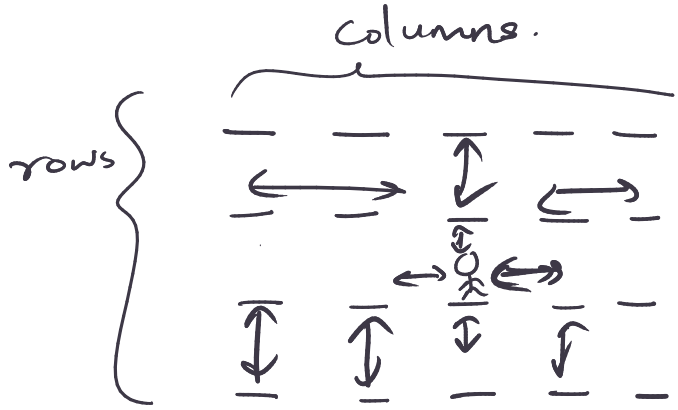
4 questions

1. Student swap problem



Q. Is it possible for all students to swap their position so that nobody is on its original seat?

Assumption : You can swap only once

o/p : Yes/No i/p : m n

Trick : start with easiest subproblem.

m=1 n=1 No

m=1 n=2 Yes

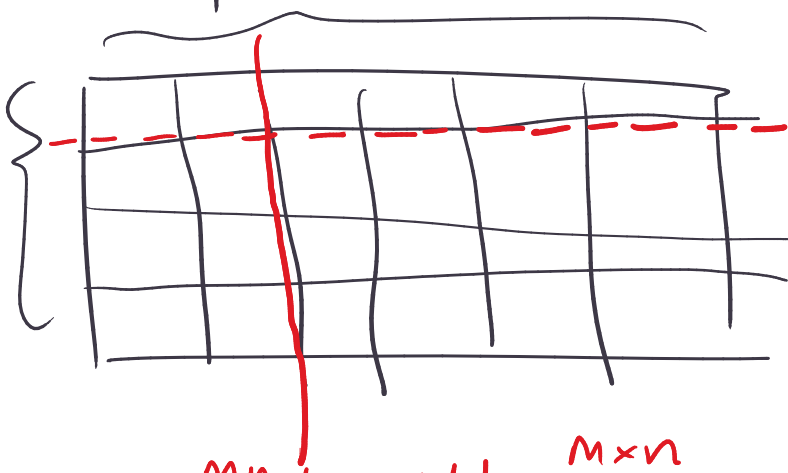
m=2 n=1 Yes

m=2 n=2 Yes

m=3 n=3 No

if m%2 != 0 and n%2 != 0 :
    ans = No
else
    ans = Yes

2nd problem : chocolate break problem



2 players: me & you

me → I start first

Q. Given m & n, can I win ??

mn-1 = odd

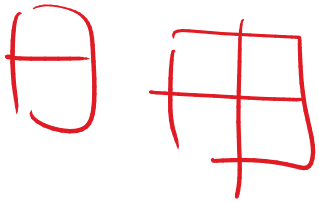
mn = even

m×n

mn-1

Sol^.

m=1	n=1	No
m=1	n=2	Yes
m=2	n=1	Yes
m=2	n=2	



if m%2 != 0 and n%2 != 0:

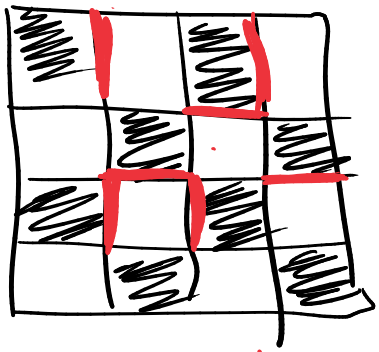
ans = No

else ans = Yes

Q.3 chessboard cutting problem

2 players: me & you

I start 1st.



Q. Can I win? Win cond^

My chessboard shouldn't get split into 2 parts.

Start

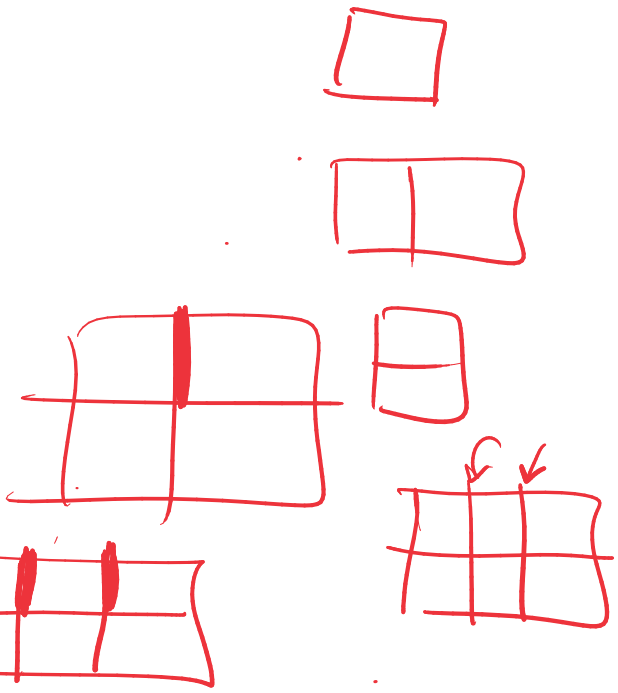
M=1 n=1 No

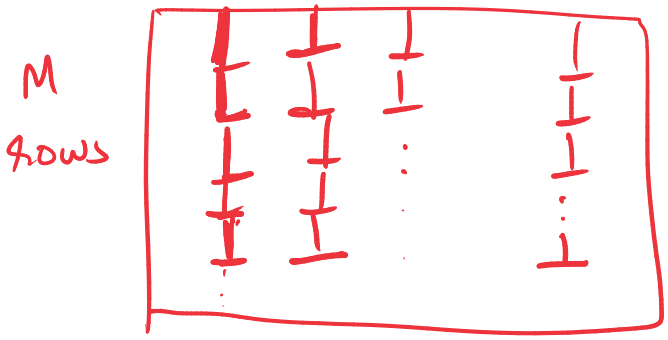
M=1 n=2 No

M=2 n=1 No

M=2 n=2 Yes

M=2 n=3 No





Max cuts = $(M-1) \times (n-1)$ odd

if (M-1)%2 !=0 and (n-1)%2 !=0:

ans = Yes

else ans = No

Eg^ 2 scenarios

[80 100 70 60) 200

Case 1 :

4 people, weight

↳ lift, 200 kg

Q. Is it possible for all to go?

Case 2 :

4 fruits → per kg price

200 Rs. budget.

Q. Can you buy all 1 kg each?

DSA : Abstract. (It lacks context)

24 What comes to your mind?

Information = Data + ~~Context~~

processing

Context ⇒ Problem solving skills ✗ How? ~~many~~

✗ abstract data ⇒ 1 question ⇒ 3 in context

ad-hoc

Template / parent question

↳ abstract question: Given 2 integers check if both of them are odd.

Variants / child question (context as much as possible)

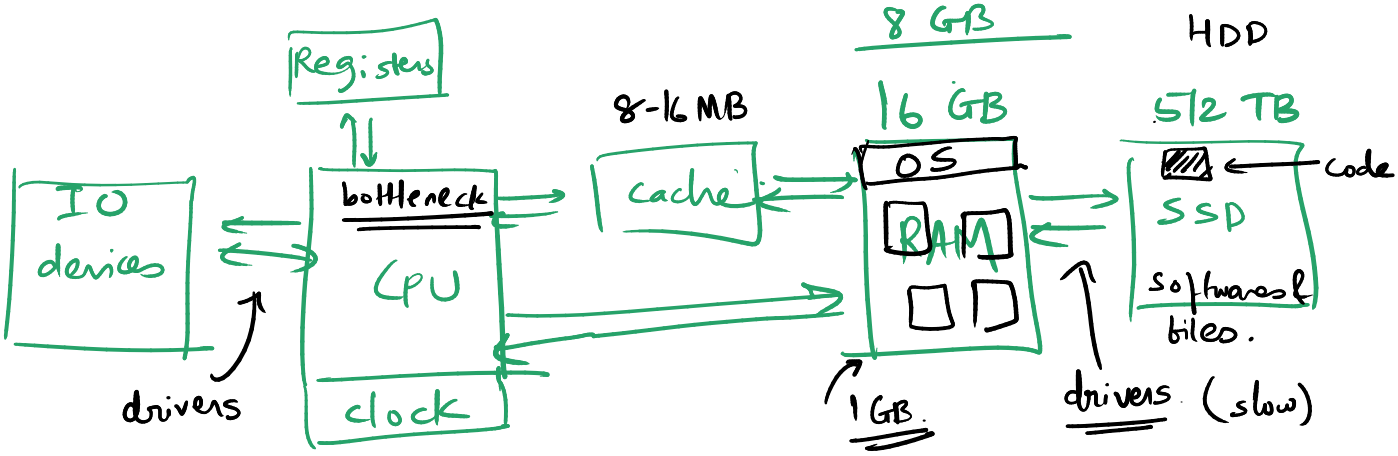
↳ student swap

↳ choc. break

↳ dress cut

Time complexity.

Desktop / laptop ⇒ what features / specs do you look for?



1. CPU and Hard disk do not communicate directly.
2. HDD ⟶ SDD | appl^n loading gets faster
3. RAM ↑ multitasking ⇒ a lot of things parallely.
4. CPU ⇒ "3 GHz" ⇒ 3×10^9 instructions / second

~ 1.8 - 2 GHz

OS, Drivers, Task managers, Chrome | Code ⇒ 10^8 instructions / second.

— × ——— × ——— × ——— × ———

1st : 10^8 instructions / second : CPU

2nd : 32 GB : RAM

10^8 instructions | Code

Amazon.

1. How many customers ?

10^6

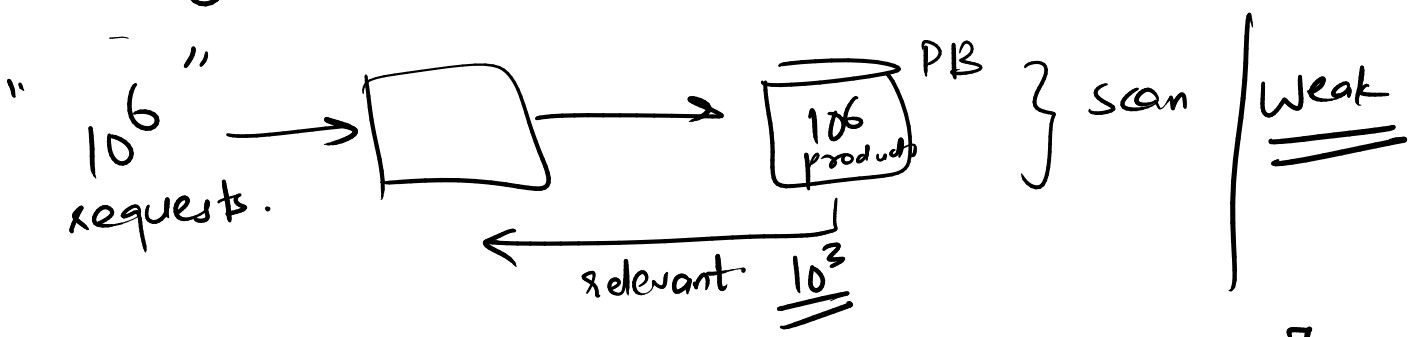
100 M ~ 10^8 customers

1% are active

10^6

How many products ? 10^6 products

10^8



2. How many use UPI ? ~ 30 cr (10^7)

3×10^8 people

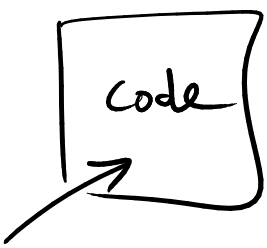
1 person in 1 year ⇒ 100

10^{10} } Volumes of transactions ~ 4-5 Trillion !!

Conclusions :

"10^8 operations" ⟶ 1 second

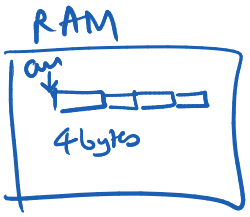
P.S.



⟶ if I have to cap myself at 1 second only, What's the max input size I can take ?

Naive: Count instructions.

sum of n integers given in an array



Doesn't work !!

```
int arr[n] = {...}      ⇒ 1
                1 1
for (int i = 0 ; i < size ; i++)      1 1
    ↳ sum = sum + arr[i]              1 2 2
print (sum)                           1
```

"4n+2" incorrect

powerful idea

Total : $1 + 1 + n + 2n + (n-1) + 1$

$= 4n + 2$ instructions.

$10^8 = 4n + 2$ $\quad n \sim =$ | max value

1 s.

programming langs ⇒ differently.

```
a = [1, 2, 3]
int a[3] = {1, 2, 3}
```

py list
C++

~~100n+64~~
~~4n+2~~

exact formula

intel CPU
AMD CPU
Apple M1/M2

→ 1. code
2. lang
3. hardware

class of: formulas $O(n) \sim$ C++ $\{ 4n+2,\ 100n+3,\ n+1 \} \longleftrightarrow$ linear $an+b$

C++ 10^8 elements 1 s Py: 2 s

absolute number $\times$ 10^9 elements 20 s 20 times $\times$ ~ 20 times

order of growth

```
for (int i=0 ; i<n ; i++)
    for (int j=i ; j<n; j++)
        a = i+j
        print (a)
```

Exact

i: 0 1 2 ..... n-1

#j: n n-1 n-2 ... 1

print(a): $1 + 2 + 3 + \dots + n-2 \to n-1 + n$

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

$O(n^2)$ code $\sim$ $an^2 + bn + c$.

