CSE221 ALGORITHMS

Topic: Divide & Conquer

Prepared by:
Saad Bin Sohan

BRAC University

Email: sohan.academics@gmail.com

GitHub: https://github.com/saad-bin-sohan

Divide & Conquer Approach

Merge sort

Kartasuba Multiplication

Max sum subarray

Kartasuba Multiplication

the usual convention of multiplication

$$
\begin{array}{r}
1234 \\
\times\ 5678 \\
\hline
9872 \\
8638\underbrace{0}_{} \\
7404\underbrace{00}_{} \\
617\underbrace{000}_{} \\
\hline
7006652
\end{array}
$$

number length = 4 হলে,

$4 \rightarrow \underbrace{4*4}_{\text{number of multiplication}} + \underbrace{\text{padding}}_{\text{zero's add করা}} + \text{addition}$

number of multiplication (এক নাম্বারের সব digit এর সাথে অন্য নাম্বারের সব digit গুণ)

number এর length n হলে,

$n \rightarrow n*n + \underbrace{\text{padding} + \text{addition}}_{\text{linear } (n)}$

$\Rightarrow n^2$

so conventional method এর time complexity $O(n^2)$.

can we find approach with lesser complexity?

Example

let two numbers,

$X = 1234 \quad Y = 5678$

$$X = \overbrace{1\ 2}^{a=12} \mid \overbrace{3\ 4}^{b=34}$$

$$Y = \underbrace{5\ 6}_{c=56} \mid \underbrace{7\ 8}_{d=78}$$

number গুলোকে দুই ভাগে ভাগ কর)

step-1 $a * c = 12 * 56 = 672$

step-2 $b * d = 34 * 78 = 26252$

step-3 $(a+b) * (c+d)$

$= (12+34) * (56+78)$

$= 6164$

step-4

3 - 1 - 2

⇒ result from step 3 − result from step 1 − result from step 2

$$P = (a+b)(c+d) - ac - bd$$

$$(a+b)(c+d) - ac - bd$$

$$= ac + ad + bc + bd - ac - bd$$

$$= ad + bc$$

direct ad+bc use
না করে এত বড়
$(a+b)(c+d) - ac - bd$ কেন?

step-5

ac... (padd n 0's)

bd

p (padd n/2 times 0's)

=>

```
6720000
   2652
 284000
-------
7006652
```

same outcome we got from conventional multiplication method.

how possible from this — ?

$\boxed{\text{Explanation}}$

$$X = \overbrace{12}^{a} \mid \overbrace{34}^{b}$$

$$1234 = 12 \times 100 + 34$$

$$= 12 \times 10^{2} + 34 \qquad 2 \to n/2$$

$$X = a * 10^{n/2} + b$$

n

$$Y = c * 10^{n/2} + d$$

$$XY = (a * 10^{n/2} + b) * (c * 10^{n/2} + d)$$

$T(n)$

$$= ac * 10^{n} + ad * 10^{n/2} + bc * 10^{n/2} + bd$$

$$= \underbrace{ac}_{T(n/2)} * 10^{n} + (\underbrace{ad}_{T(n/2)} + \underbrace{bc}_{T(n/2)}) * 10^{n/2} + \underbrace{bd}_{T(n/2)}$$

$n/2$

length এর দুইটা
নাম্বার গুলোর
জন্য time
complexity

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

comparing with Master's theorem,

$T(n) = a\,T(n/b) + c * n^k$

$a = 4, \quad b = 2, \quad c = 1, \quad k = 1$

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

so, $b^k < a$

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

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

$= O(n^2)$

(ad+bc) বাড়াতে কমবে time

complexity দাঁড়ায় $O(n^2)$

এখন, $(ad+bc)$ এর জন্যে,

$P = (a+b)*(c+d) - ac - bd$ use করলে (both share same value)

$$XY = ac * 10^{n} + \quad * 10^{n/2} + bd$$

$$\{(a+b)*(c+d) - ac - bd\}$$

$T(n/2)$ $\quad$ $T(n/2)$ $\quad$ $T(n/2)$

4টা $(n/2)$ এর জন্যে we get 3টা

(n/2). lets see the difference it makes.

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

using masters theorem,

$a = 3, b = 2, c = 1, k = 1$

$b^k = 2^1 = 2$ $\boxed{b^k < a}$

so,

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

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

$= O\left(n^{1.59}\right)$

(ad + bc) use করলে $O(n^2)$

p use করলে $O(n^{1.59})$. the difference is invisible when multiplying two small numbers. but the visible difference in execution time when multiplying two very large numbers (e.g. numbers with length of millions) is in hours when we use p instead of (ad+bc).

N.B: this multiplication methods

efficiency does not apply today.

