Greedy → Optimisation ← DP, B&B

Problems

Largest No

295

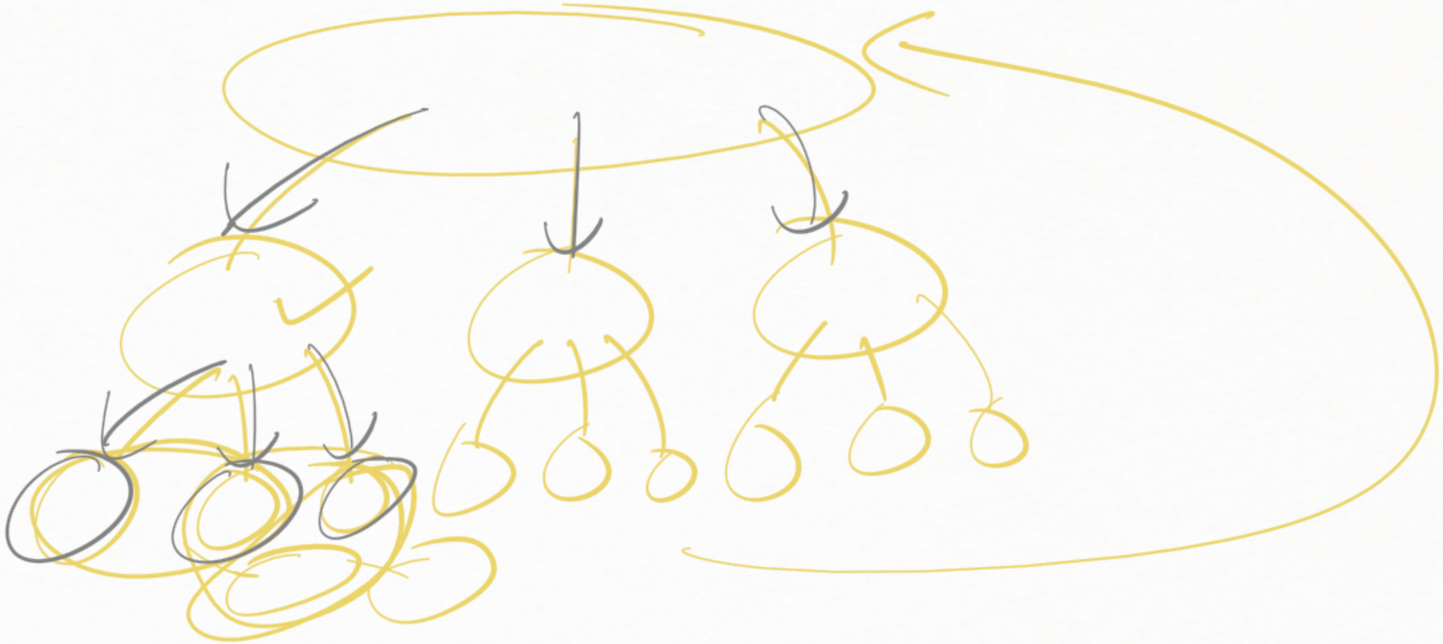
952

Greedy Choice

Next Subproblem

Divide & Conquer

Divide a problem into similar but not overlapping subproblem to solve them recursively and combine them in the end.

