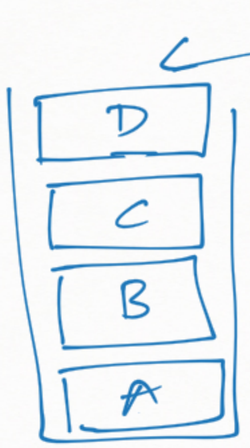
LIFO →

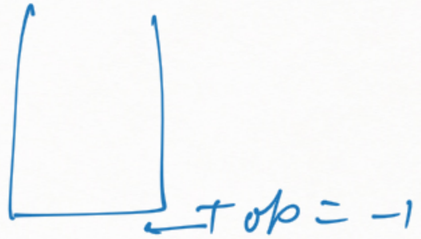
IN First

A
B
C
D

OUT out

D
C
B
A

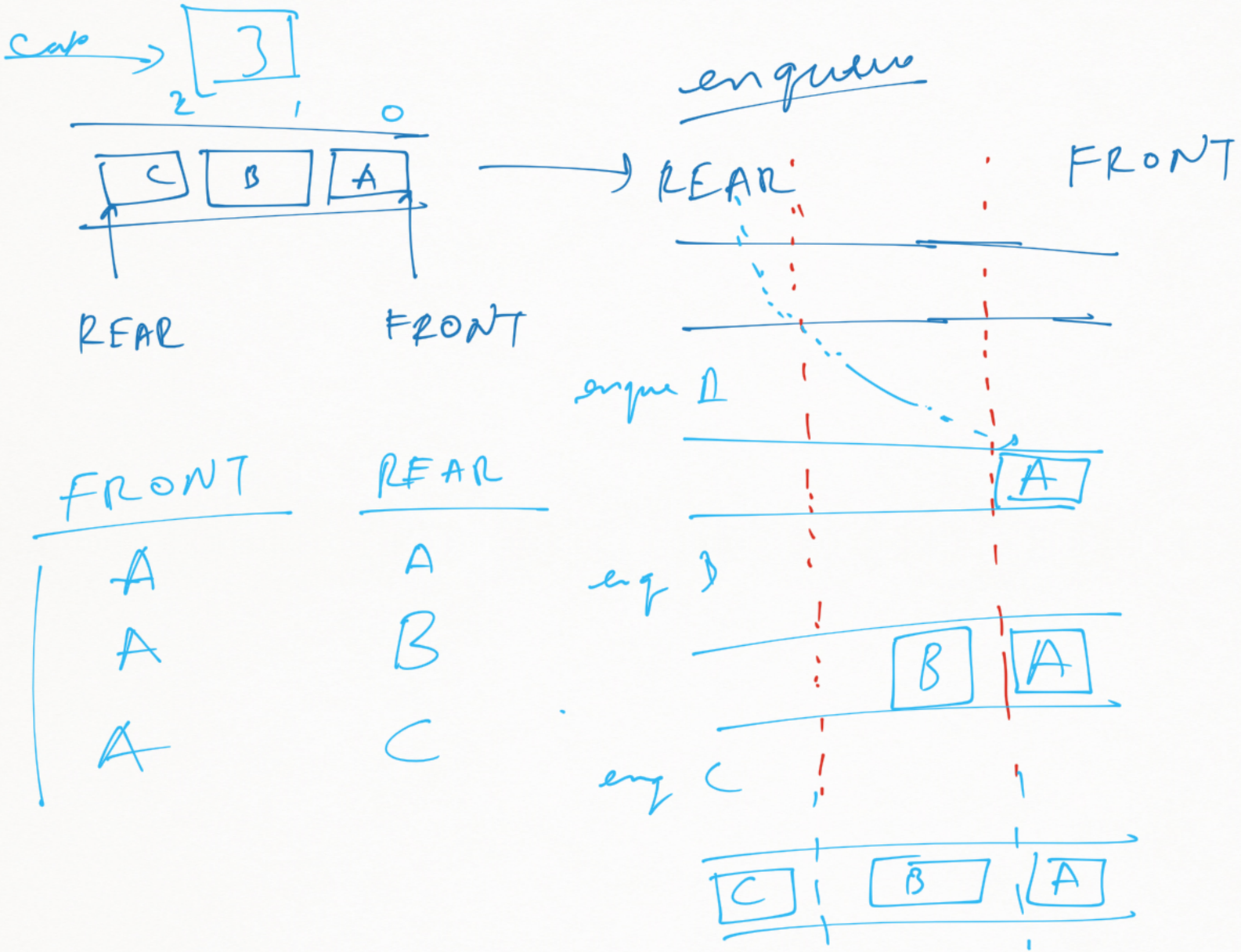




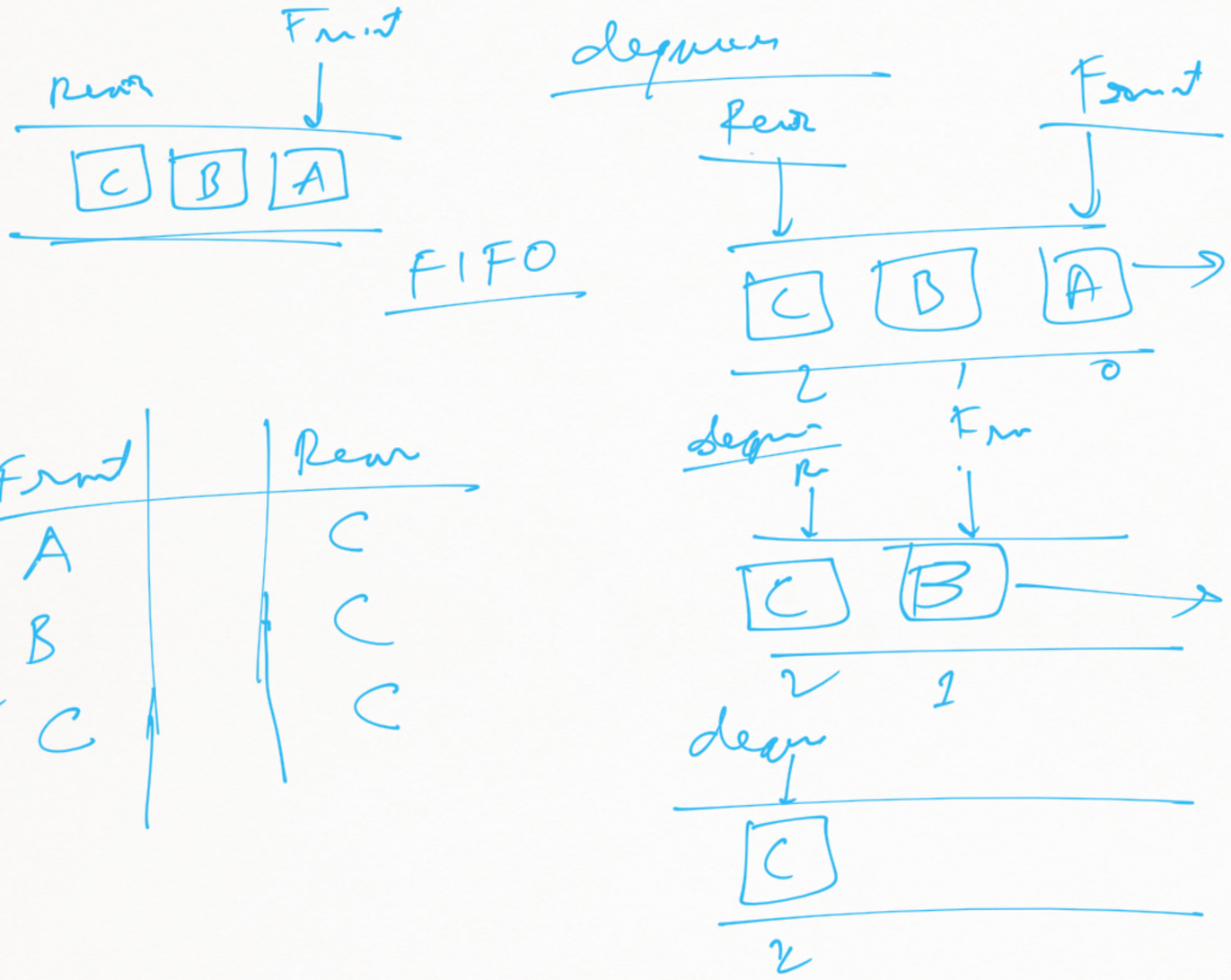
Queue

FIFO

A → B → C → D



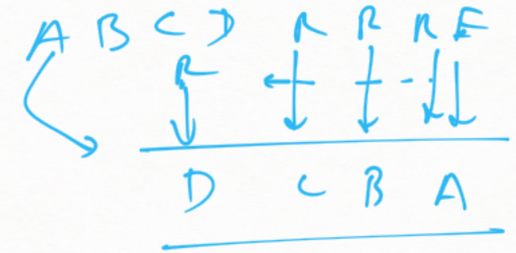
FRONT	REAR
A	A
A	B
A	C

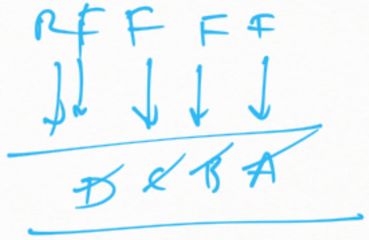

Front
Rear
C
B
A
FIFO
Front
Rear
C
C
C
Rear
Front
C
B
A
2
1
0
Rear
Front
C
B
2
1
Rear
C
2