$O(n^2)$ 10^8 ops $n_{max} = 10^4$

$\sim$ 10 thousands

Table ★★ — 1 second cap, 10^8 ins. (1 s cap)

$n\sqrt{n}$

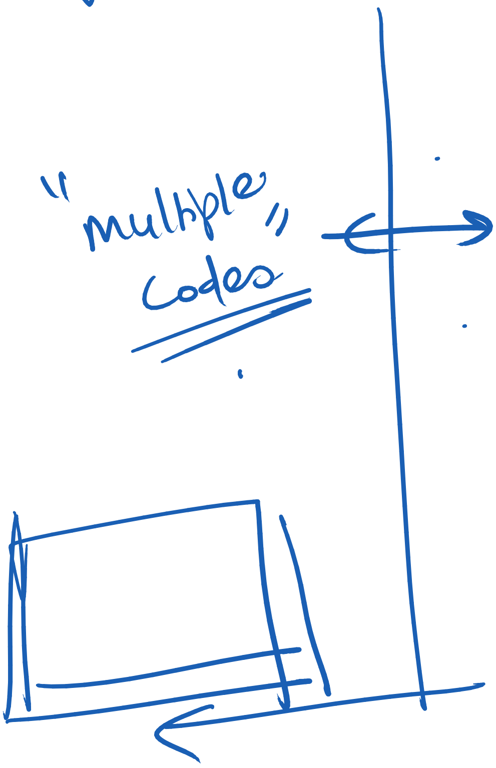
$n\log n$

~

Class of f^n's	max value of n.
$O(1)$	∞
$O(\log_2 n)$	2^{10^8}
$O(n)$	10^8
$O(n \log_2 n)$	10^7
$O(n\sqrt{n})$	10^6
$O(n^2)$	10^4
$O(n^3)$	$500 \sim 10^3$
$O(2^n)$	50
$O(n!)$	15

"multiple" codes ↔

code max value n → 10^6



n^2 code

$10^4 \Rightarrow 1\ s$

10 lakhs $10^6 \sim 10^{12}$ — 10^4 seconds

~3 hrs — $\frac{1}{8}$ — 10k — 86k 1 day ~24 hrs

3 Find all possible (a,b,c) pythagorean triplets < 100.

3 4 5 6 8 10 7 24 25 5 12 13 9 40 41

```
count = 0
for(int a = 0; a<n; a++)
    for(int b=a; b<n; b++)
        for(int c=b; c<n; c++)
            if (a*a + b*b = c*c)
                print(a,b,c)
                count += 1
print(count)
```

(a,b,c)

a < b < c

n=100

n=60

$O(n^3)$

10^3 1s — 1 to n

$10^4 \sim 10^{12}$ 3 hrs — a to n, b to n, c to n

n^2 60×60 seconds 0 to 60

pythagorean $\Longrightarrow O(n^3)$

10^8 limit per second

TC (n^3)

1000 times

$10^3 \longrightarrow (10^3)^3 = 10^9 \sim 10s$

10 times

$10^4 \longrightarrow (10^4)^3 = 10^{12} \sim 3hrs$

$10k$

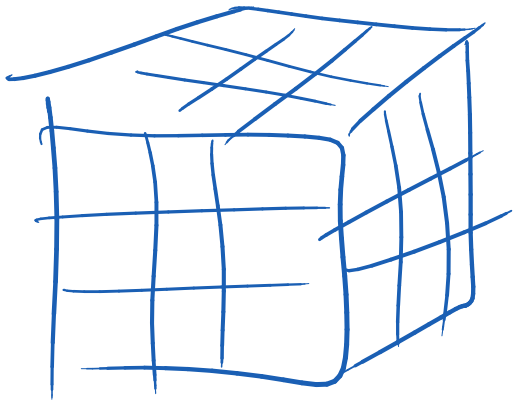
```
for (i = 0; i < 500; i++)
    for (j = 0; j < 500; j++)
```

$O(1)$ Yes.!! n

$n + 10^8 \rightarrow O(n)$

n rows

$O\left(\frac{n^3}{\log n}\right)$



3×3

6 faces

1s

1000 × 1000

1000 rows

25 ⟹

$10^6 \cdot \frac{n^3}{\log n}$

code ⟶ O(⟲ f(n))

n ↘ f(n)

10^8

8 GB

8×10^9 bytes

"RAM" ⟹

8 bytes → 1 int

2nd

OS: 1 GB 10^8 integers 10^9 integers

$n = 10^5$

1 lakh users input

`int arr[n][n];`

$n = 10^3$

$10^5 \times 10^5$

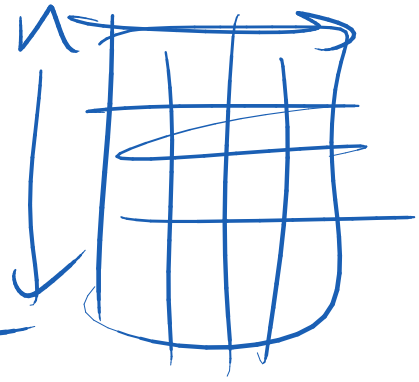
$\sim 10^{10}$

S.C

integers : 10^8

`int arr[n][n][n]`

Memory Exceeded

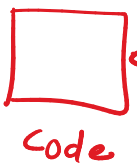


`int arr[n]` $O(n)$ space

`arr[n][n]` $O(n^2)$ space

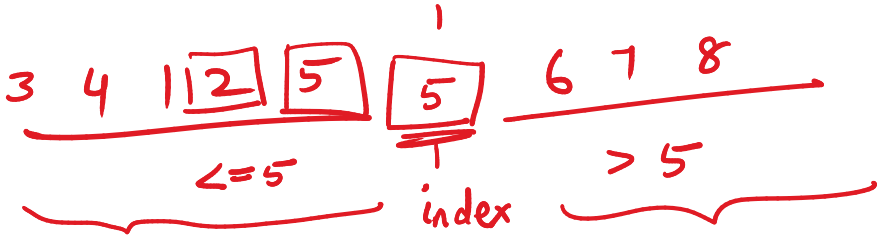
Sorting

Bubble sort	~ 450 - 500 s	$O(n^2)$
$O(n)$ extra space Merge sort	~ 0.4 s	$O(n \log n)$
— no extra space — Quick sort	~ 0.4 s - 0.5 s	



Q. [3 4 1 6 5 2 7 8 5] — space: arr | sort

1. pivot ⇒ last element of the array. pivot = 5
2. Put pivot at its right position | Partition.

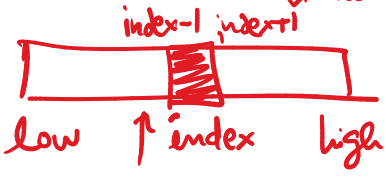


3. Quicksort Quicksort

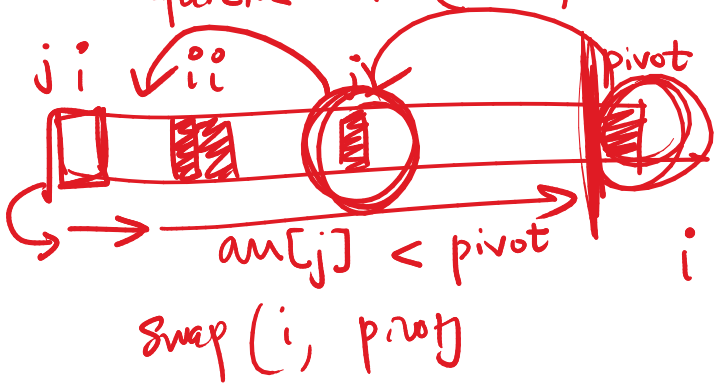
1 2 3 4 5 5 6 7 8

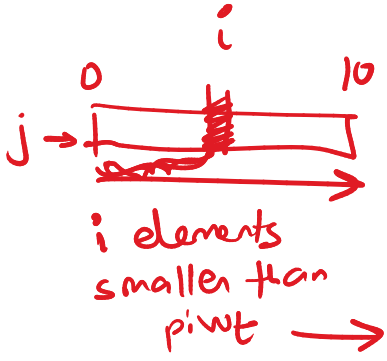
```
quick_sort (arr, low, high):
    if low < high :
        p_index = partition (arr, low, high)   *
        quick_sort (arr, low, p_index - 1)
        quick_sort (arr, p_index + 1, high)
```

inductive manner



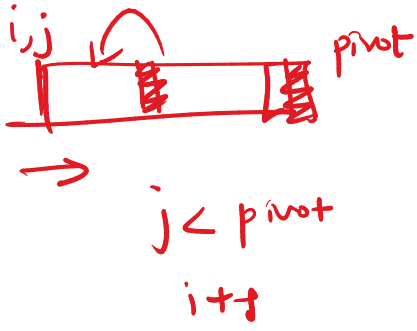
smaller

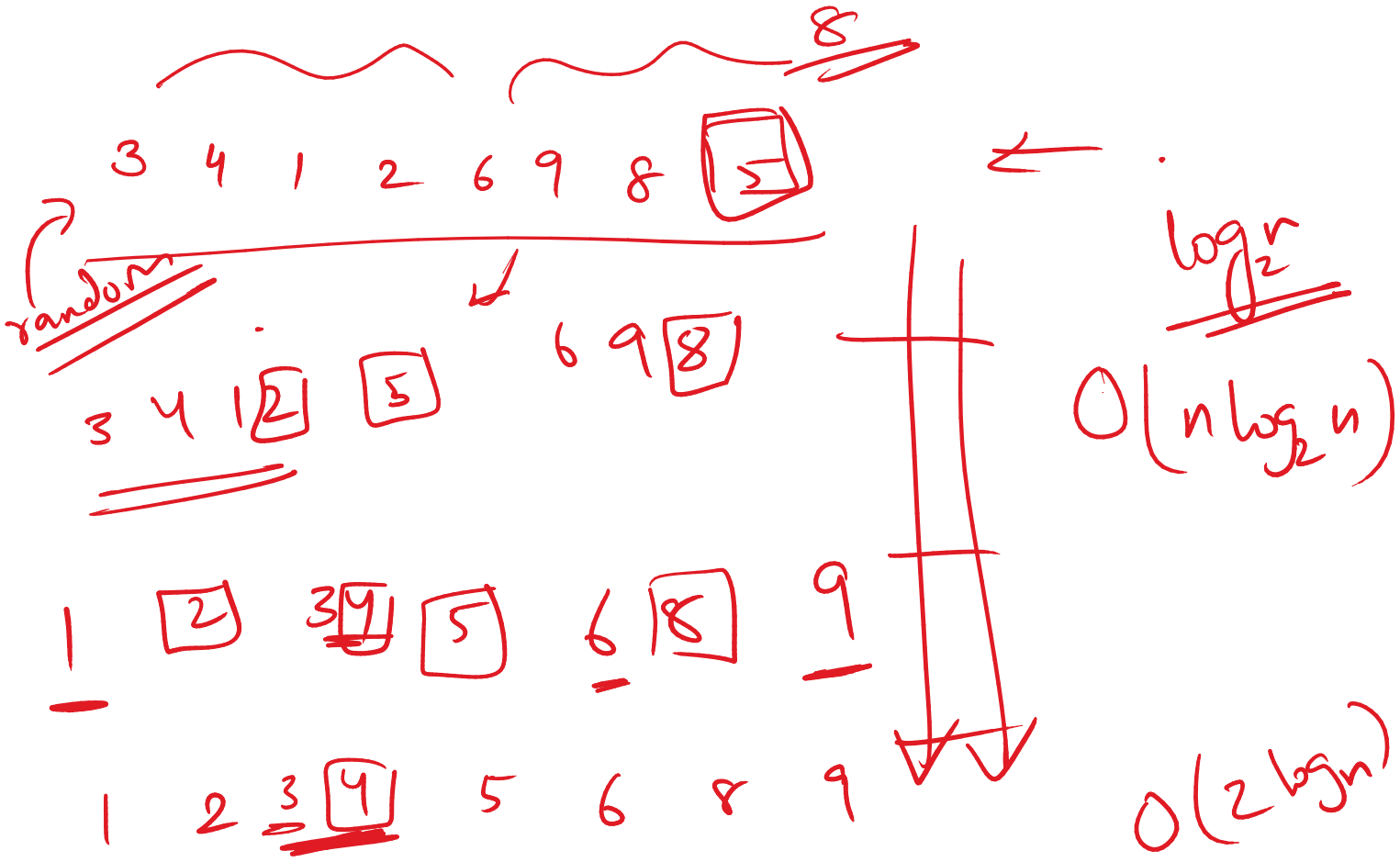




Recursion
- Base
- Decomposition
- Recomposition ✗

```
partition (arr, low, high)
    pivot = arr[high]
    i = low - 1

    for j in (low, high):
        if arr[j] < pivot:
            i += 1
            swap (arr[i], arr[j]
    swap (arr[i+1], arr[high])
    return i+1
```





1. Merge (stable) vs. Quick (unstable)

Stability of a sorting algo. → How to resolve tiebreaks?

[{sagar : 12}, {hrishi : 18}, {hari : 16}, {vishnu : 14}, {priti : 16}]

→ [{sagar : 12}, {vishnu : 14}, {hari : 16}, {priti : 16}, {hrishi : 18}]

Sorted? Yes

STABLE

[sagar : 12, vishnu : 14, priti : 16, hari : 16, hrish. : 18]

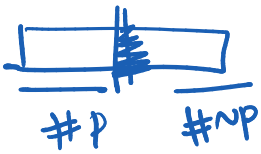
Sorted? Yes

ranking →

UNSTABLE

variants :

partition(arr, low, high)



partition

O(n)

low = i - 1

for j in range (low, high):

if arr[j] != 0 : cond^n is true left

i += 1

swap (arr[j], arr[i])

+ve -ve

even odd

non zero zeros

swap (arr[i+1], arr[high])

return i+1

#1 :

#2

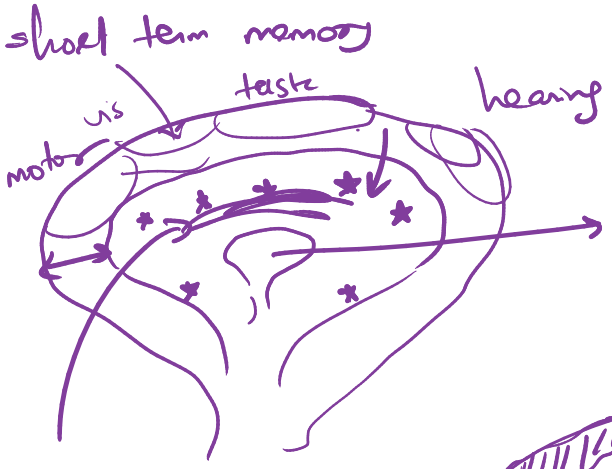
#3

Push zeros to the end

[0 1 0 2 3 0 4 0 0 0]

↳ [1 2 3 4 0 0 0 0 0 0]

Brain

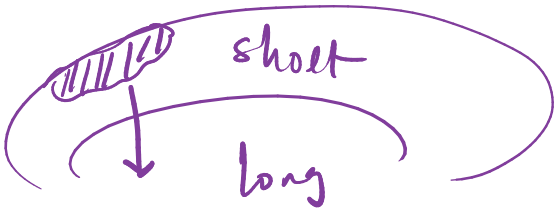


thin

till 15 years

30 years

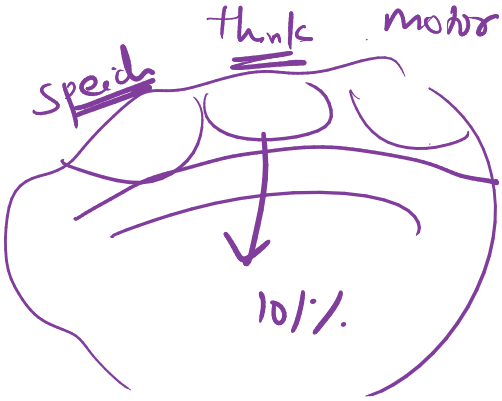
long term memory



Time for which info is on short-

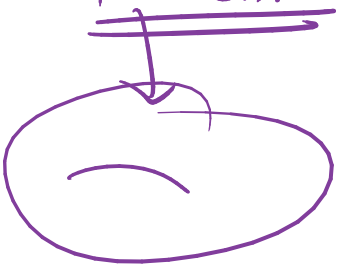
Listen.