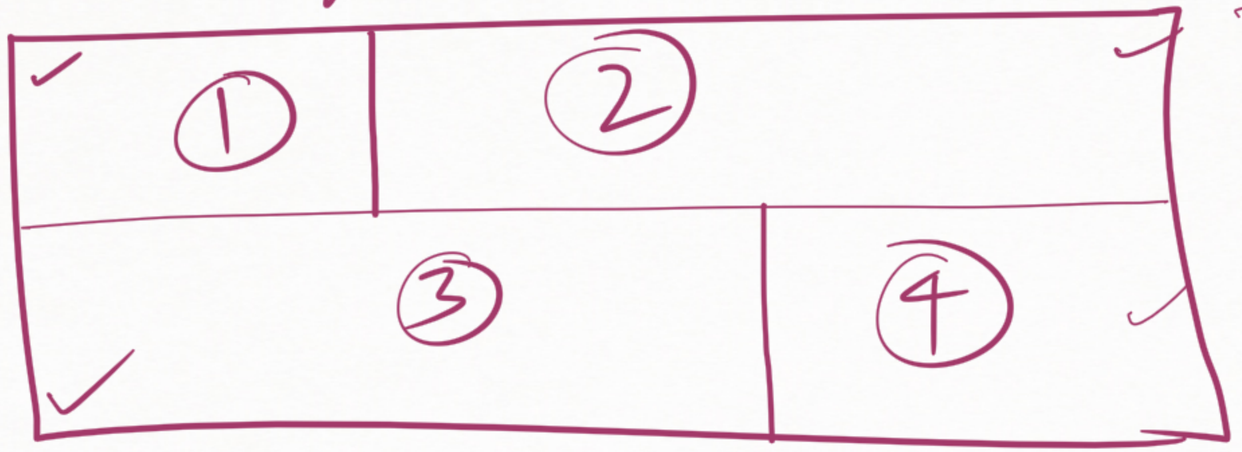


British
↓
India
BIT

Divide

A problem to be solved

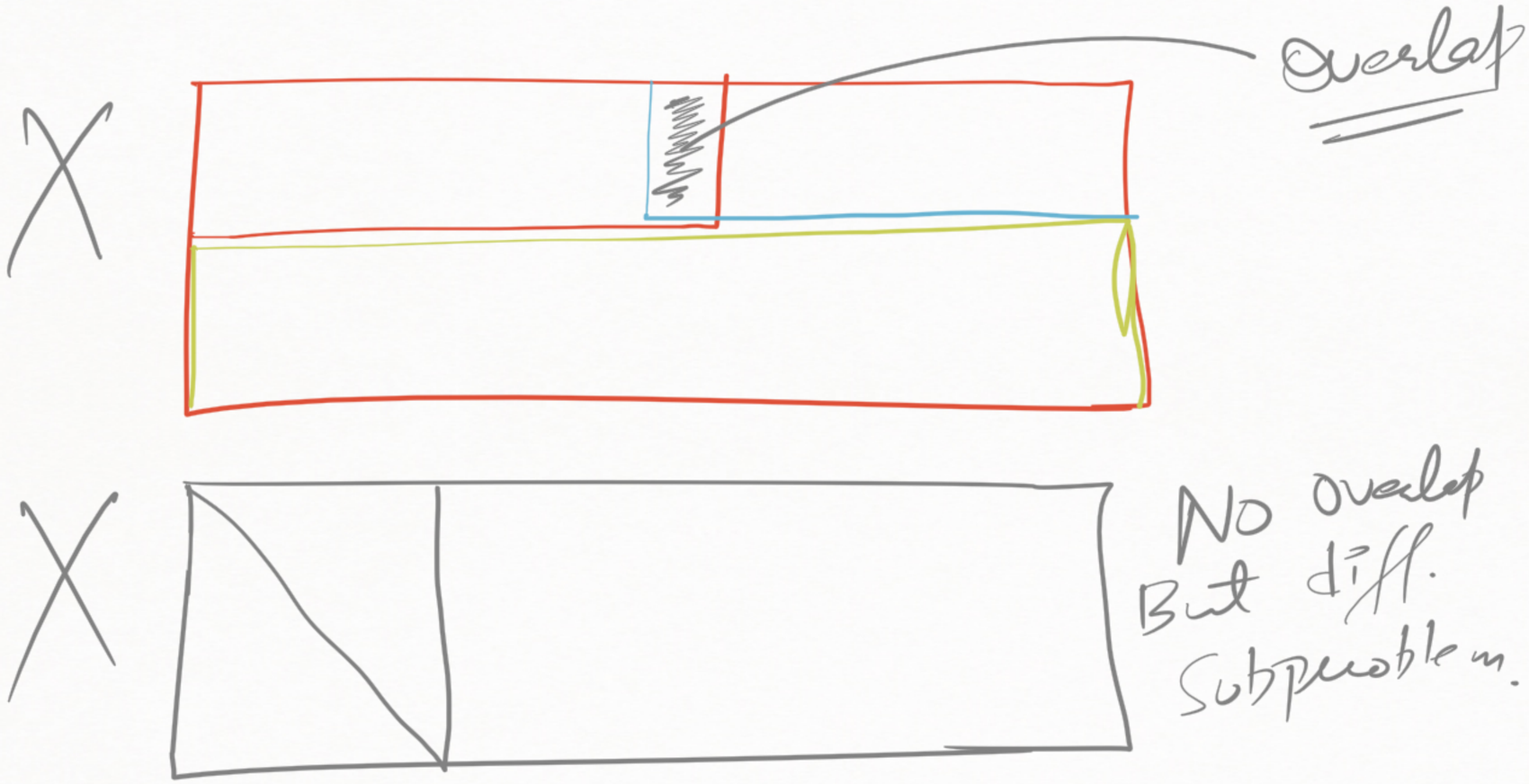
Break Problem into non-overlapping subproblems of the same type



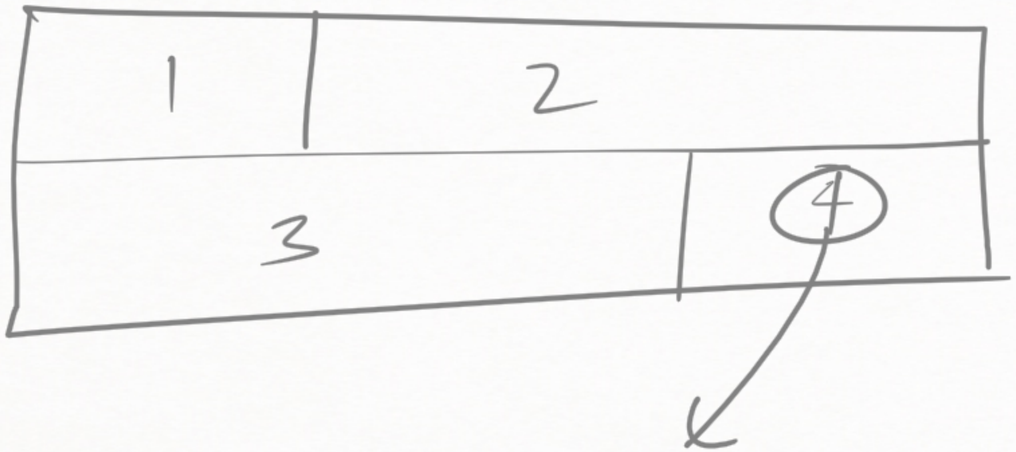
4 Diff. subproblems

Valid Divide

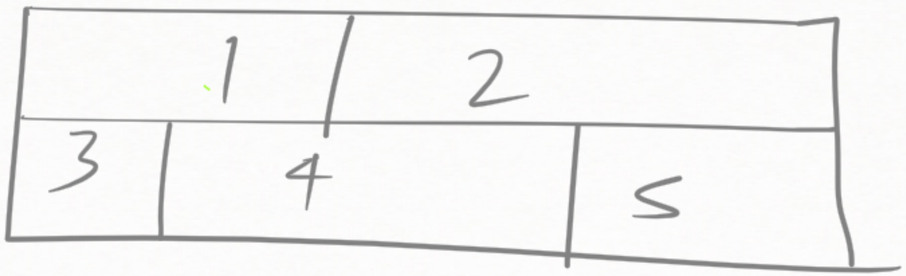
Invalid Divide Problem



How each subproblem is solved?