But in previous times, it was a revolution.

extra info:

---

0 add করো

length odd থাকলে 123 => 01 | 23

456 => 04 | 56

length different হলে,

1234 => 12 | 34

567 => 05 | 67

Max Sum Subarray

Max → max value of the sum

Sum → Summation of elements in subarray

Subarray → any sub part taken from the array. like if we are given

-2	6	-1	2

we can make subarrays like $\boxed{-2}$, $\boxed{6 \mid -1 \mid 2}$, $\boxed{6}$, $\boxed{-1 \mid 2}$. but you cannot skip elements. the subarray must have consecutive elements as they were in parent array.

like we cannot take | 6 | 2 |

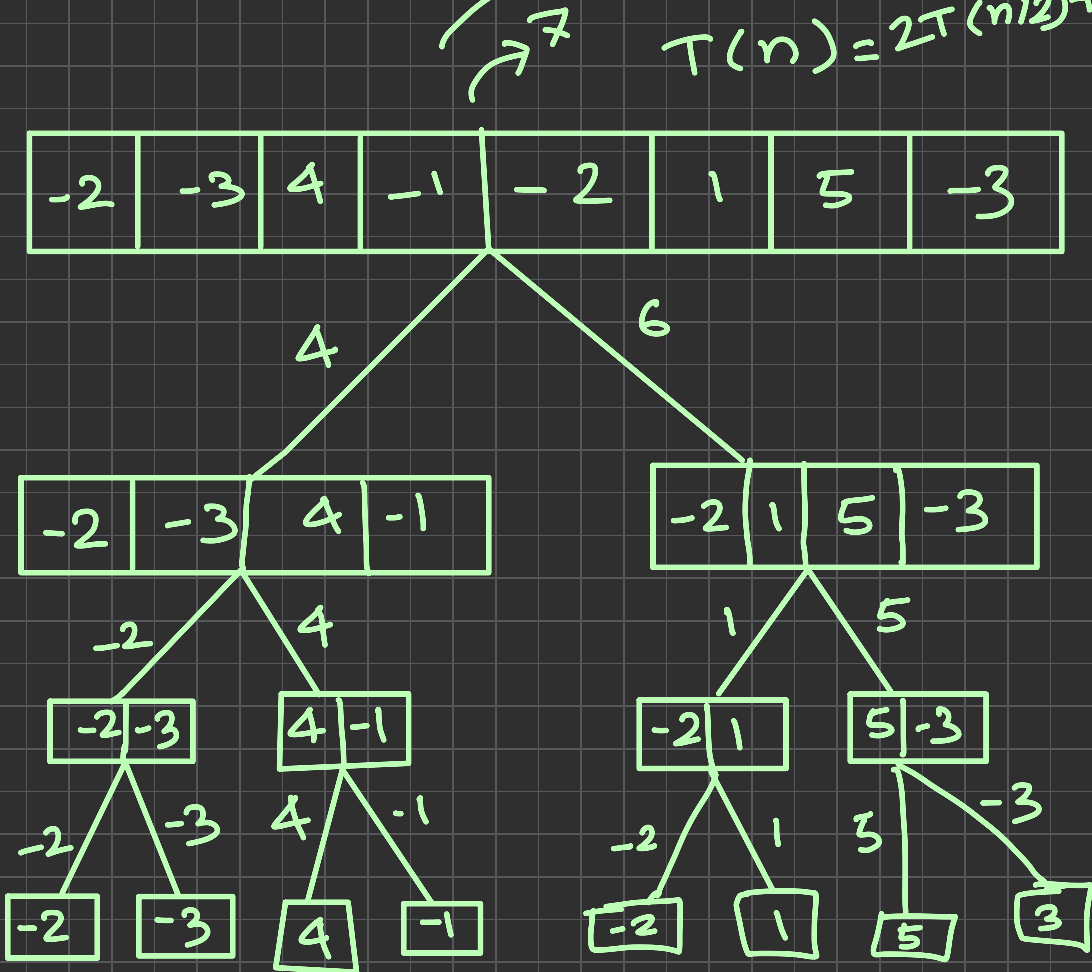
↓

-1 is skipped

Process

$O(n\log n)$

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



return max(lss, rss, css)

lss = left sum subarray
rss = right sum subarray
css = cross sum subarray

lss = returned value left subtree

rss = returned value right subtree

css = left (max continuous sum) +
right (max continuous sum)

return max (lss, rss, css)

For example

2	-1	5	3	-2	1
6	4	5	3	1	2

$5 + (-1)$ $\quad$ $3 + (-2)$

$4+2$

$6, 4, 5, 3, 1, 2 \Rightarrow$ continuous sums

$css = 6+3$