Writing



1. listen
2. think
3. hand movement
4. Feedback

4 chunks



1. Inner codes { — — — — }

Easy 20-30

Sorting : Custom sort.

sorting → merge, quick.

Use the in-built sort f^n. sort( ). sort

Sorting ? → ["9", "81", "17", "56", "28", "98"]

Q.1 Arrange these numbers in such a way when concatenated they form largest possible number ?

98117562898 — 998815628 17 — largest possible

How ? Tie ? 9 98 . 998 → 989

Dictionary — m

lexicographic sorting.

m (roman 50) 9 98

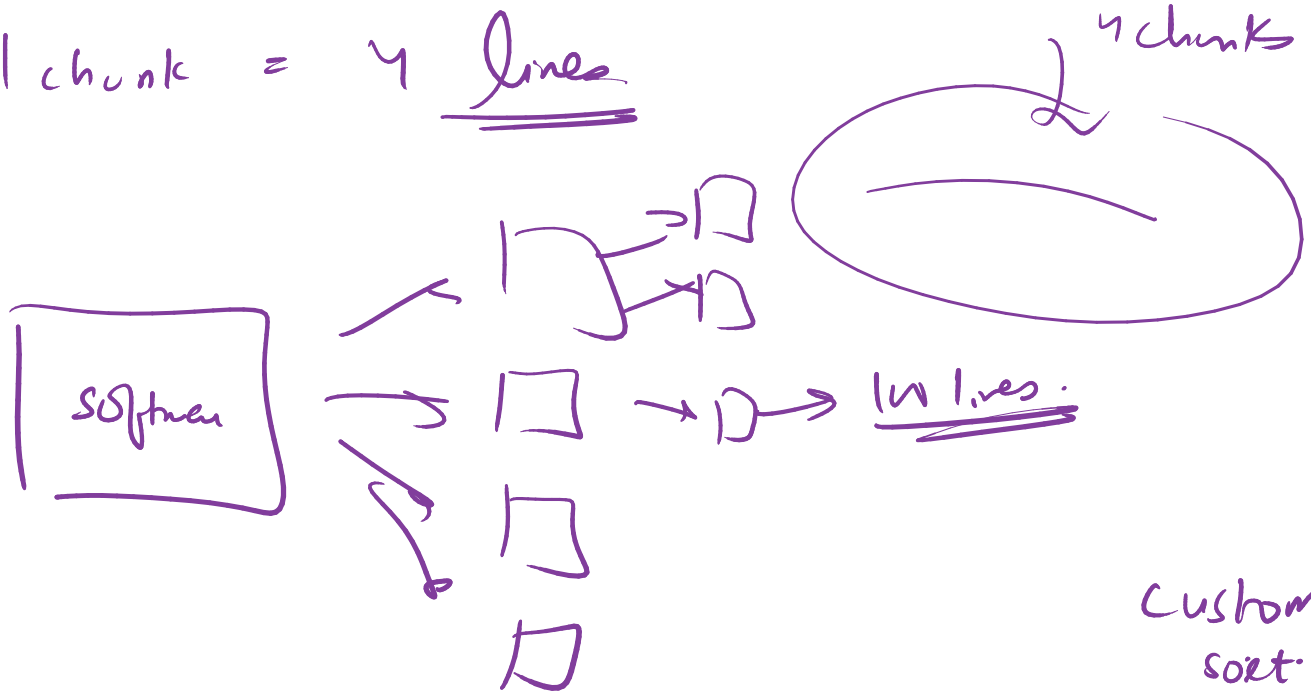
ma

maa

mas

mau

1 chunk = 4 lines



Custom sort.

1. Sort numbers as if they are strings.
2. Sort strings as if they are numbers.
3. Sort strings as strings only but their variation

Key: Arrange / Sort ⇒ doesn't need to be ascending descending !!

"My name is Sagar and I am studying at HeyCoach"
(2) (4) (2) (5) (3) (1) (2)

Smaller sized words 1st.
Tie ⇒ 1st word should appear 1st in input.
(stability)

"my is am" ⇒ "2"

⇒ I my is am at and name Sagar studying HeyCoach.

Q.3 ["apple" "berry", "cherry", "maple", "marry", "ample", "google", "fast", "last", "sample"].

Arrange these words so that they form groups of rhyming words.

["apple" "maple", "ample", "sample", "google", "fast" "last" "berry" "cherry", "marry"]

Reverse: elppa elpam elpma elpmas

sorting!!

C++ / Java

Sort(start, end, how)
how → compare*
start, end → ascending values

C++ → not stable
quick + insertion

Java → merge+ insertion
stable

compare(a, b){
// Transformation on a & b
if (a < b) return True
else return False;
}

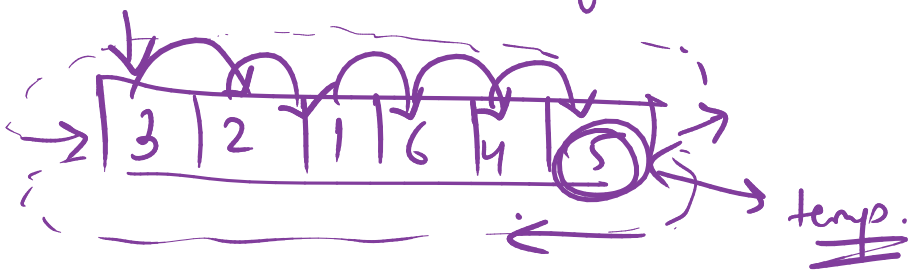
Python

arr.sort(how)
how → compare

Py: not stable.

Compare(a):
// basis on which you want to sort
return len(a)
a.reverse()

Solving various kinds of problems : Arrays.



operations.

1. Rotations.

→ right rotation

← left rotation

1st: [5] 3 2 1 6 4

2nd: 4 5 3 2 1 6

2 1 6 4 5 3

TC. ? O(n) to perform 1 rotation.

If k rotations. ? O(nk)

k = n/2

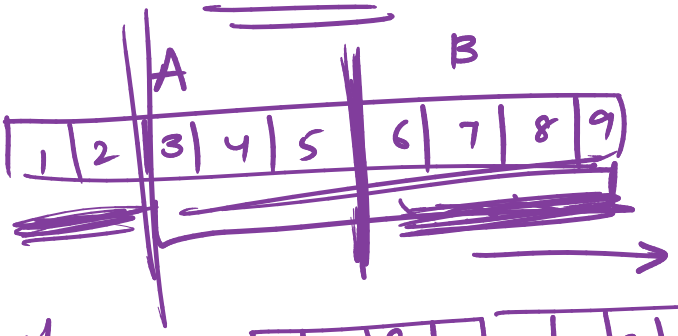
O(n²)

k=7

Can we do better? Yes !!

Reversal algo

21



n=9

k=4

Reverse A

(n-k)

5 4 3 2 1 6 7 8 9

o/p: 6 7 8 9 1 2 3 4 5

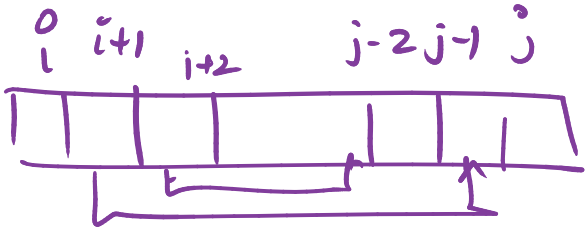
Reverse B O(k)

5 4 3 2 1 9 8 7 6

Reverse entire AB. O(n)

6 7 8 9 1 2 3 4 5

Reversing : O(n)



O(2n)

O(n)

Original logic : $O(nk)$ $k \sim n/2$

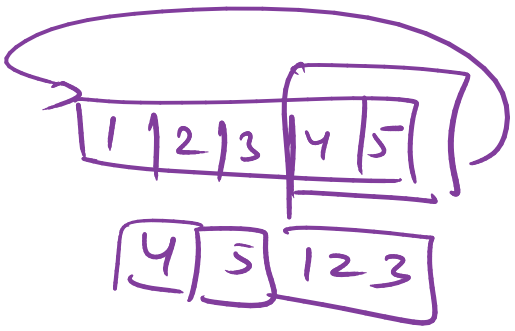
Reversal algo : $O(n)$ time, $O(1)$ space

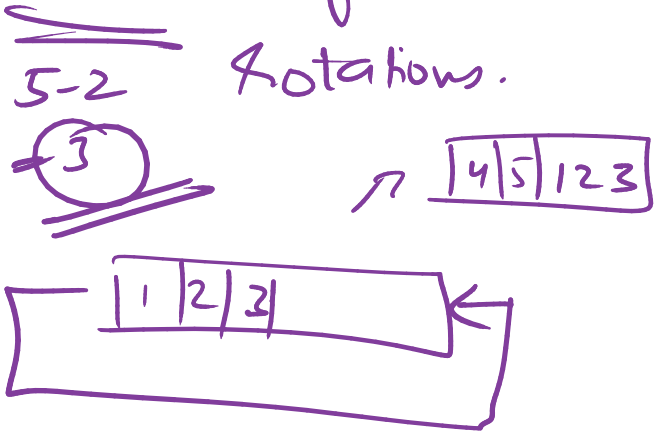
#1: You can't add elements in the start of the array.

$K > n$ $K \% n = k$

2 i right rotations = $n-i$ left rotations.

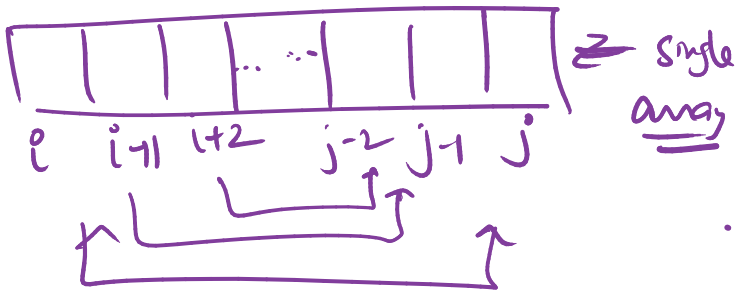
$5-2$





(2) Reverse

Hint: 2 pointer method.

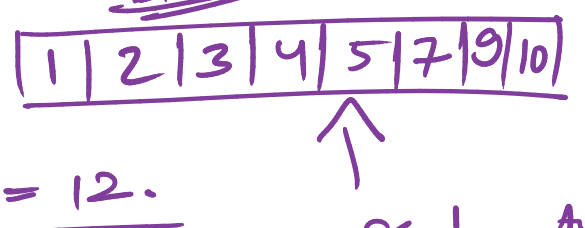


PS1: Two sum problem

2 elements / pair

Sum = 12.

sorted



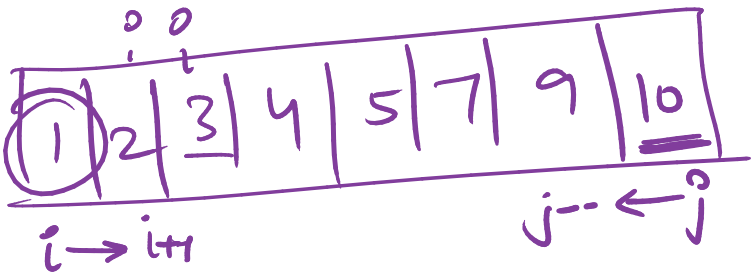
Brute force : from all pairs & check the sum.

1, 2	2, 3	3, 4
1, 3	2, 4	3, 5
1, 4	2, 5	3, 7
1, 5	2, 7	3, 9
1, 7	2, 9	3, 10
1, 9	2, 10	
1, 10		

$n-1 + n-2 + n-3 + \dots + 3 + 2 + 1 = \frac{n(n-1)}{2}$

Checking all pairs of an array $\sim O(n^2)$

Two pointer (1, 7)




```
if arr[i] + arr[j] == sum:
    print (i,j)   i++
                < sum :
i++
                > sum :
```

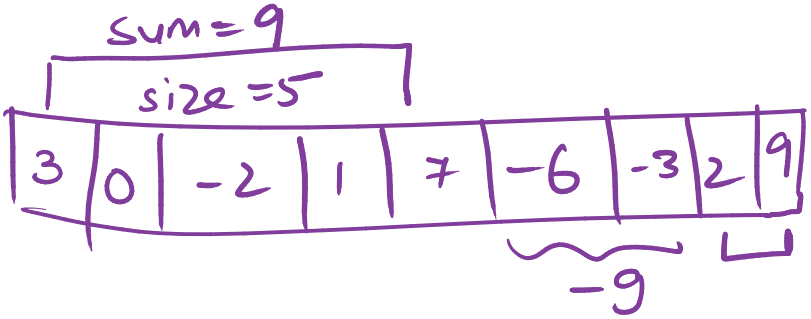
i < j

You shouldn't assume : 2 pointers works only on sorted arrays!!

X

2nd example ⟹ Kadane's algorithm.

Q.



+ve / -ve

sum=11
size=2

max sum of any subarray.