Stack using queue

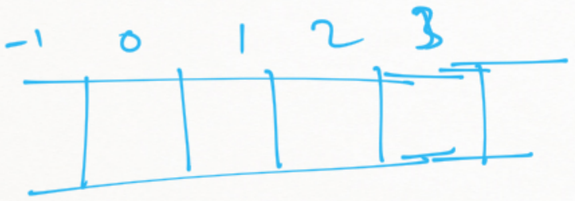
Stack → LIFO

queue → FIFO

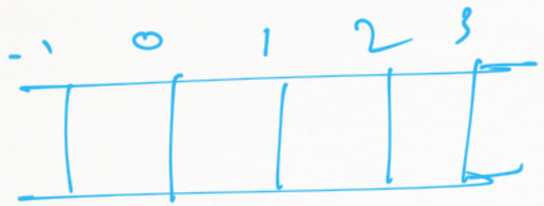


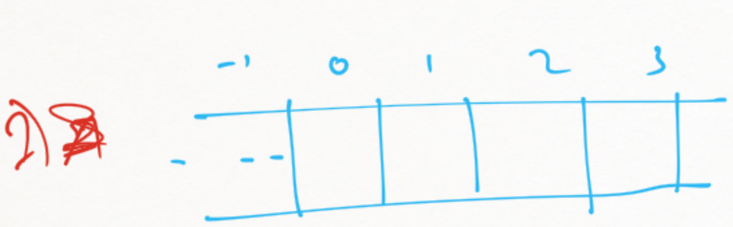


A

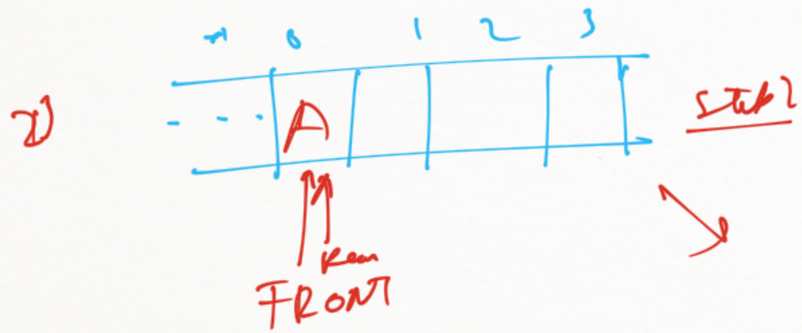


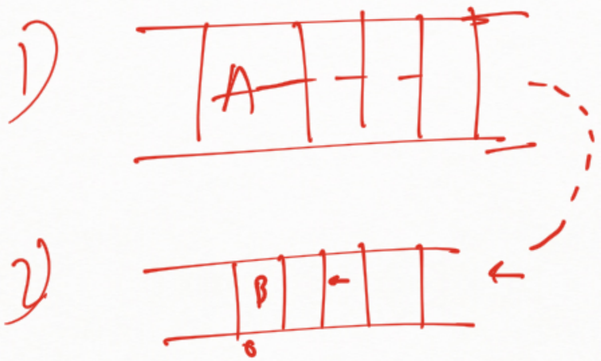
B

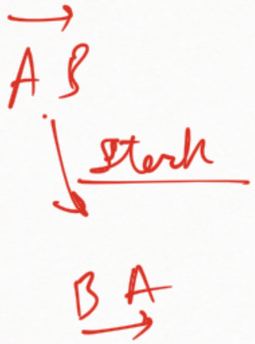