4 subprob.

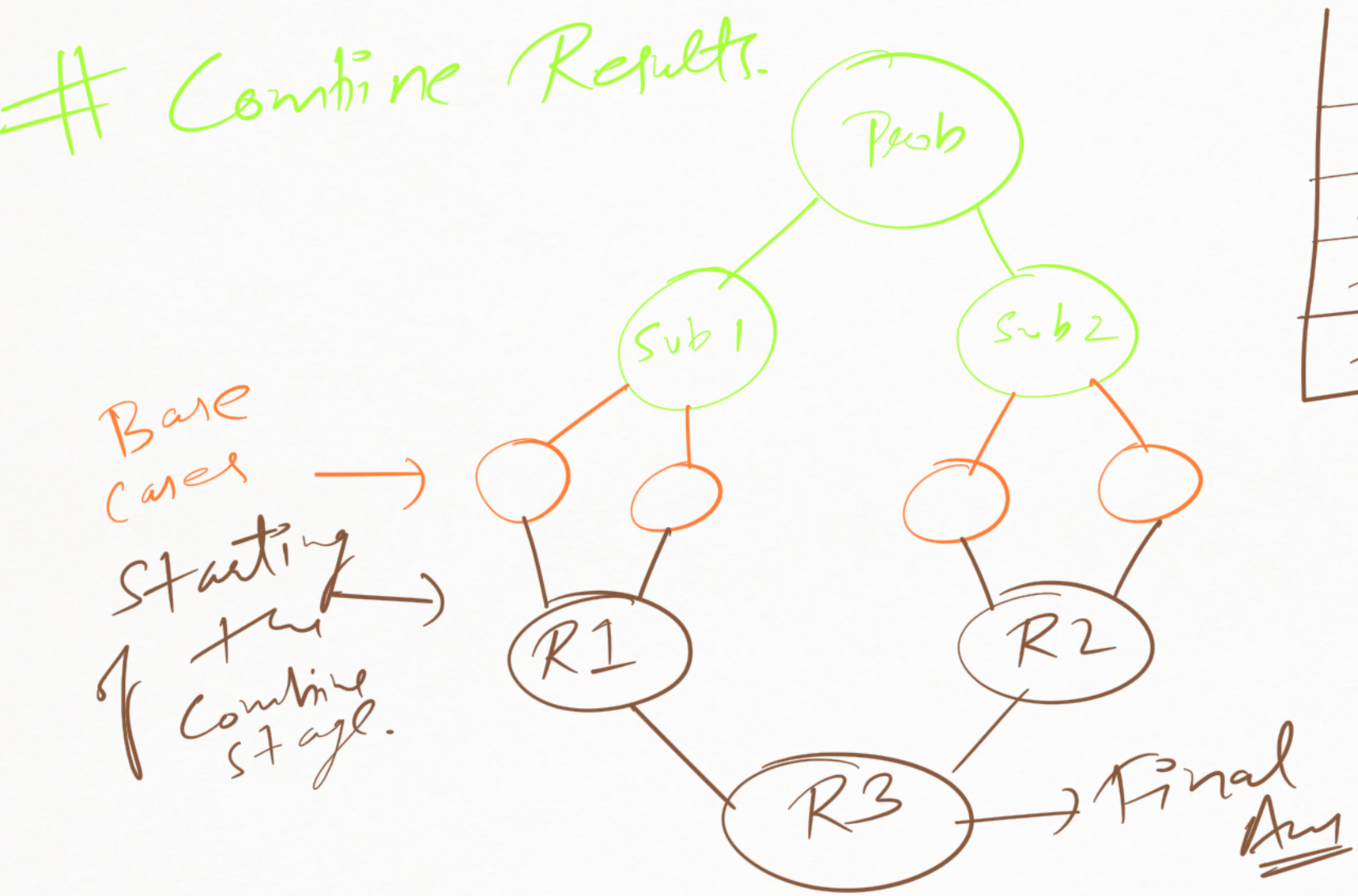


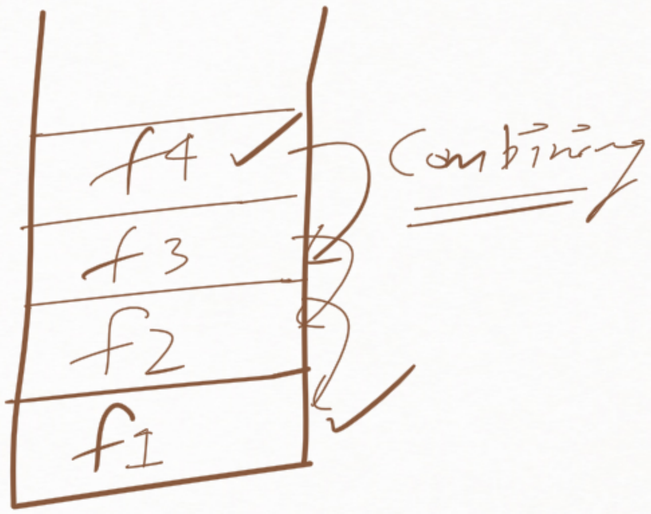
5 sub prob

Do it till Base Case is arrived

Terminating condition for recursion.