Point: 11

9+2=11
size=n

2 pointers

max_sum max_sum_so_far

I, J

arr

max_sum=0

max_sum_so_far=0

3	0	-2	1	7	-6	-3	2	9
i	i	i	i	i	i	i	i	j

max_sum_so_far += arr[j] j++

MS	MSSF
3	3
3	3
3	1
3	2
9	9
9	3
9	0
9	2
11	11

< -ve

max_sum_so_far = 0

8 times

I & J

ans 11

Kadane's algo

2 pointer

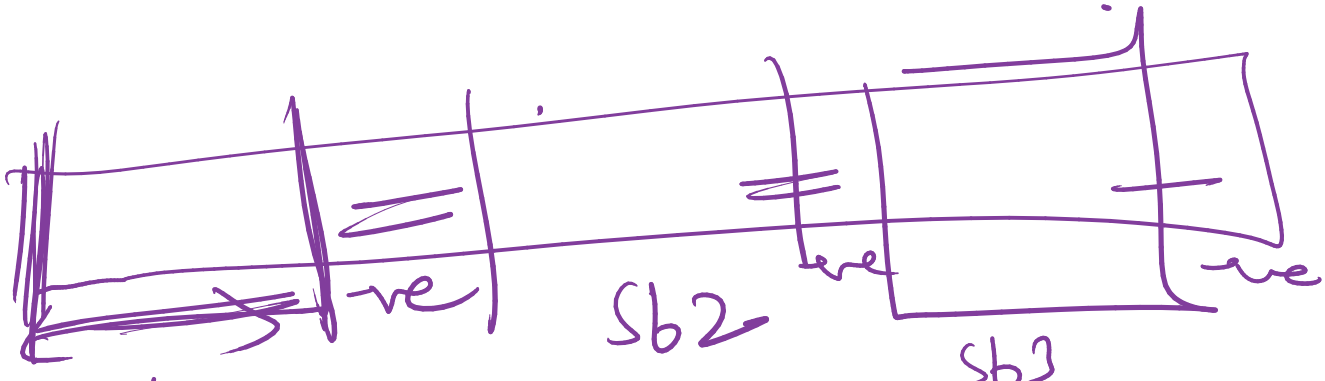
$O(n)$ time

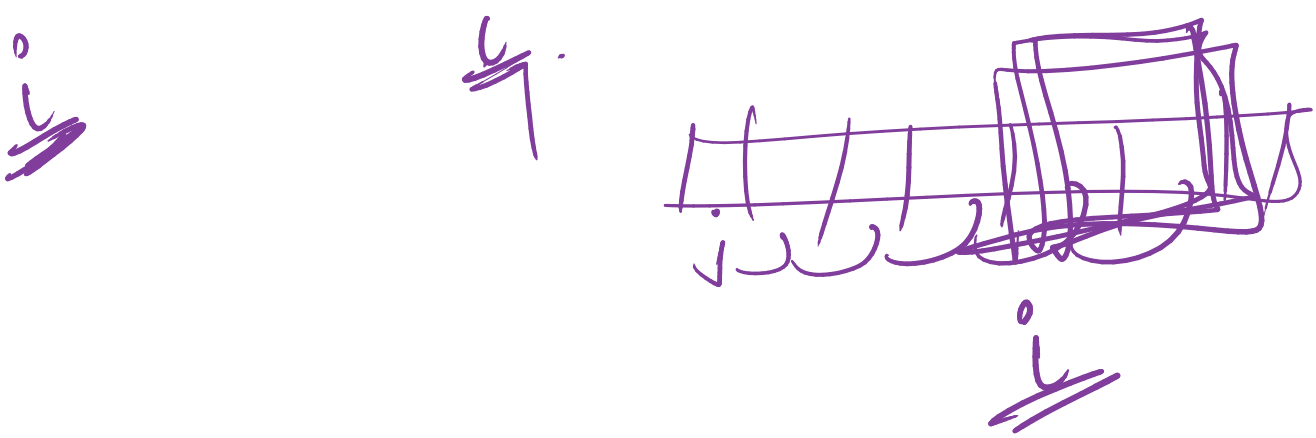
```
int kadane(arr, n)
int mssf, ms, i, j = 0.
while (j < n) {
    mssf += arr[j], j++
    if mssf > ms :
        ms = mssf
    if mssf <= 0 :
        i = j
        mssf = 0
}
return ms
```

B.F. ① generate all subarrays $O(n^2)$

② generate sum of all → $O(n)$ ∴ $O(n^3)$

$O(n)$





int i, j, ms, mssf = 0. s_i, s_j

while (j < n):

mssf += an[j]

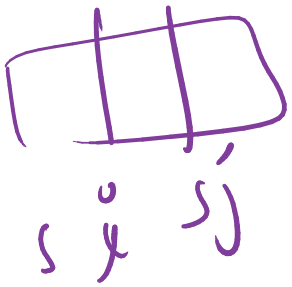
s_i
s_j

if mssf > ms:

ms = mssf

s_i = i

s_j = j



if mssf <= 0:

mssf = 0

j++

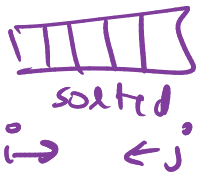
i = j

ms

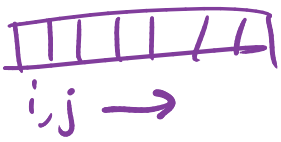
j++

Rotation | RA → reverse i→ ←j (2 ptr method)

2 pointer → 2 sum



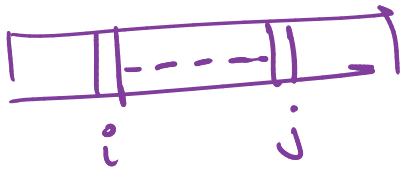
,



Kadane's algo

(pairs)

sliding window



String → substring ⇒ all unique characters.

i j

p w w k e w

i

j. i j

hashset

if str[j] is not set:

0 1

si sj

max_len = 2

pw

si = i
sj = j

- arr[i]

pwwke k

(4) fav. Bulb toggle question

Q. [1 0 1 0 1 1 0 1 0 0 0 1 0 1 0 1 1] n

1 ⇒ bulb is on
0 ⇒ bulb is off

q queries. (li, ri) interval

(3,8) → toggle

[1 0 0 1 0 0 0 1 0 1 0 1 1]

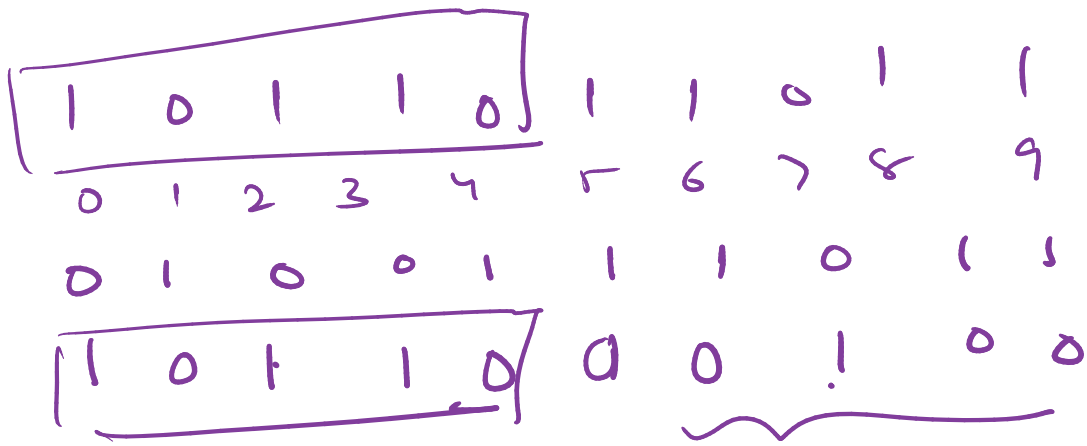
(2,5) (7,9) (3,9) (2,6)

After q, total number of bulbs that are ON!

Brute force & O(q.n)

1,100
1,100
1,100
1,100

Redundant work? → Hell yes!!



10 elems

1-5
1-10

1-5
1-5 6-10

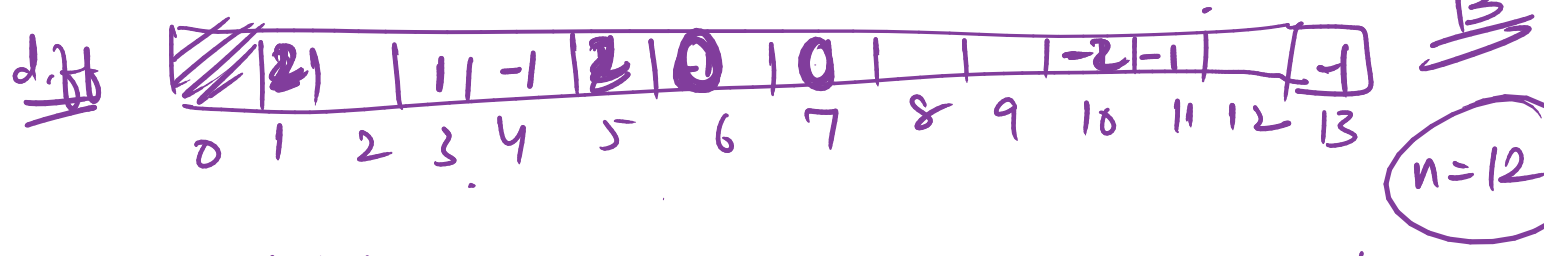
How?

diff_arr = [ ] n+1

For each (l_i, r_i)

diff_arr[l_i] += 1

diff_arr[r_i + 1] -= 1

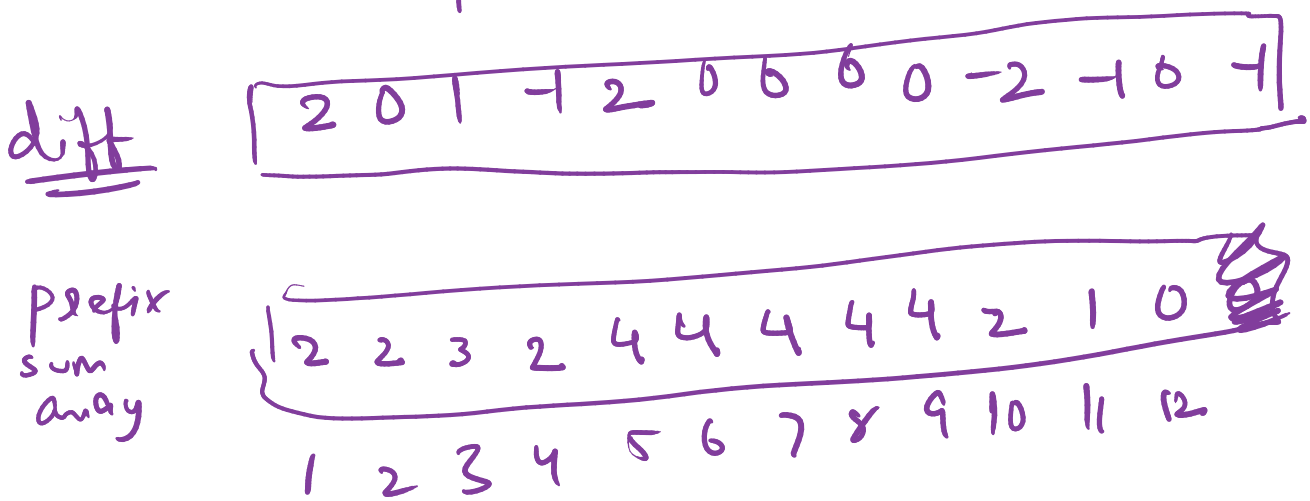


13

n = 12

1-based indexing

ps[i+1] = ps[i] + diff[i+1]



bulb 1 ✓

3, 10 ✓

6-9 ✓

1-3 ✓

7, 9 ✓

5, 6 ✓

5-12 ✓

if ps[i] is even. → don't toggle bulbs

odd → toggle it

for each query } O(q)

diff[l] += 1 diff[r+1] -= 1

prefix sum → O(n)

if ps(i) is even → no toggle → O(n)

toggle → O(qn)

sum → O(n)

O(q+3n) ~ O(q+n)

Method 1 : q queries :

```
for i in range(q):
    li, ri = q[i]
    for j in range(li, ri):
        an[j] = 1 - an[j]
```

T.C. $O(q \cdot n)$ → optimization

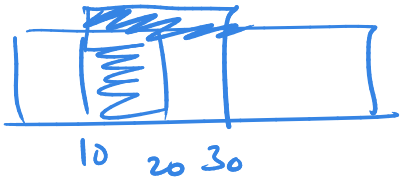
Toggle $O(q \cdot n)$

```
print(sum(an))
```

↳ $O(n)$

Method 2 : Optimization ⇒ Am I doing redundant work?

Yes !!



100

Duplicate toggling ←
- 10 - 20
- 10 - 30
- 10 - 20 ⇒ original
- 5 - 50

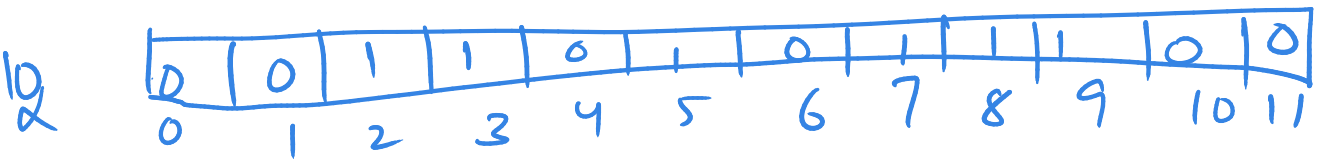
Idea: I want to keep track of ranges / intervals.