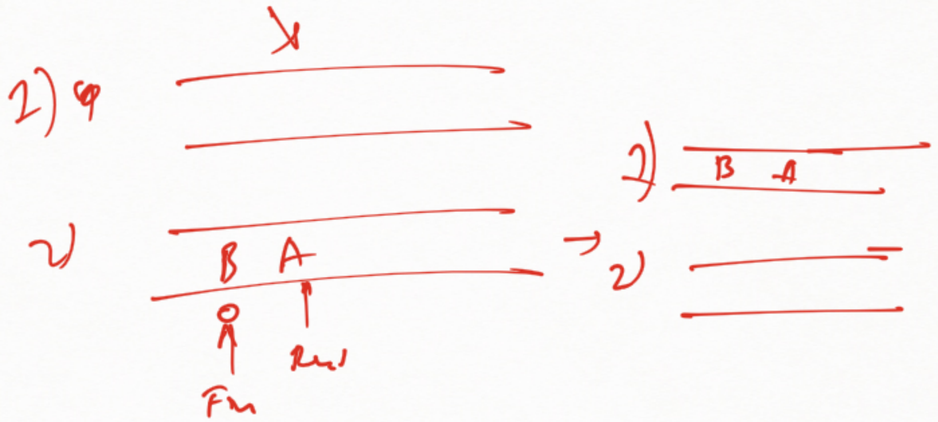


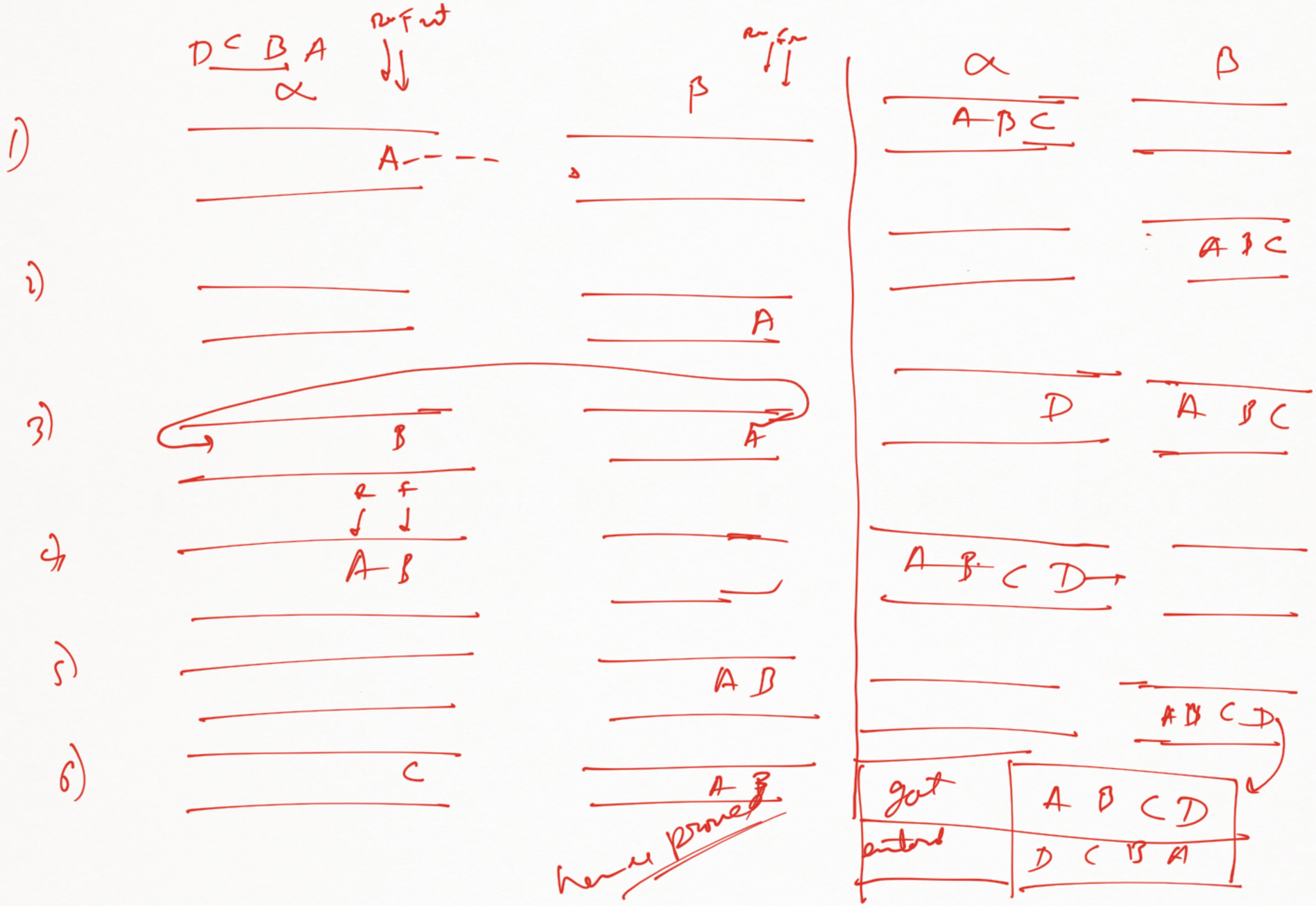
dequeue from B and enque to A



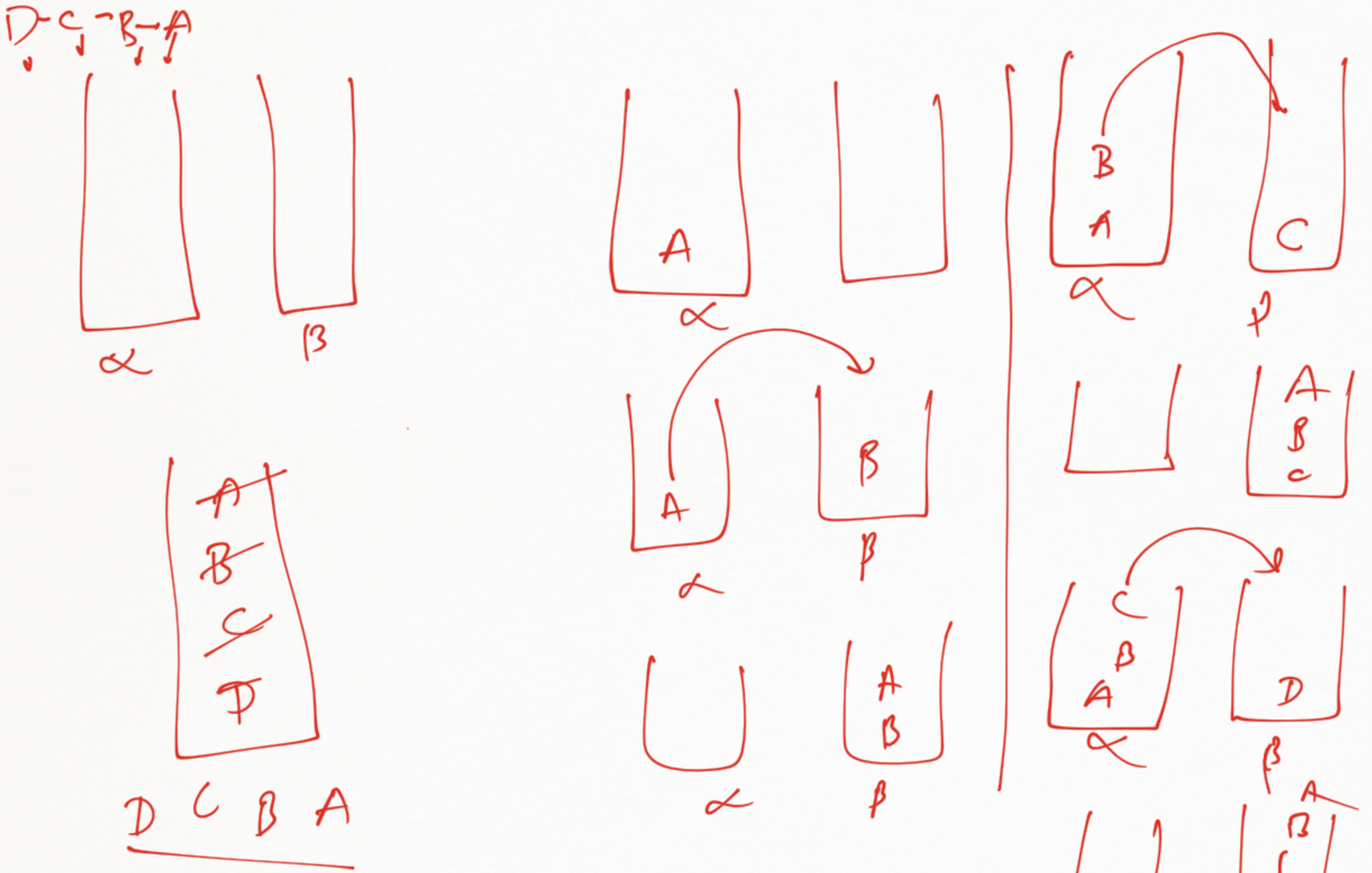







D C B A
α
ref
ref
β
α
β
A B C
1)
A
2)
A
A B C
3)
B
A
D
A B C
ref
4)
A B
A B C D
5)
A B
A B C D
6)
C
A B
got
entered
A B C D
D C B A

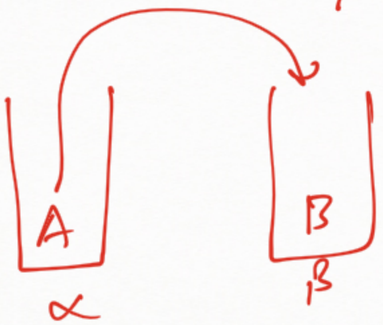
Queue using stack