Combine Results.





Divide & Conquer Strategy

1. Break into non-overlapping subproblems of the same type.
2. Solve subproblems.
3. Combine Results.

21 Q's in Game

Guess that Number

You

[Has a hidden number $1 \leq x \leq N$]

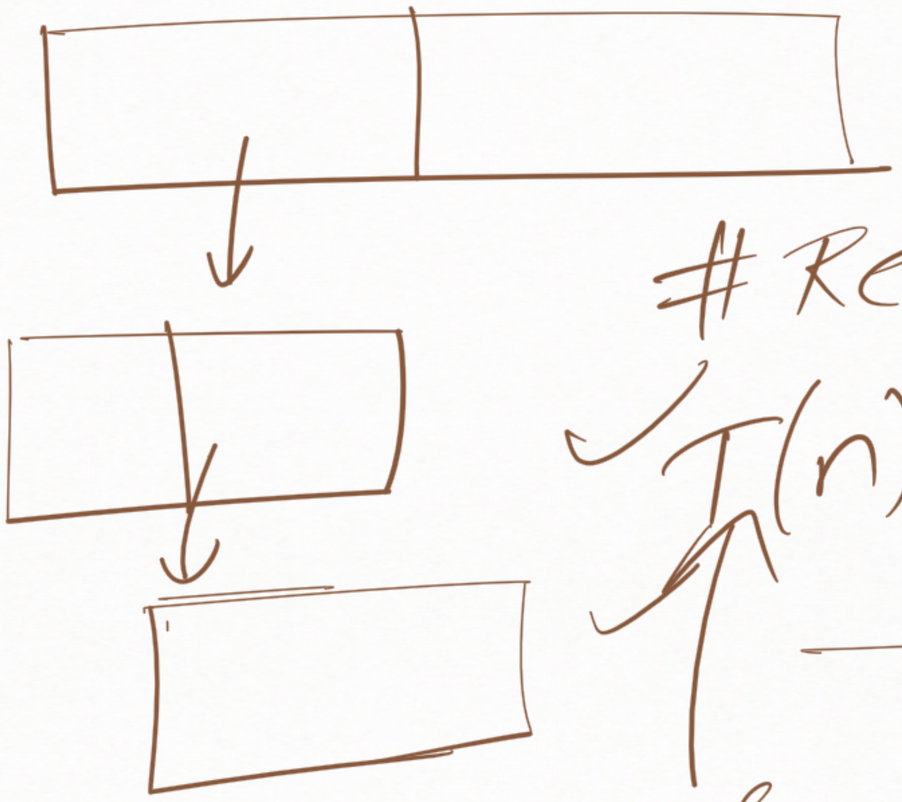
Computer

Computer will answer if your guessed no is equal to, or greater than the hidden no.

Binary Search

Exact same subproblem

Divide

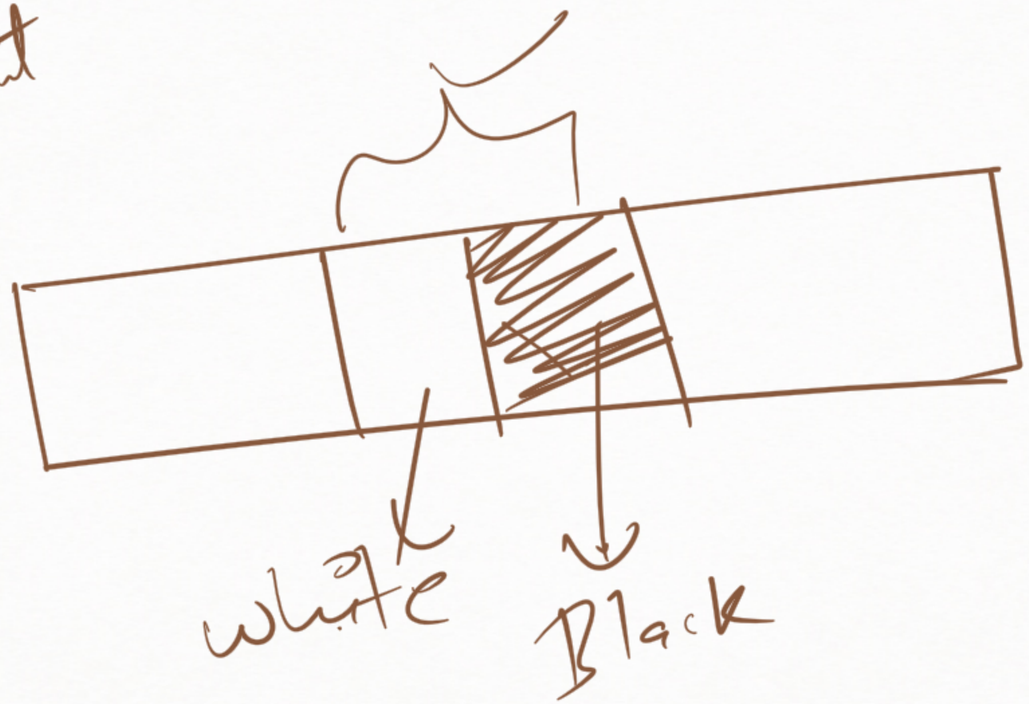


Recurrence Relation

$$T(n) = T\left(\frac{n}{2}\right) + C$$

Time complexity

Two adjacent cell game



Binary Search

Polynomial Multiplication (Divide & Conquer)