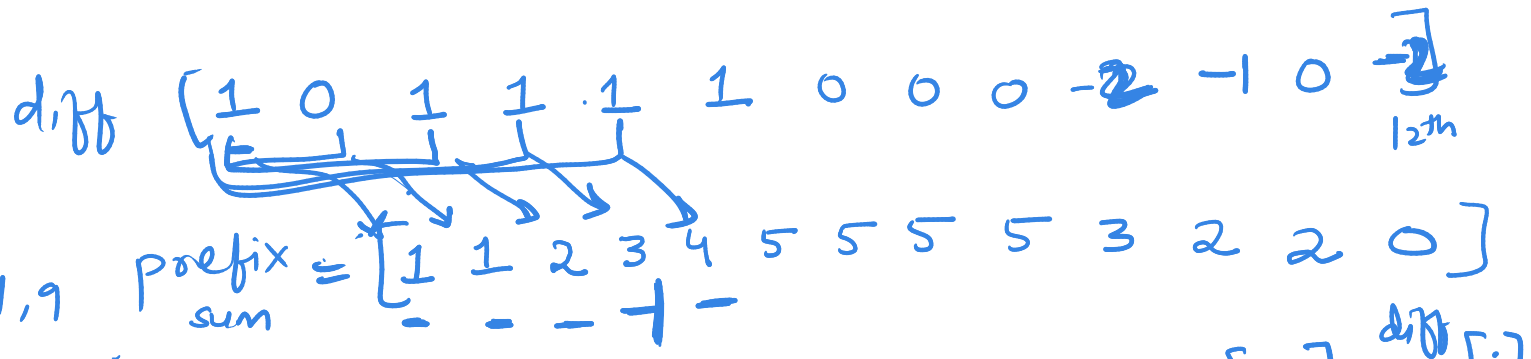
I will toggle only at the end !!

bulls = [ ]n

diff = [$+1$ at l^{ith} , -1 at $r^{ith}+1$]$_{n+1}$

l_i, r_i





4,9
5,11
3,8
0,11
2,8

ps[i] = sum of initial i elements.

ps[i] = ps[i-1] + diff[i]

ith bulb is toggled ps[i] times.

if ps[i] % 2 != 0 :
bulbs[i] = 1 - bulbs[i]

sum(bulbs) ⟹ ans.

Logic:

```
for i in range(q):
    li, ri = q[i]
    diff[li] += 1
    diff[ri+1] -= 1
```

TC: O(q)

diff = [0 ....] n+1
ps = [0 ....] n+1

★ ps[0] = diff[0]

```
for i in range(1, n+1):
    ps[i] = ps[i-1] + diff[i]
```

O(n)

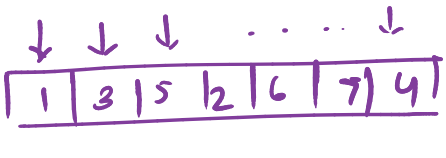
```
for i in range(n):
    if ps[i] % 2 != 0:
        toggle(bulb[i])
```

O(n)

sum(bulbs)
ans.

Searching ps: list of elements, whether a particular element key → present or not !!

1) linear search



key = 4 ? Yes!!

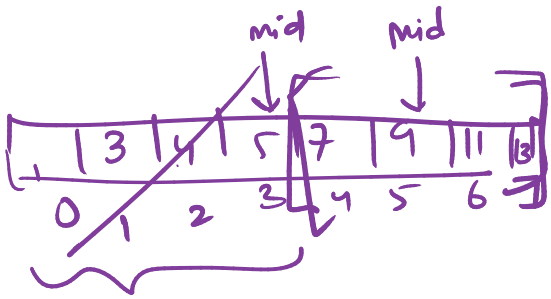
```
for (auto x : arr)
    if x == key
        return True
return False
```

O(n) time

O(1) space.

2) Binary search

assmp: Array is sorted.



key = 9

Yes !!

8

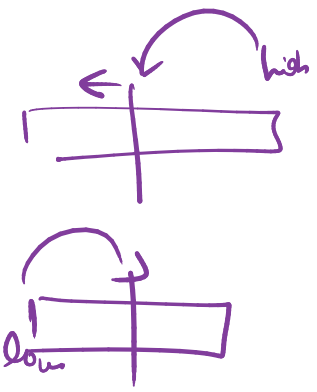
Every single iteration

n → n/2 → n/4 → ⋮ → 1

$O(\log_2 n)$

```
while (low <= high):
    mid = (low + high)/2
    if arr[mid] == key:
        return mid
    else if arr[mid] > key:
        high = mid - 1
    else:
        low = mid + 1
return -1
```

[low ↓mid high]



Array :	Linear	Binary
unsorted	O(n)/per search	O(nlogn) + O(log n)/search
sorted	O(n)/search	O(log n)/search

2 cases :

① 5 elements — N 5 → ~~ → O(n log n)

9 searches.

* search x

3 elements — 8 → ~~ → O(n log n)

* search x

10 elements — 18 → ~~ → O(n log n)

* search x

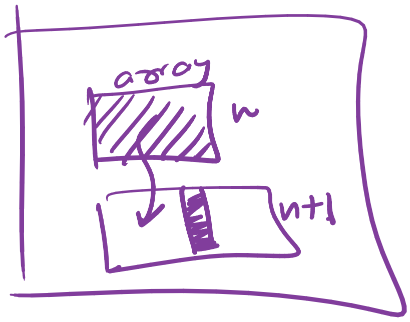
Binary $O(n \log n \cdot q) + O(q \cdot \log n) \Rightarrow$ Total time.

$O(n \log n + \log n)$ / search $\sim O(n \log n)$

linear $O(nq)$ $O(n)$ / search $\sim O(n)$

n+1

O(n) work !!



② all values are given in advance.

q queries. $O(n \log n)$ 1 time.

BS: $O\left(\frac{n \log n}{q} + \log n\right)$ / search $O(\log n \cdot q)$ searches

linear : $O(nq) \Rightarrow O(n)$ / search

B.S winner !!

BS concept:

searches ↑↑ 10^8 billion 10^6 million

massive

linear $O(n)$

$n: 10^6$

$q: 10^8$

$\Rightarrow$

P.S all data is in advance

$O(n \log n) + O(q \log n)$

B.S. is slow!! $O(\log n)$ $O(q \log n)$ 10^8 ops second.

Can I do it faster ?? Yes!!

$O(1)$: Constant time.

Computers don't need to perform searches. They already know all the elements they have memorized.

Hashing

hashing → integers x strings

a→1	j→10	u 21
b→2	k→11	v 22
c→3	l→12	w 23
d→4	m→13	x 24
e→5	n→14	y 25
f→6	o→15	z 26
g→7	p→16	
h→8	q→17	
i→9	r 18	
	s 19	
	t 20	

"sagar"

s+a+g+a+r

19+1+7+1+18 (25)

20 1

(46) % 8 ⇒ 0 to 7

6

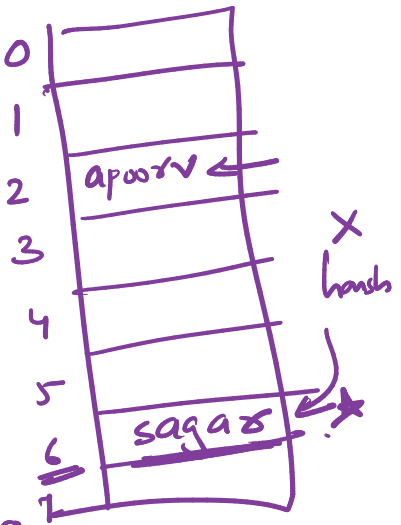
"harsh"

8+1+18+19+8

9 27 27

54 % 8 ⇒ (6)

array / table



elements = 8

search ("sagar") → 46 % 8 = (6)

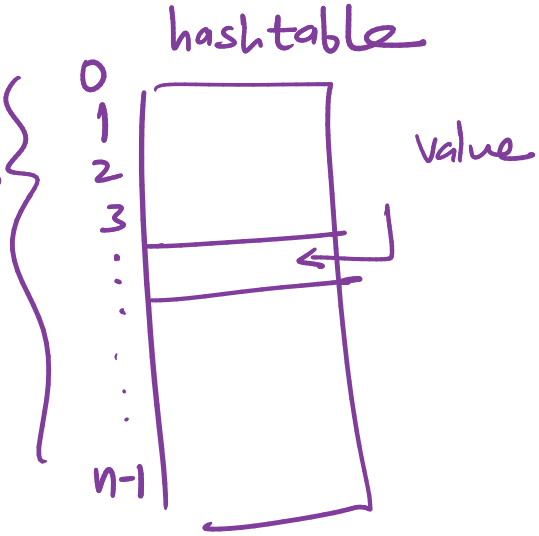
search ("harsh") ⇒ (6)

$O(1)$ search

input → f() → index

hash function

hash indices value.



6 No!! aaaaa sagar hassh

Problem : ① Collisions

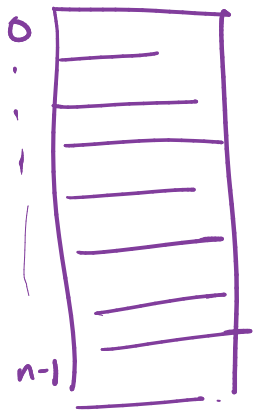
How ?? ② Table size

inp1 → hash() → same index
inp2 →

data can come sequentially.

. sagar ragas

Hashing : size = n values. 0 ... n-1



1st string. ⇒ p(collision) = 0

2nd string p(collision) = $\frac{1}{n}$

3rd string p(collision) = $\frac{2}{n}$

Till #inputs so that collisions are occasional ?

n/2 elements p(collision) ~ 50%

half of table is filled, collisions won't be occasional.

#logic ★ : whenever half of table is filled, +1 element,

↳ 2* size of table & then add this element.

1 $\xrightarrow{2x}$ 2 $\xrightarrow{2x}$ 4 → 8 → 16 .. $\xrightarrow{2x}$ 32 $\xrightarrow{2x}$ _

1st 2nd 3rd 4th 5th 6th 9th 17th

1/1 2/2 3/4 4/8 5/16 6/16 9/32

7/16 10/32

8/16 11/32

12/32

13

14

15

16

more preferred

2 → 4 → 8 → 16

1/2 2/4 3/8 4/8 5/16

50% 50% <50

$\frac{\#\text{elements}}{\text{size}} < \frac{1}{2}$

② hash()

sagar ragas ⇒ 100% collision.

↳ O(1) v.v.fast / CPU → v.v.simple

↳ order sensitivity

↳ char → ascii
unicode }

%n ⇒ prime

1 + 2 ⇒ Tried to reduce the possibility of collision. Some chance

Resolve collisions ① Probing
② chaining (use LL)

hash(inp) ⇒ (x)

2nd hash ⇒ (a.x)%n ⇒ prime

x'

a → param

0

x

value!!

n-1

$(ax^2+bx)\%n$

a, b → params

x'' ⇒ $ax''^2 + bx''$

x' p(x) x''

x p(x'')

Ensure:

6 → 4 → 1 → 3

0-7

2

5 ← 0 ← 7

8 elements

0-7

p(x)

Group Theory ⇒ Abstract Algebra

③ a) size of table ⇒ prime

b) params of prob f^n ⇒ prime

prime

size of table is prime collisions ↓

a % p

prime

b % p

prime

p doesn't have common factors with a & b

p = 12

2, 3, 4, 6

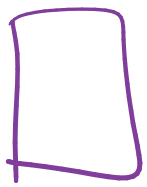
Hashtable

① 2 → 5 → 11 → 23

prime

elements

elements

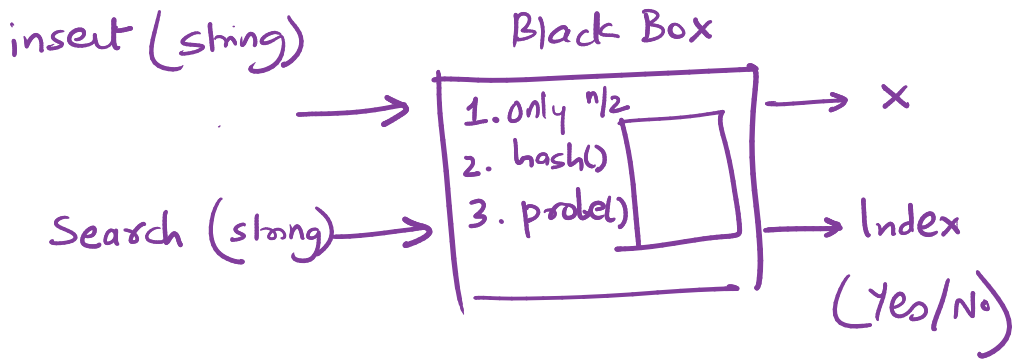


50%

② hash() + probing → params are prime

③

Time complexity



1. Insert(s) if occupancy > 0.5
   - ↳ double the size & size as next prime
   - ↳ rehash the previous values & put them back in appropriate cells.

O(1) time

hash(s) → x

if x gives collision: } ⇒ x such that no collision.
x' → probe(x)

index x ⇒ place(s) | table[x] = "s".

2. search(s)

hash(s) ⇒ x

table[x] == s if not

it's not on x, it might be on x'.

x' = probe(x)

table[x'] != s if it's null

if empty → return No.

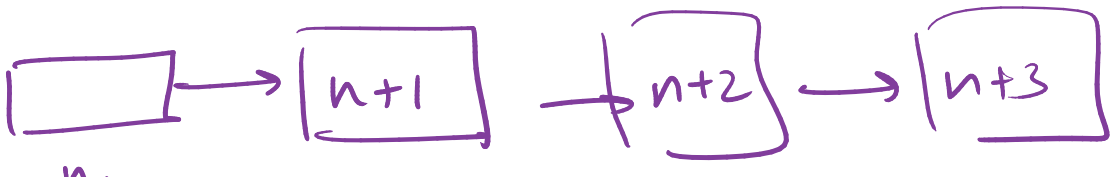
Keep probing until you find s or empty cell.