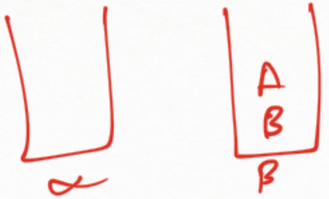
D C B A →

→ Step 1

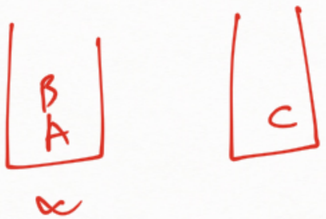
Step 2

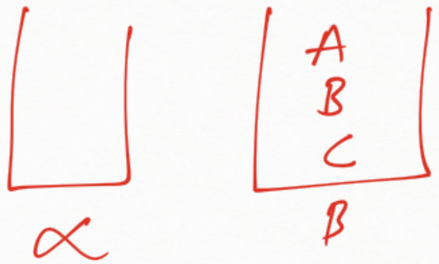


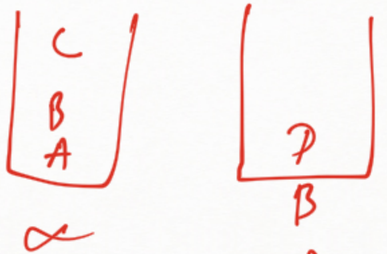
Step 3



Step 4







A → B → C → D

Code 1

1 s

30 s

TIME COMPLEXITY

Space

Time

A > B

A < B

A = B

Code 2

O(1)

