn \quad \text{------ (iv)}$$

If we apply master method then we can show that

$T(n)$ is $\Theta(n^2)$.

Unfortunately this is not better than the grade-school Algorithm.

However, The Master method gives us some insight into how we might improve this algorithm. If we can reduce the no. of recursive call then we can improve the complexity. Consider the product.

$$(I_h - I_l)(J_l - J_h) = I_h J_l - I_l J_l - I_h J_h + I_l J_h \quad \text{...(v)}$$

This strange product in equation v has an interesting property. when expanded out it contains two product that we want to compute ($I_h J_l$ & $I_l J_h$) & two product can be computed recursively. ($I_h J_h$ & $I_l J_l$). Thus we can compute $I*J$ as follows:

$$I * J = I_h J_h 2^n + [(I_h - I_l)(J_l - J_h) + I_h J_h + I_l J_l] 2^{n/2} + I_l J_l$$

$$\text{..... (VI)}$$

~~Thus~~

This computation requires the recursive computation of three product of $n/2$ bits each, plus $O(n)$ additional work.

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

or, $T(n) = 3T(n/2) + cn$ . . . . (vii).

for constant $c > 0$.

Using Master's theorem

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

$= O(n^{1.585}) = O$