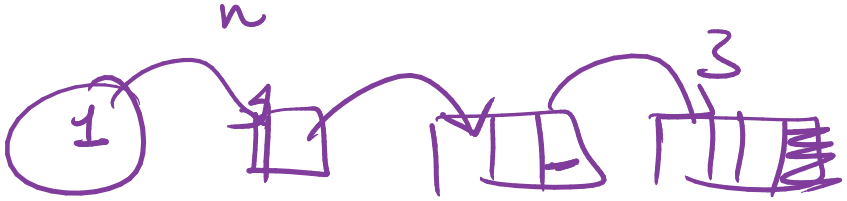
O(1) time

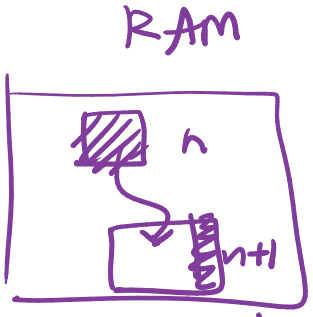
Insertion: 1 element $\xrightarrow{\text{ht}}$ O(1) time

n elements $\longrightarrow$ O(n) time ~ array

O(1)





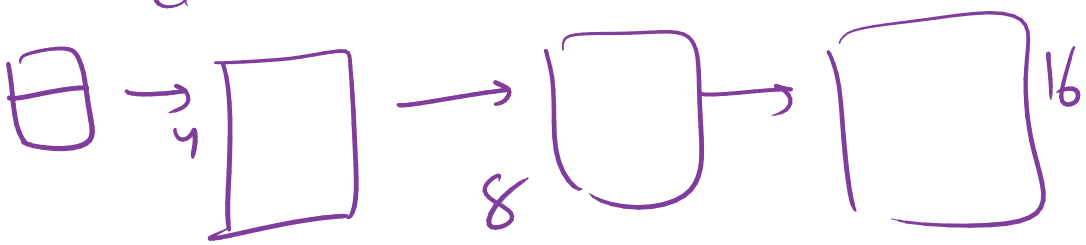


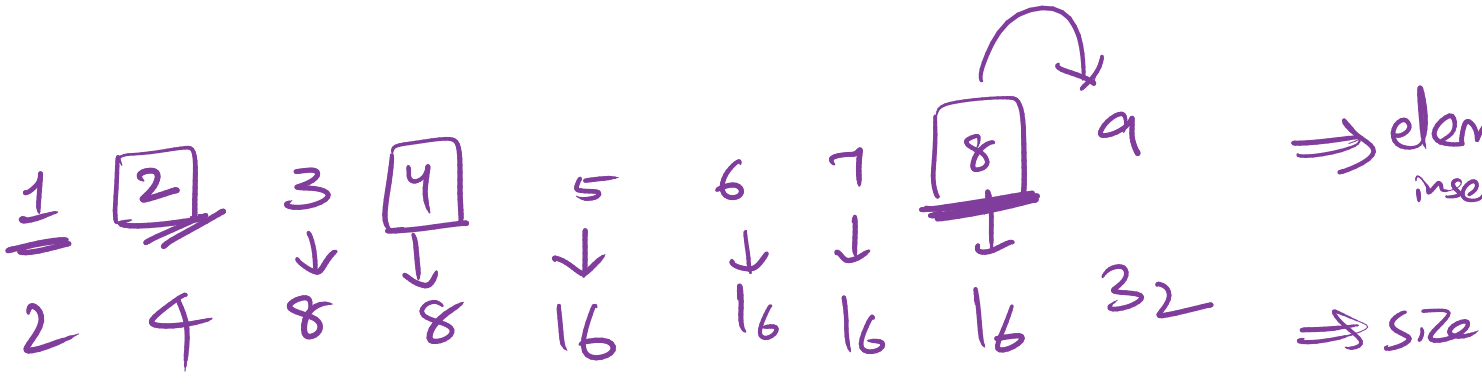
$1 + 2 + 3 + 4 + \cdots + n-1 \Rightarrow$ n

Insert n elements

Total work to insert elements $= \frac{n(n-1)}{2} \sim O(n^2)$

Insert 1 element $= O(n)$ / element.





$n + \frac{n}{2} + \frac{n}{4} + \cdots 8 + 4 + 2 + 1$

$$1 + 2 + 4 + 8 + \cdots + \frac{n}{4} + \frac{n}{2} + n$$

$$n\left(\frac{1}{n} + \frac{2}{n} + \frac{4}{n} + \frac{8}{n} + \cdots + \frac{1}{4} + \frac{1}{2} + 1\right)$$

$$n\underbrace{\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots \frac{4}{n} + \frac{2}{n} + \frac{1}{n}\right)}_{GP}$$

$$\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots\right)$$

$$S = \frac{a}{1-r} \quad \frac{1}{1-1/2} = 2$$

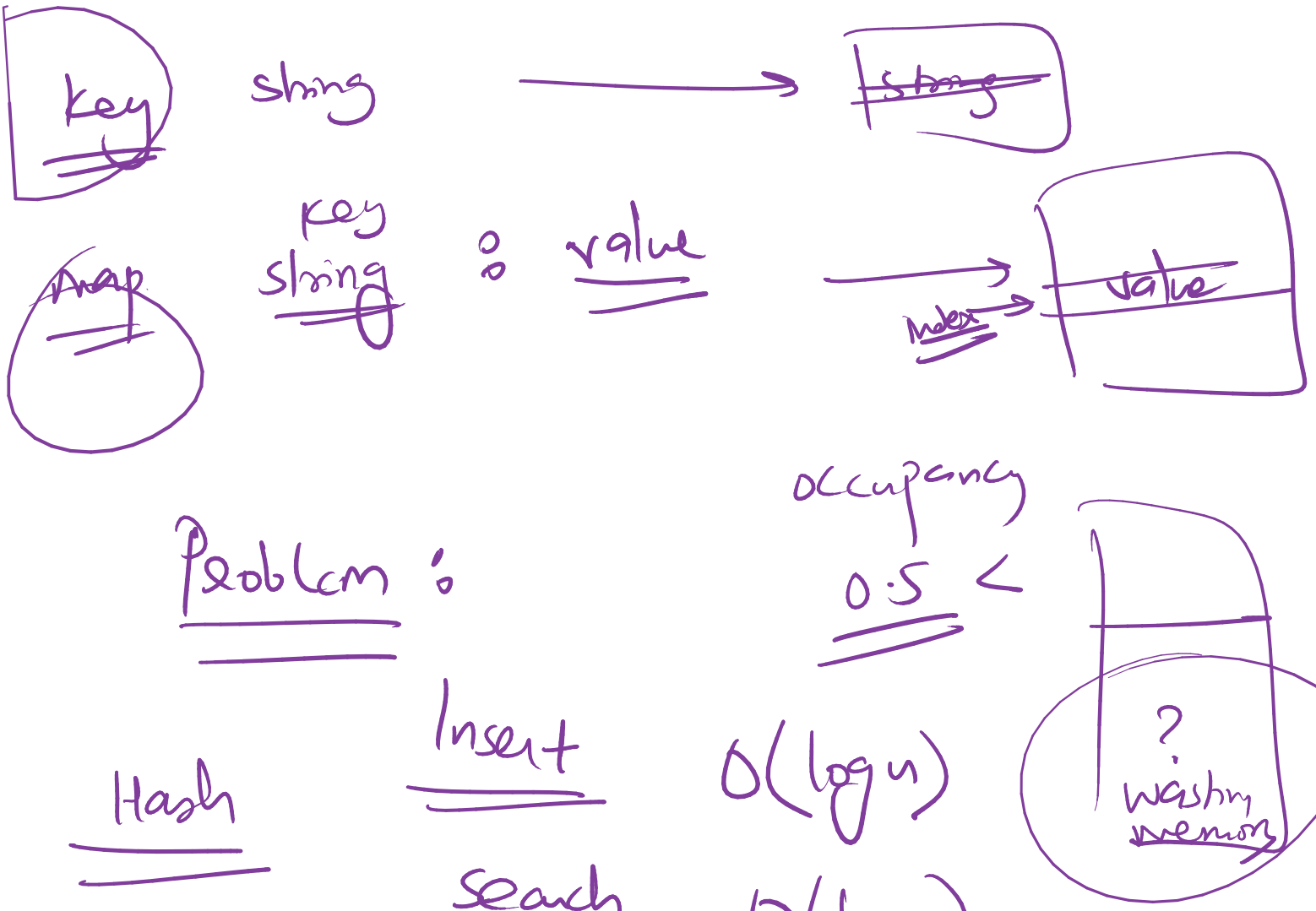
$\sim (2n)$ $O(n)$ work to insert n items.

per item insertion $\Rightarrow O(1)$

string : int

{ Hashtable } $\Rightarrow$ Hashset

$\longrightarrow$ Hashmap



occupancy

Problem : 0.5 <

	Insert	Search	Waste
Hash	O(log n)	O(log n)	O(1)
			O(1)

B.S $O(\log n)$ $\log n$ hash

1

Insert $O(n)$ $O(\log n)$ Insert

Q.1

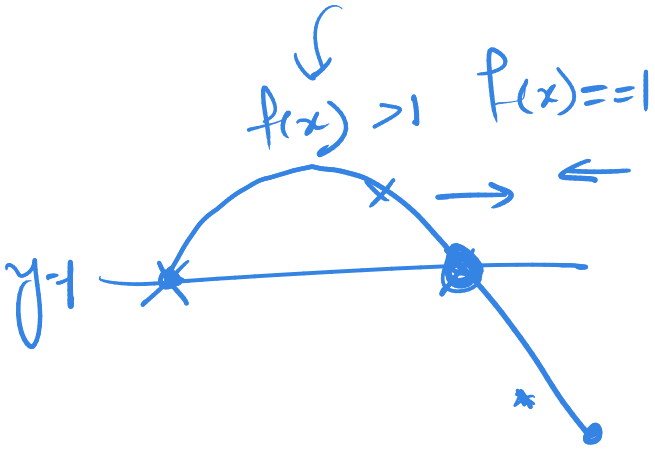
In case of peak element using binary search Suppose my input array is [5,10,15,20]
so at mid say at 10 we can see 10 is definitely not peak as is < 15 but >5
So now we need to move either on left or right For right there is 15 which is > 10 ,
so a possibility is there For left we see 5 < 10 so 5 cannot be candidate here .

But what if I had other element on left Like 1,2,7,... [5,10,15,20] so array is [1,2,7,5,10,15,20]
So in this case we could have got maximum in left side as well right ??as 7 is one of peaks
As there can be multiple peaks ,
so its fine , we know that as 15>10 so we will always have solution on right hand side ,
as slope is increasing .
And even if it keeps on increasing we will get last element as peak at least But is that the only reason ??

Isn't it a bit fuzzy logic here.

https://leetcode.com/problems/find-peak-element/description/

Binary search. I have a number ⇒ 1 - 100
minimum guesses you will make to find the numb



A learner from heyCoch loves finding unique things, so in the next task by heyCoach, he has been assigned to check if in a given array say nums say s is the maximum sum out of all the subarray he can get from an array nums, and p is the maximum product he can get out of all subarrays, his task is to check if s is greater than p or not.

```
if(s>p) return 1;
if(s<p) return 0;
if(s==p) return -1;
```

s == p.

s ⇒ max sum of subarray
DP. ↳ Kadane's algo.

Sample Input:
5

9 12 18 66 94 ← all +ve

p ⇒ max product all subarr

Sample Output:
-1 0

S = entire array
P = entire array
S > P.

Constraints:
The length of the array nums will be between 1 and 10^5.

For all 0<i<nums.length, -10^4<=nums[i]<=10^4.

P > S

product of all elements $a_1\ a_2\ a_3\ \ldots\ a_n$

$$p = \prod_{i=1}^{n} a_i \qquad s = \sum_{i=1}^{n} a_i$$

$\Longrightarrow$

5	-12	9	18	-23

array $\Rightarrow$ ~~signs~~

sign $\rightarrow$ [1 -1 1 -1 -1]

mag $\rightarrow$ [5 12 9 18 23]

$$\log(a_1\ a_2\ a_3) = \log a_1 + \log a_2 + \log a_3$$

$\Longrightarrow$ sum.

i. max

even $\Rightarrow$ [-ve]

Kadane's $\rightarrow$ [log5 log12 log9 log18 log23]

~~$\log^2$~~

a_i —— $\sum a_i b_i$

b_i ——

↑ sum product

log. ×

$f(x)$ ⇒ monotonically increasing if x is increasing.

$x_1 < x_2$

$\hookrightarrow f(x_1) < f(x_2)$

2 5

$\log 2 < \log 5$

$\sqrt{}$ ve numbers

$\sqrt{2} < \sqrt{5}$

$\sqrt[2]{}$ $\sqrt[5]{}$

-ve

-ve

$\sqrt[3]{}$

$f(x)$ ⇒ -ve integers

↳ turns × very low. ÷ — —

↳ sign magnitude

max_sum

max_sum_so_far

Kadane

size × subarray ⇒ sum max.

i j

6+5+4

sub1

9	3	-2	8	-10	6	5	4	-2	1	-5	4

break

sliding window + 2 pointer.

i j

1 ⇒ n subarrays

2 ⇒ n-1 subarrays

3 ⇒ n-2 subarrays

⋮

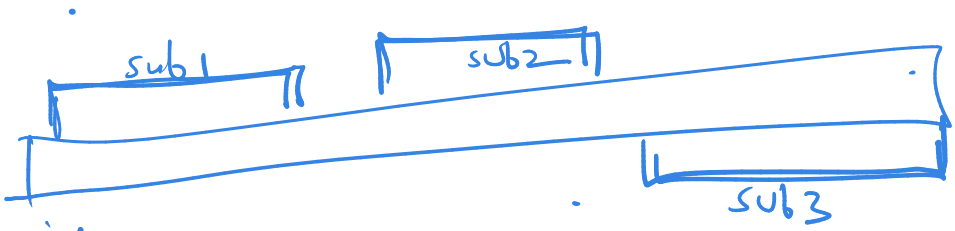
n ⇒ 1 subarray.

All subarrays ⇒ $\frac{n(n+1)}{2} \sim O(n^2)$

3/4/5...

Sum. ⇒ O(n) work. Brute force : $O(n^3)$

Kadane ⇒ O(n) sol^n.



max of all i,j

max_sum ⇒ ans.

sub2 ← max_sum_so_far

-ve

max_sum_so_far ×3

S = max_sum

Bucket Sort

You are the manager at "Dream Wheels" which is a Car Showroom and there are 'n' number of cars available here.
You read an analysis that customers tend to buy expensive rare cars if they come across them after the common cheaper cars.
Your task is to arrange the cars in the decreasing order of their frequency, cars are represented by a string of characters, if two or more cars have the same number of frequencies sort them lexicographically.