Fast

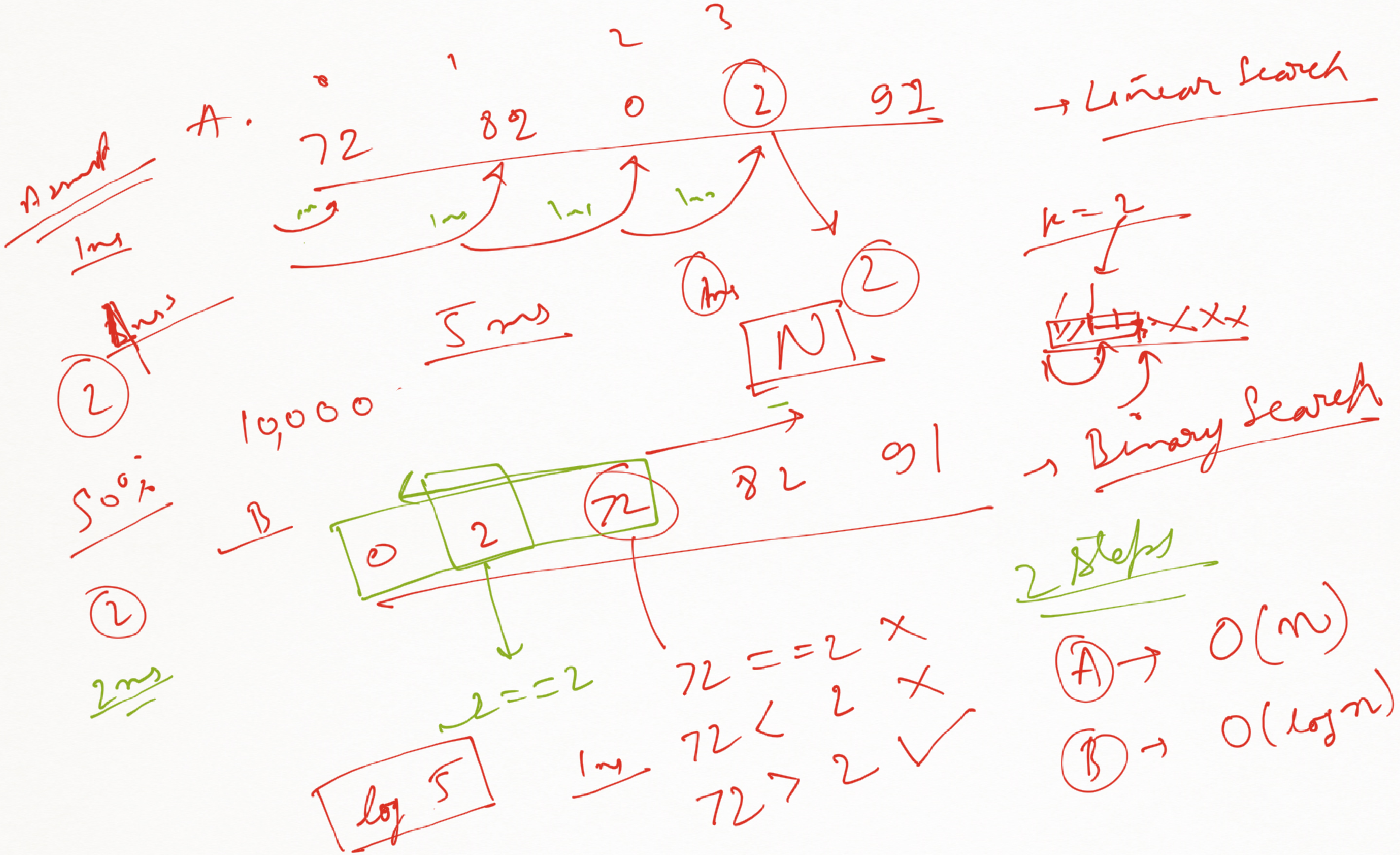
efficiency

Asymptotic notation

worst → O Big oh Notation

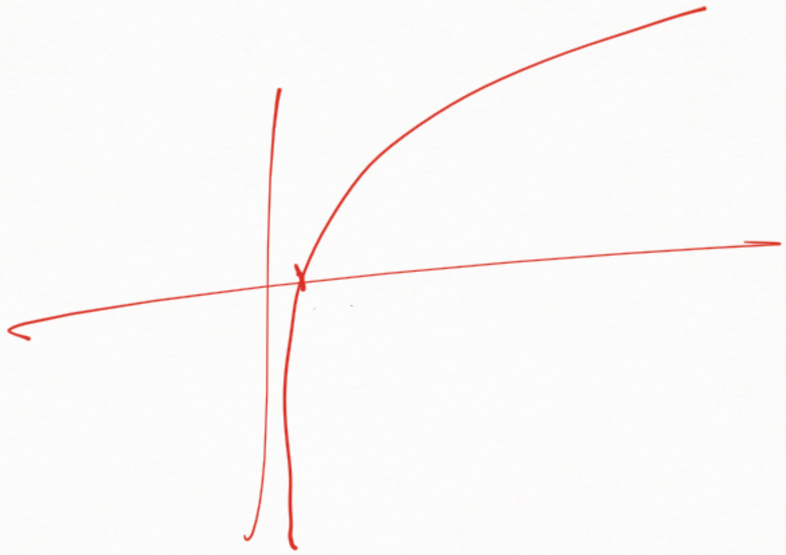
Ω Big Omega

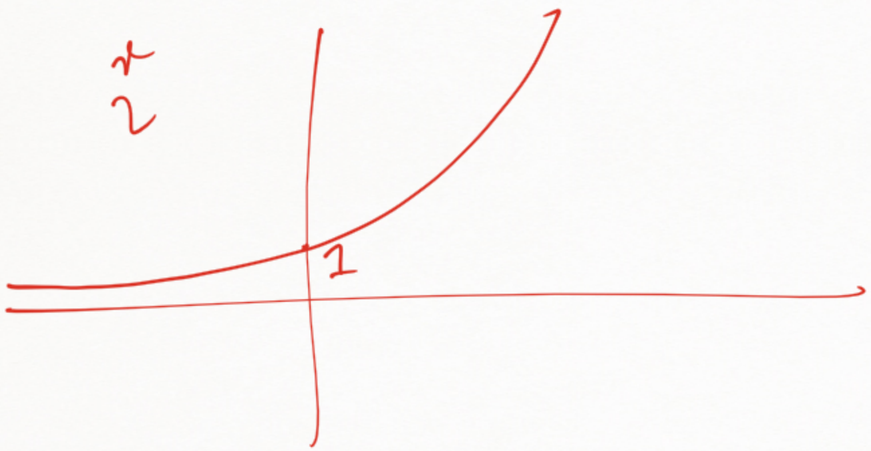
Θ Notation


A.
72 82 0 2 91
→ Linear Search
5 ms
10,000
B
0 2 72 82 91
→ Binary Search
k = 2
2 steps
2 == 2
72 == 2 ✗
72 < 2 ✗
72 > 2 ✓
log 5
A → O(n)
B → O(log n)
50%
1ms
2ms