$A(x) = 3x^2 + 2x + 5$

$B(x) = 5x^2 + 1x + 2$

$A(x) \cdot B(x) = 15x^4 + 3x^3 + 6x^2 +$
$10x^3 + 2x^2 + 4x +$
$25x^2 + 5x + 10$
$= 15x^4 + 13x^3 + 33x^2 + 9x + 10$

$10 + 3 + 10$

11

B

Page rank dip

→ IB Search engine

→ Hooke

Multiplying polynomials

Input: Two $n-1$ degree polynomials:

$$a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0$$

$$b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \cdots + b_1 x + b_0$$

Output: The product polynomial:

$$c_{2n-2}x^{2n-2} + c_{2n-3}x^{2n-3} + \cdots + c_1 x + c_0$$

where:

$$c_{2n-2} = a_{n-1}b_{n-1}$$

$$c_{2n-3} = a_{n-1}b_{n-2} + a_{n-2}b_{n-1}$$

...

$$c_2 = a_2 b_0 + a_1 b_1 + a_0 b_2$$

$$c_1 = a_1 b_0 + a_0 b_1$$

$$c_0 = a_0 b_0$$

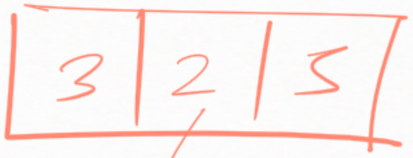
Multiplying Polynomials

Example

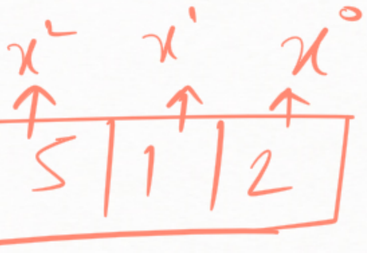
Input: $n = 3, A = (3, 2, 5), B = (5, 1, 2)$

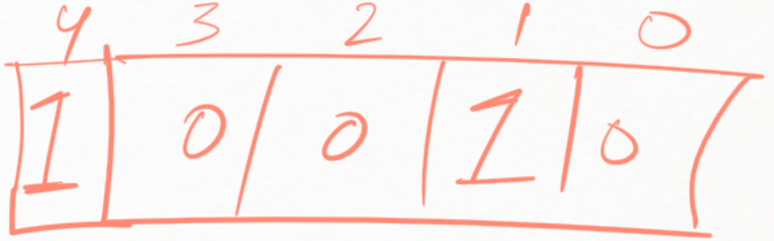
$$A(x) = 3x^2 + 2x + 5$$
$$B(x) = 5x^2 + x + 2$$
$$A(x)B(x) = 15x^4 + 13x^3 + 33x^2 + 9x + 10$$

Output: $C = (15, 13, 33, 9, 10)$



$3x^2 + 2x + 5$



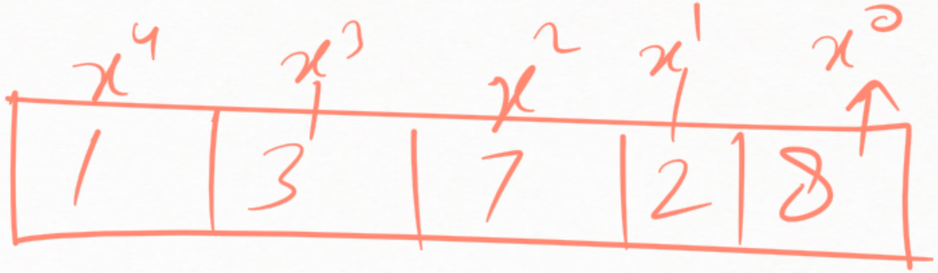


$5x^2 + 1x + 2$

$1x^{(9)} + 1x$

$2*3 - 1$

$x + 1$



$x^4 + 3x^3 + 7x^2 + 2x + 8$

$3x^2 + 7x^5 + 2$

Every polynomial can be written in an Array.

5	4	3	2	1	0
7	0	0	3	0	2

3	2	1	0
4	0	2	0

$\rightarrow 4x^3 + 2x$

Naive → Algo, without any efficient approach, just the way we can solve a problem.

→ w/o applying any algo technique like DP, Greedy, Divide & Conq.

Efficient → Opposite of the above ✓ DP, Greedy, DS

```
MultPoly(A, B, n)
pair ← Array[n][n]
for i from 0 to n − 1:
  for j from 0 to n − 1:
    pair[i][j] ← A[i] ∗ B[j]
product ← Array[2n − 1]
for i from 0 to 2n − 1:
  product[i] ← 0
for i from 0 to n − 1:
  for j from 0 to n − 1:
    product[i + j] ← product[i + j] + pair[i][j]
return product
```

Naive

Input : A, B → Polynomial

n → Size of Array.

→ optimal for the earlier problem

pair[i][j] → 2D Array

[[ ], [ ]]

[[1 2 3], [4 5 6]]

Product Array (2n-1)

Time Complexity of Naive Approach.

$O(n^2)$ → multiplications

$O(n^2)$ → Additions

$O(n^2)$ ✓

Divide & Conquer App.

$O(n^{1.58})$

$n, \log n$ ✗

why $O(n^{1.58})$ makes it better sense?

→ we talk about n values which are very very huge.