Sample Input:

```
ssgysyqa
```

Sample Output:

```
sssyyagq
```

Constraints:

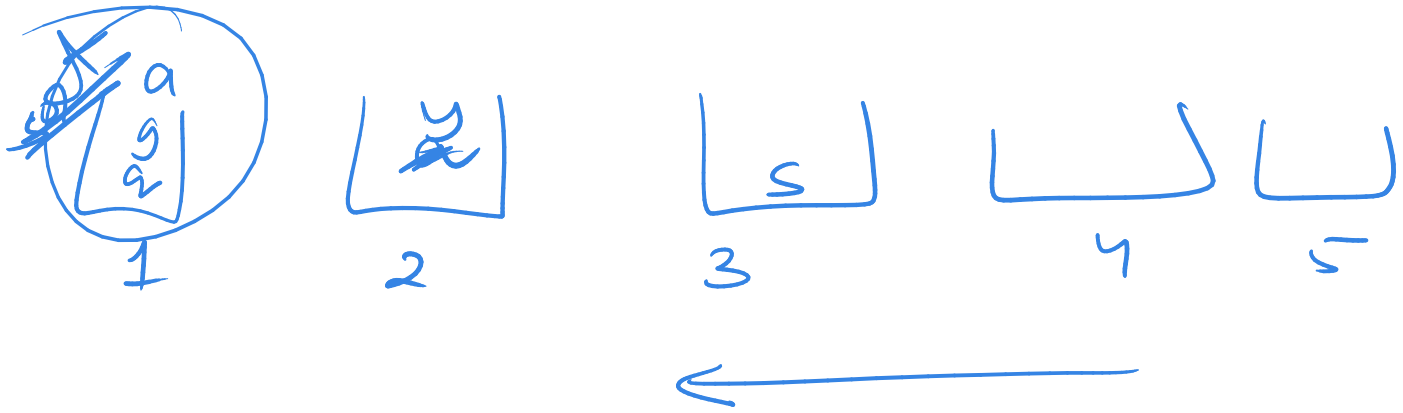
- `1<= n <= 10^5`

`All characters are lowercase only and are English alphabets.`

s → 3 g → 1 y → 2 → q → 1 → a → 1

Common → rare

sssyyagq



s y aggq

n characters $\Rightarrow$ $O(n \log n)$ $\Rightarrow$ $O(n)$

Sort map: key: characters

[(a,1), (g,1), (q,1), (s,3), (y,2)]

a : 1
g : 1
q : 1
s : 3
y : 2

C++ map

py value map

sort it on the 2nd element

↳ if tie, resolve the tie on 1st element.

Question 1
Mangoes and Pineapples

There are 'n' fruits lying in a row on a long table. Each of these fruits is either a 'mango' denoted with the letter 'M' or a 'pineapple' denoted by the letter 'P'. There is at least one of each on the table.

You and your friend want to divide the fruits on the table: you will take a prefix of this row (several leftmost items), and the friend will take the rest. However, there are several restrictions:

1. Each person should have at least one item.

2. The number of your mangoes should differ from the number of your friend's mangoes.

3. The number of your pineapples should differ from the number of your friend's pineapples.

Find the lowest prefix selecting which division goes with these restrictions and if there is no answer print -1.

Input1:-

The first line contains one integer n. The number of fruits on the table.

The second line contains a string of length n consisting of letters 'M' and 'P'.

Output1:-
Return the lowest possible integer 'ans' such that, if you take 'ans' number of leftmost items and your friend takes the remaining (n−ans) fruits, each of you and your friend get at least one fruit, your number of mangoes is different from your friend's, and your number of pineapples is different from your friend's. If there are no possible answers, return the number −1.

Adhoc question

Just try to code what is asked.

Sample Input:

3

PMP

Sample output:
1

Sample Input 2:-

4

PMPP

Sample Output 2:

1

(Note that in output 2, the number 2 also satisfies the conditions but it is not the lowest possible so the answer is one.)

Constraints:

2<=n<=200

```
func lowest_prefix (string s):
    mangoes = s.count ("M")
    pineapples = s.count ("P")

    if mangoes == 0 or pineapples == 0:
        return -1

    my_mangoes = 0
    my_pineapples = 0

    for i in (0, n-1):
        if s[i] == 'M':
            my_mangoes += 1
        else
            my_pineapples += 1

    // adhoc cond^n.
    if (my_mangoes != 0 and my_pineapple != 0)
      and
        my_mangoes != mangoes - my_mangoes
      and
        my_pineapples != pineapples - my_pa.
                return i+1

    return -1
```

C++

```
map<char, int> m
for char c : s
    m[c]++
```

"MP" M | P

"PM" P | M

1

Median of 2 sorted arrays.

m: 1 3 7 9 11 15 18 } unequal size.

n: 2 9 13 21 25

$O(m+n)$

if we merged them, median

1 2 3 7 9 (9) 11 13 15 18 21 ~~25~~

$m+n$

2 middle even

even: median = avg (2 middle elements)
= $\frac{9+11}{2}$ = 10 ans.

odd: median = 9

1 middle

$m+n \Rightarrow$ odd

arr 1: 1 3 | 7 9 | 11 15 18 — 7

arr 2: 2 9 | 13 21 | 25 — 5

$m+n$ = 12 (1st left + 1 elem)

half = 6.

[1 3 2 9 13 21] 1st partition

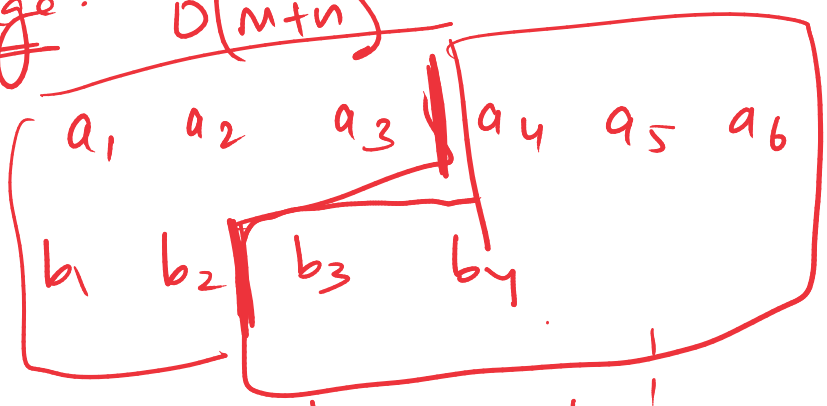
[7 9 11 15 18 25] 2nd partition ⟹ 6

[1 3 7 9 2 9] 1st.

[11 15 18 13 21 25] 2nd

all smaller or at max equal to all element of 2nd partition

merge: $O(m+n)$

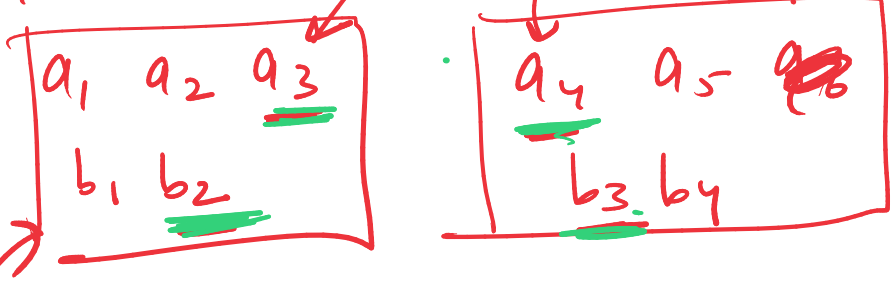


$\Rightarrow$ 6 } 10
$\Rightarrow$ 4 } 5

a_1 a_2 b_1 a_3 b_2 a_4 a_5 a_6 b_3 b_4

$\Rightarrow$ merged sorted.

5 bigger | smaller 5



smaller or equal to Right partition

median = avg (b_2, a_4)

$\Rightarrow \frac{b_2+a_4}{2}$

$\frac{a_3+b_3}{2}$

a_1 a_2 b_1 b_2 a_3 b_3 a_4 a_5 a_6 b_4

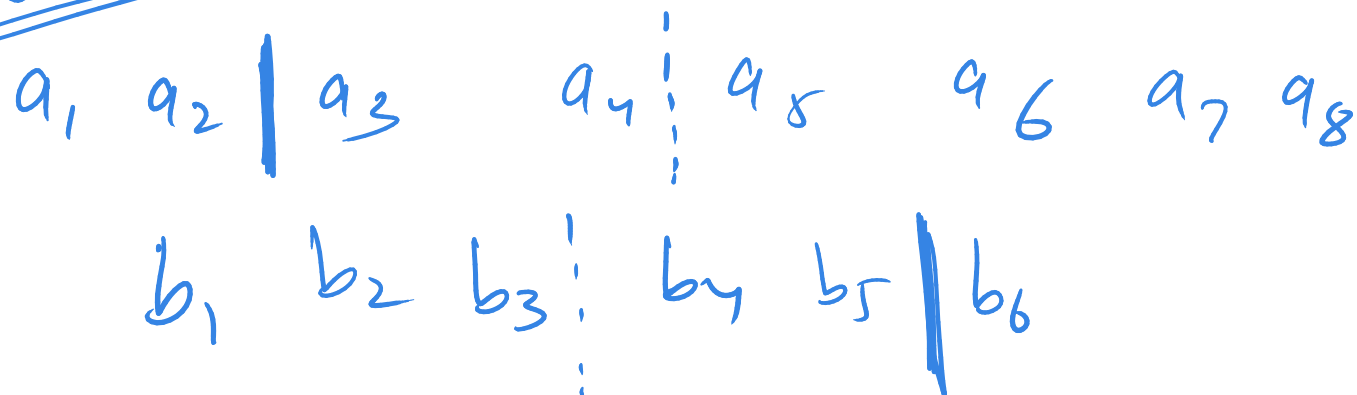
$a_i < b_j$

$b_{i'} < a_{j'}$

even. $\Rightarrow$ median = $\max(a_3, b_2) + \min(a_4, b_3)/2$

odd $\Rightarrow$ Median = $\max(a_3, b_2)$

2 element



2 partitions for a & b ✗ Total: 6+8

You find for a, b automatically fixes. half = 7

pseudo code: $f(a, b)$: l, r

m : len(a) n: len(b) total = m+n half = $\frac{m+n}{2}$

Binary search on a ⇒ partition/split. } ∵ total elements in partition should be $\frac{m+n}{2}$

Automatically split in b is fixed

$$\text{partition\_a} = \frac{l+r}{2}$$

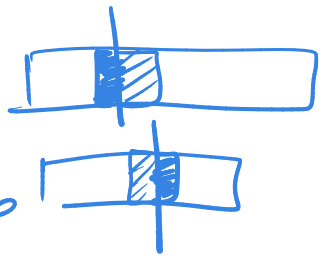
$$\text{partition\_b} = \frac{m+n}{2} - \text{partition\_a}$$

∵ partition_a + partition_b = $\frac{m+n}{2}$

Q. Is this split perfect ?

$a[\text{partition\_a}] < b[\text{partition\_b}+1]$

$b[\text{partition\_b}] < a[\text{partition\_a}+1]$



if m+n is even

median ⇒ $\max(a[\text{partition\_a}], b[\text{partition\_b}]) +$

$\min(a[p\_a+1], b[p\_b+1]) / 2$

else

median = $\max(a[p\_a], b[p\_b])$

If p_a ← p_a = $\frac{p\_a + r}{2}$ ⇒

else p_a → p_a = $\frac{p\_a + l}{2}$ ←

a = [1 2 3 4 5]
b = [5 6 7]

5
3
} 8

half = 4.

1 2 3 4 5 5 6 7

$\Rightarrow \frac{4+5}{2} = 4.5$

5 ⇒ mid(5) ⇒ 3

1 2 3 | 4 5

There is element in left partition

l = 0

3 < 6 ⇒ Yes

or 5 | 6 7

some element on right side

r

5 < 4 ⇒ No.

1 2 3 5

4 5 6 7

p_a ⇒

$p\_a = \frac{3+5}{2} = 4$

p_b = 0

4 < 5 ⇒ Yes

nothing < 5 ⇒ Yes

1 2 3 4

5 5 6 7

median ⇒ max(4, nothing) + min(5, 5) / 2

$= \frac{4+5}{2}$

$\Rightarrow 4.5$

ans.

Computers

64-128 bytes ★ .

registers

CPU ⟺ cache

64 KB

I/O pipeline ⟺ RAM ⟺ SSD

8-16 GB

512 GB - 1 TB

primary memory

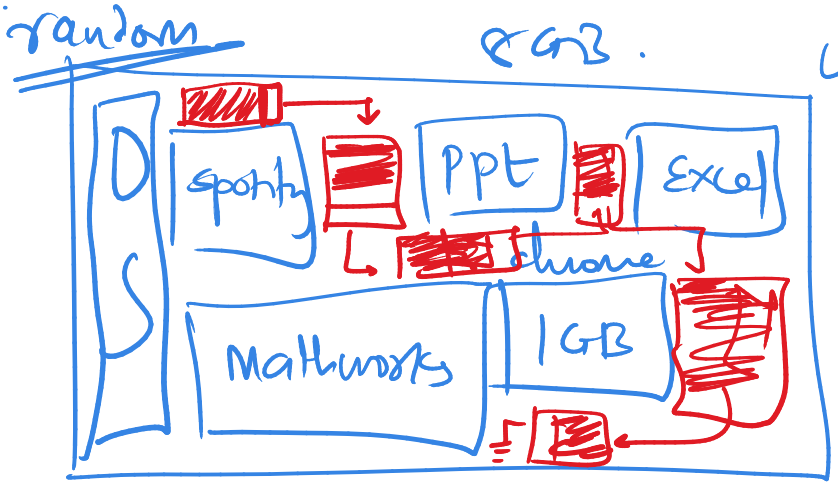
drives

50 GB

software

data

huge



only arrays.

3 GB free

strings = array of chars

Linked list

RAM ⇒ Random Access Memory.

software ⇒ 1 GB //

Array X

big scale

10^6 elements ⇒ integers 1 integer ⇒ 4 bytes min.

4×10^6 bytes ⇒ 10^3 kilo 10^6 Mega 4MB

XAI ⇒ Grok

↑

$3.14 \times 10^2 \times 10^9$

314 billion numbers matrix

$3.\times 10^{11}$ bytes ⇒ RAM

Microsoft Windows Mac

They fragment it.