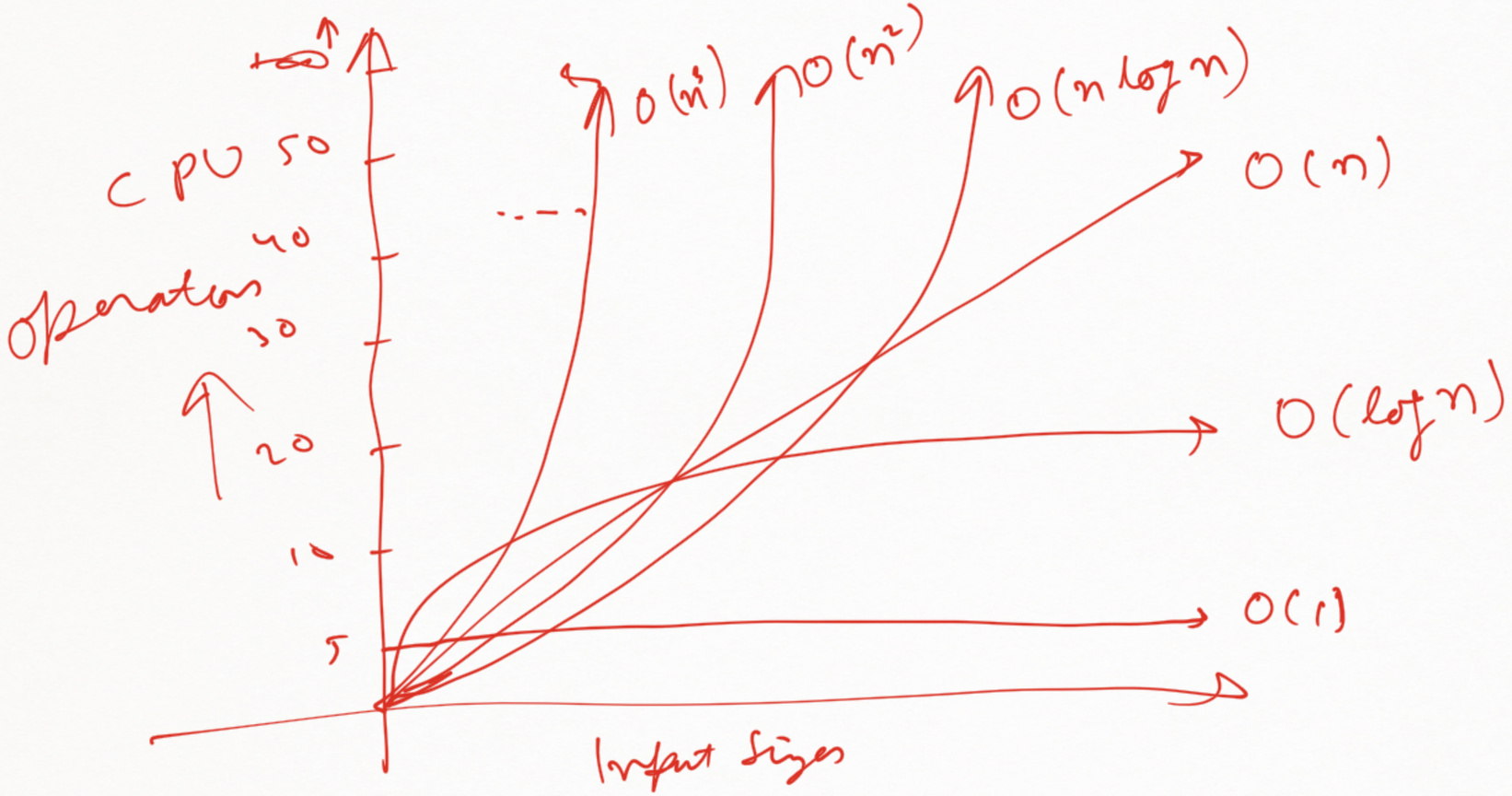
$2 = 2^x$

$\log 2 = x$





$$T(n) = O(4n^2 + 7n + 3)$$


CPU
Operation
50
40
30
20
10
5
O(n!)
O(n^2)
O(n log n)
O(n)
O(log n)
O(1)
Input Sizes

O(1) ⟶ O(n)

a = 5

b > c O(1)

S1
S2
⋮
SN

total = time(S1) + time(S2) + ⋯ + time(SN)

T(n) = t(S1) + t(S2) + t(S3) + ⋯ + t(Sn);

Sum of 3 numbers

```
def fun(a, b, c):
    sa = a          → O(1)
    sb = b          → O(1)
    sc = c          → O(1)
    s = sa + sb + sc → O(1)
                    → O(1)
    return s
```

$T(n) = O(1) + O(1) + \cdots$

$= 9O(1) \approx O(1)$

↑ input

for i in range(length): n