$2^{1.58}$ 2^2

→ n = 10 billion

$(10\text{billion})^{1.58} \ll (10b)^2$

Master Theorem. (Time Complexity topic)

→ Recurrence Relation (Recursion Algo)

Theorem

If $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$ (for constants $a > 0, b > 1, d \geq 0$), then:

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

① ② ③ → $\log_b^a$

$T(n) = 4T\left(\frac{n}{2}\right) + O(n) = O(n^2)$

Master Theorem Example 1

$$T(n) = 4T\left(\frac{n}{2}\right) + O(n)$$

$a = 4$

$b = 2$

$d = 1$

Since $d < \log_b a$, $T(n) = O(n^{\log_b a}) = O(n^2)$

$\log_2 4 = \log_2 2^2 = 2 > 1$

Recurrence relation

fun(n)

n task

fun($\frac{n}{2}$)

fun($\frac{n}{2}$)

fun($\frac{n}{2}$)

fun($\frac{n}{2}$)

$O(n^2)$

Recurrence Relation [Recursive Algo]

function(n)

{

print() → (1)

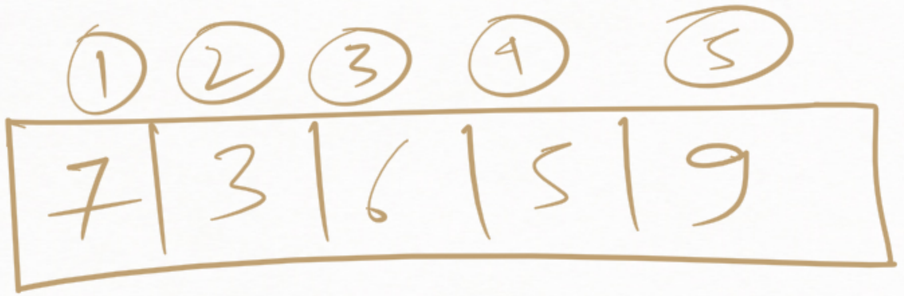
function (n/2)

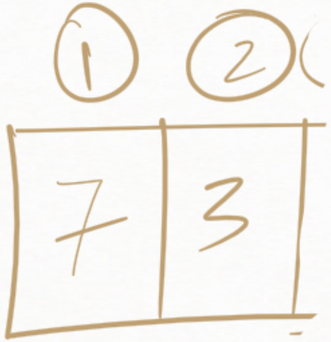
}

Suppose T(n) is the time taken

$T(n) = 1 + T(\frac{n}{2})$

Linear Search



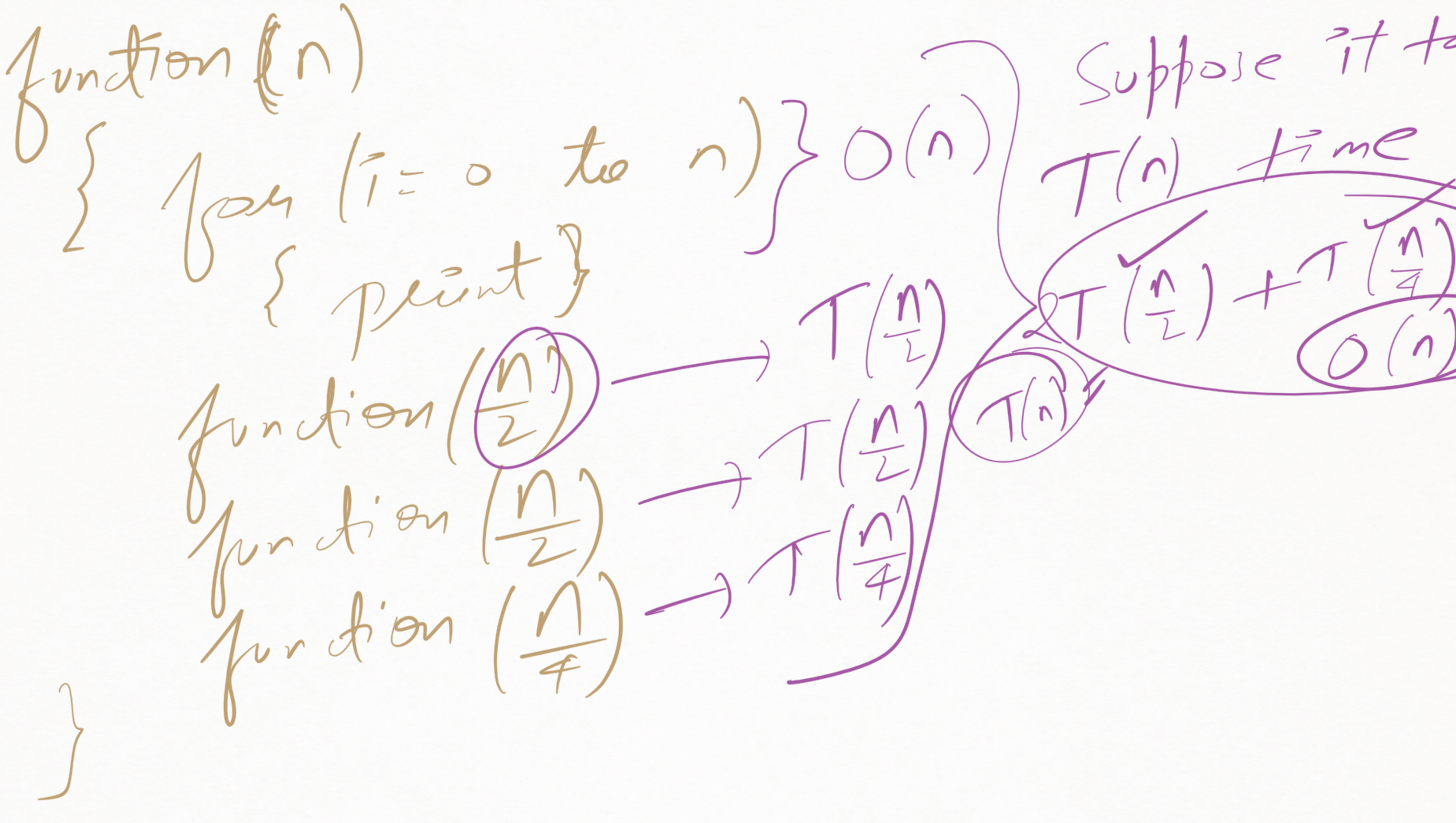


2 units of time

≤ units of time

$O(n)$ ← $n \leq 2$

Algo


function(n)
{ for (i = 0 to n) } O(n)
{ print }
function(n/2) → T(n/2)
function(n/2) → T(n/2)
function(n/4) → T(n/4)
}
Suppose it t
T(n) time
T(n/2) + T(n/4)
O(n)
T(n)

fun(n){

print( )

fun(1)

fun(1)

fun($\frac{n}{2}$)

}

$T(n)$

$T(\frac{n}{2})$

$$T(n) = 3O(1) + T\left(\frac{n}{2}\right)$$

$O(n)$ ✓ n units of

But $T(n)$ can take a… amount of time, tho d… upon n.

time → 2^n, logn

'just a math …

$$T(n) = O(n^2) + T\left(\frac{n}{4}\right)$$

```
function(n){
    for i = 0 to n
        {for j = 0 to n
            print(i, j)
        }
    function(n/4)
}
```

$O(n^2)$

$T(n)$

$T\left(\frac{n}{4}\right)$

er Theorem Example 2

recurrence eqn

$$\rightarrow T(n) = 3T\left(\frac{n}{2}\right) + O(n)$$

$a = 3$

$b = 2$

$d = 1$

$d < \log_b a$,

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

$d = 1$

> 1

< 2

$a = 3$ =

log

$b = 2$

0

er Theorem Example 3

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$a = 2$

$b = 2$

$d = 1$

$d = \log_b a$,

$= O(n^d \log n) = O(n \log n)$

$d = 1$

$a = 2 = 1$

$b = 2$

$O(n^{\log_2 1} \log n)$

er Theorem Example 4

$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

$a = 1$

$b = 2$

$d = 0$

$d = \log_b a$, $T(n) = O(n^d \log n) =$
$\log n) = O(\log n)$

$O(n^0) = O(1)$

$O(n^0 \log n) =$ dt

$a = 1$

$b = 2$

$d = 0$

$n^0 = 1$

ter Theorem Example 5

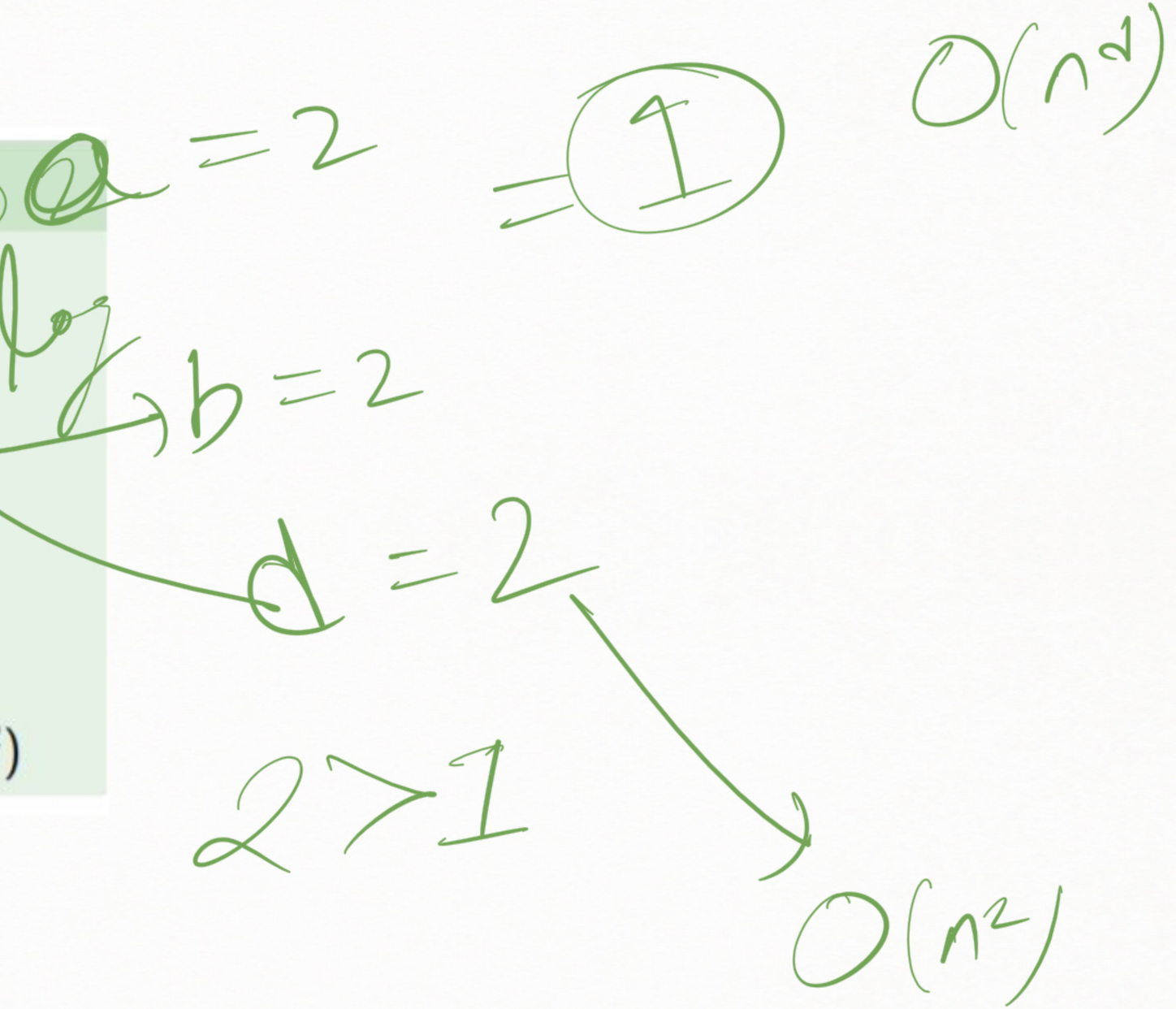
$$T(n) = 2T\left(\frac{n}{2}\right) + O(n^2)$$