S1

S2

$$T(n) = n \times [t(S_1) + t(S_2) + \cdots]$$

$$= O(n)$$

```
for i in range(n):
    so
    for j in range(m):
        S1
        S2
```

$$T(n) = n \times [t(S0) + m \times [t(S1) + t(S2)]]$$

$$= n * t(S0) + n \times m \times [t(S1) + t(S2)]$$

$$= O(n \times m) \approx O(n^2)$$

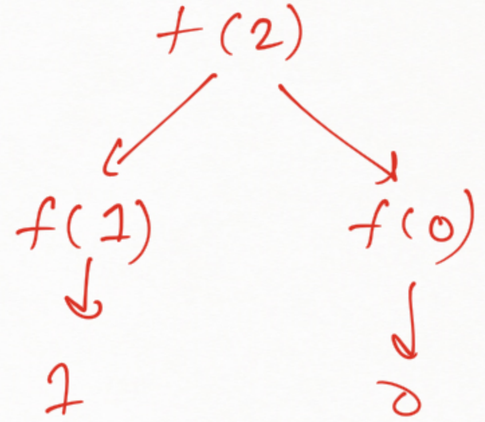
($m \to n$)

```
def func(n):
    if n < 0:
        return 0
    if n < 2:
        return n
    return func(n-1) + func(n-2)
```

$> 2^n$

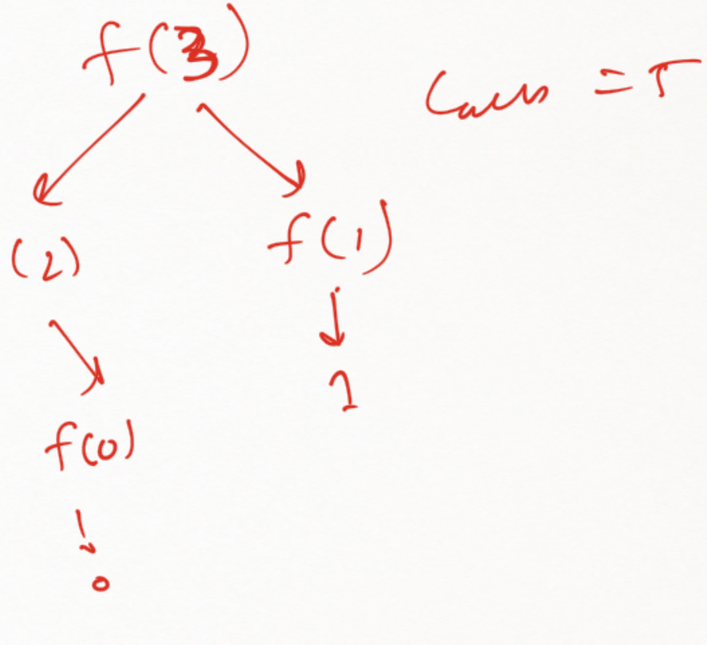
$\rightarrow O(2^n)$

$n = 2$



calls = 3

$n = 3$



calls = 5

$n = 4$