$a = 2$

$b = 2$

$d = 2$

$d > \log_b a$, $T(n) = O(n^d) = O(n^2)$



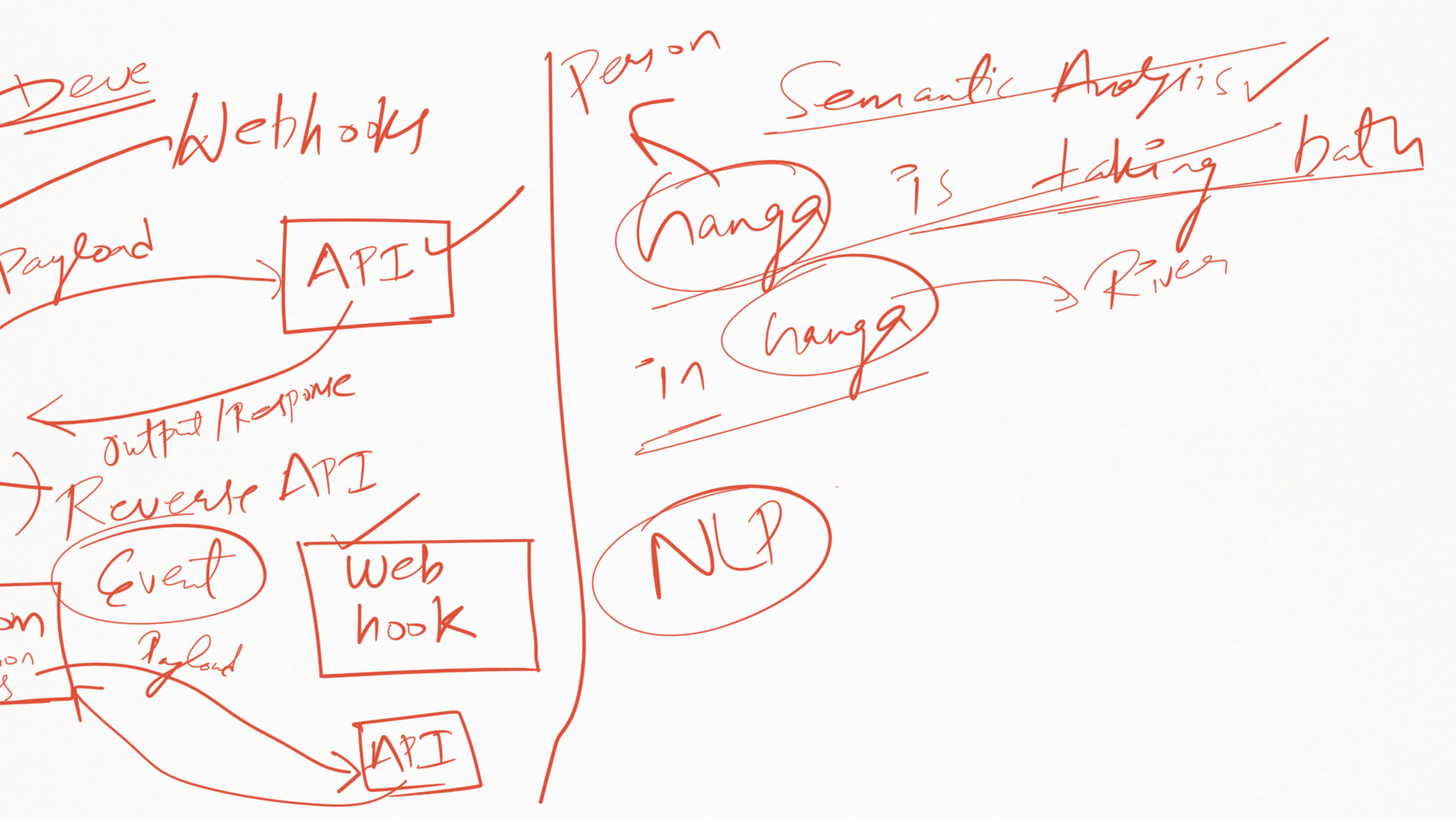
orem

$n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$ (for constants
$b > 1, d \geq 0$), then:

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

the bigger one is kept

$\Rightarrow O(n^{\text{bigger}})$


Webhooks
Payload
API
Output/Response
Reverse API
Event
web hook
Payload
API
Person
Semantic Analysis
is taking bath
in
NLP
River