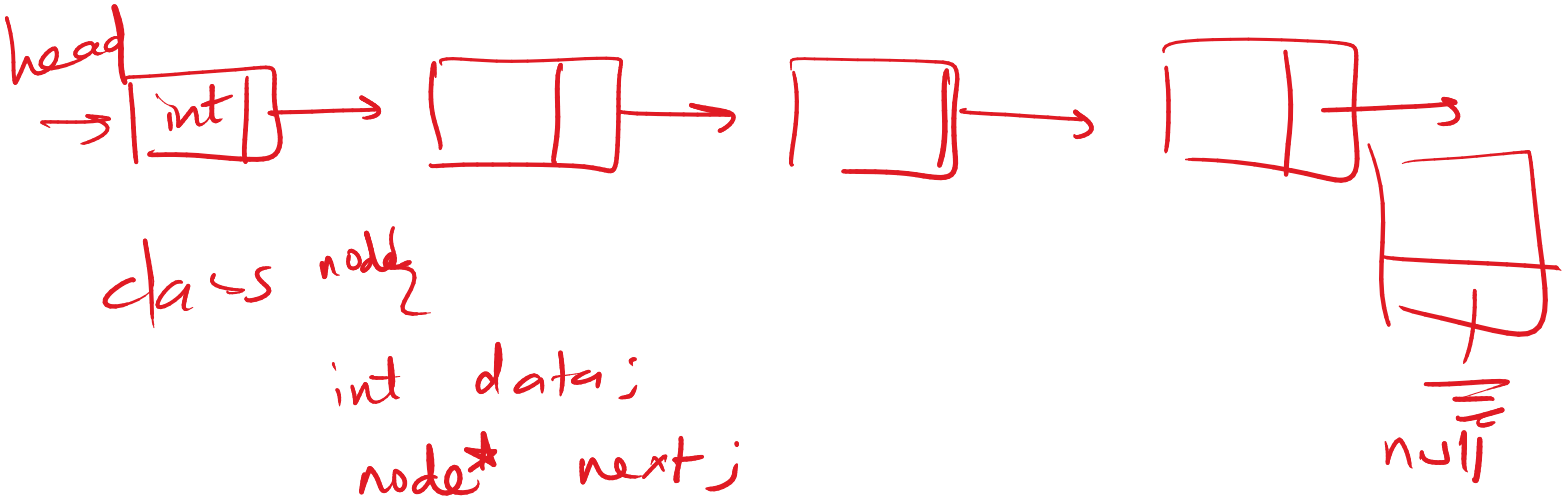
RAM

Chrome

1GB.

Data

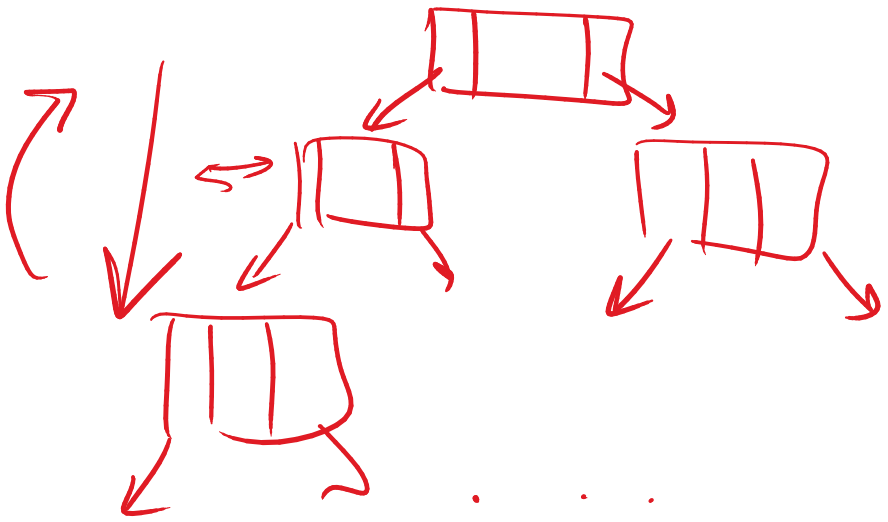
Linked list




```
class node{
    int data;
    node* next;
}
```

Why just next ?

```
class node {
  int data;
  node* left;
  node* right;
}
```



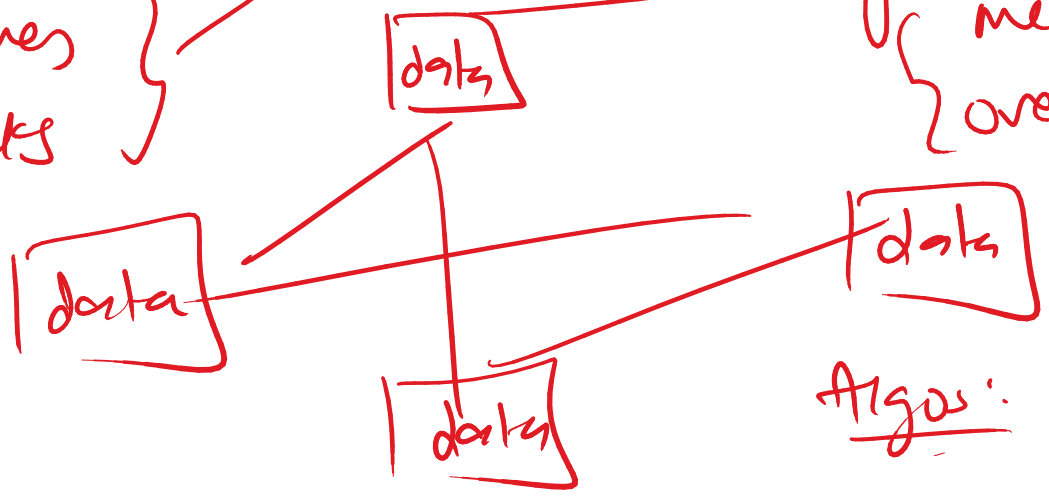
Binary Tree

Restriction

Help!

Queues
Stacks

Auxiliary



segmentation fault
memory
overflow error

Algos:

80%
arrays

Greedy
DP
D n C

20%

DS

Linear

non-linear

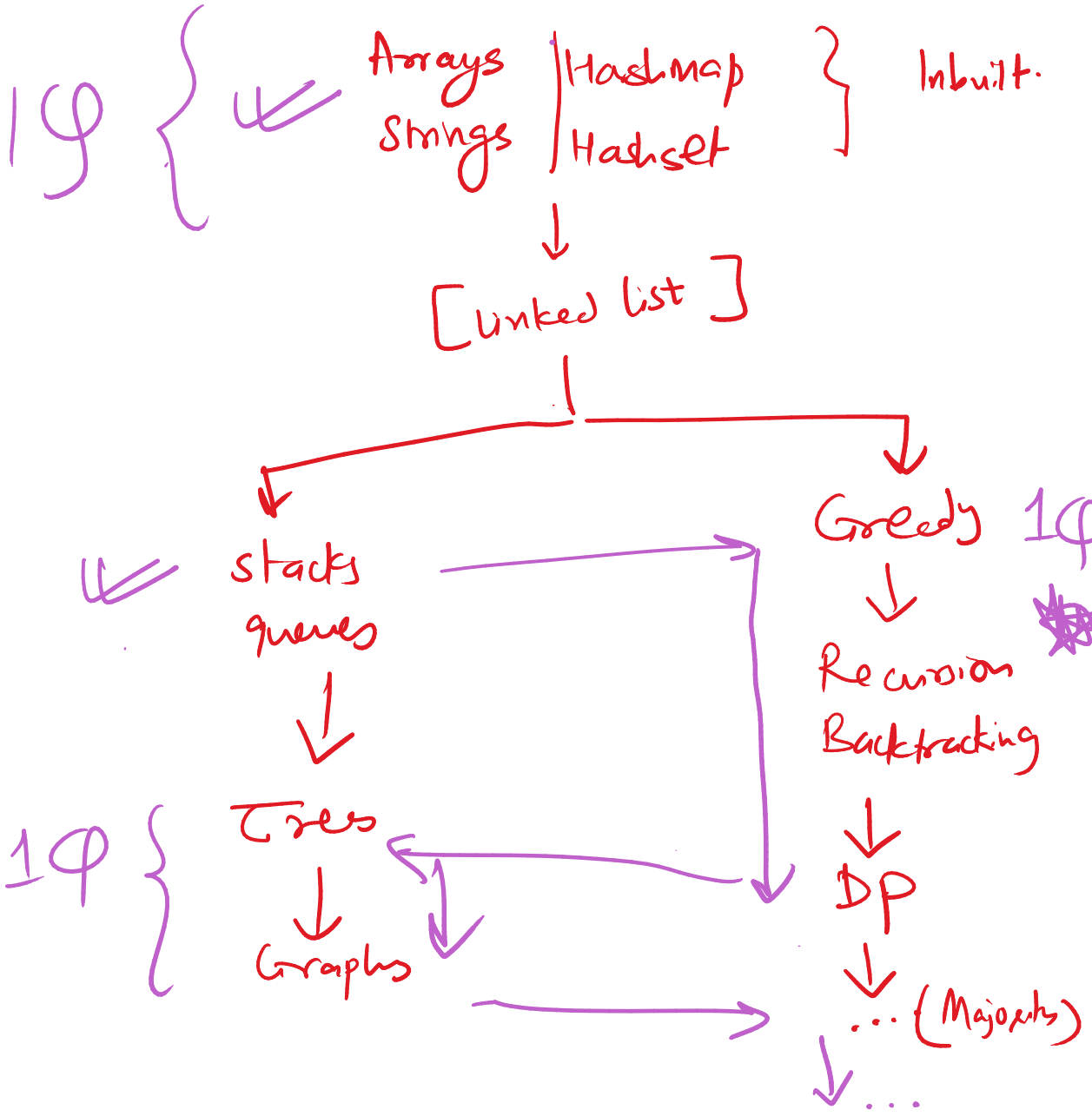
x

arrays
strings
stacks
queues

linked list

auxiliary

Trees
Graphs


1φ
Arrays
Strings
Hashmap
Hashset
Inbuilt.
[Linked list]
stacks
queues
Trees
Graphs
1φ
Greedy 1φ
Recursion
Backtracking
DP
... (Majority)
...

Strings ⇒ character array string: s = "sagar"

s[0] = 's'
s[1] = 'a'
⋮

Educational Rounds on Codeforces : 1st. A

string s = "datascience" → k elements.
k = 4 "encedatasci"

| Arrays ~ strings

check if a string is palindrome. s = "malayalam"
Yes. O(n) i ii j

string s = " ␣ " all chars are unique !!

— × — × — × — × — × —

Specific operations / methods.

P.S.A, DA } arrays

Pattern matching } strings

k = "mat" s = "mathew is a funny mathematician."
0 2

[0, 2, y] ← (3)

Naive:

"mat" ⇒ 3

h(map) = h(mat)
32 19
O(1)

ath
hew
ewi
⋮

h()

for i = 0, i < |s| - |k|, i++
if s.substring(i, i+|k|) == k
ans.push_back(i)

O(k)

O(nk) → O(n)

h(sagar) ≠ h(ragar)

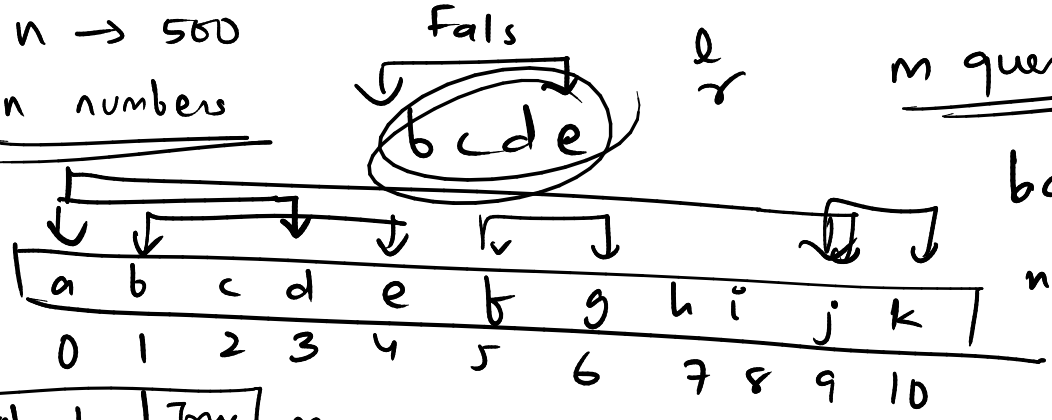
| Rabin Karp algorithm

O(n) size of substring X → KMP algo, Z algo.

array → linked list → trees → graphs } DS
array → stack / queue

searching, sorting → greedy → recursion + backtracking → dp → bitmasking → strings.

① 2 → n → 500 n numbers



a,b,c,d

AP.

O/P: | True | False | | True | m

l | 0 | 1 | 5 | 0 | 4 |

r | 3 | 4 | 6 | 9 | 10 |

(l_i, r_i) ⇒ (0,3) (1,4) (5,6) :

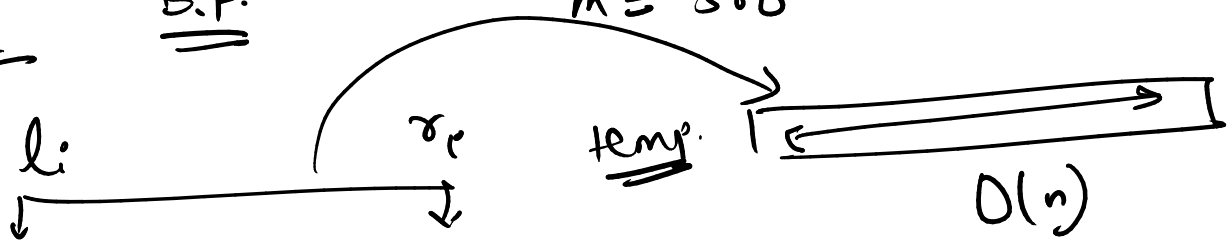
{b c a d} Yes.

b-c = c-a = a-d

m. n/2

n = 500
m = 500

1st B.F.



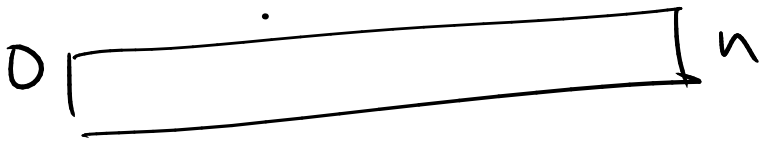
Sort them O(n log n) ↔ traverse O(n)

a sequence is AP only if it is sorted.

$O\left(m \cdot \left(\cancel{n} + n\log n + \cancel{n}\right)\right) \Rightarrow$ BF

$O(m.n.\log n)$ $\quad m = n = 500$

$n^2 \log n$



$500^2 \log 500 < 10^8$

(3 6 12 9) not AP $\longrightarrow$ sorted

bcd $\quad$ bcde

Hashsets and hashmaps.

Q. uniqueness | non-repeating | no duplicates

↳ "sets" : collection of unique objects.

Q. frequency | how many times each element appears |

count of elements

↳ "maps" : { key : value

an : no. of times

}

Q. "abccdeabcf" → "cccaabbdef"

highest count → lowest count

map : { (keys) (values)

a : 2

b : 2

c : 3

d : 1

e : 1

f : 1 }

already sorted on keys

⇒ sort on values

In a map, you can't !!

vector <pair<int, int>> C++.

{ {'a', 2}

{ (a, 2) (b, 2) (c, 3) (d, 1) (e, 1) (f, 1) }

class pair :

public:

int char

sort (compare)

for (auto x : m)

max (m.second)

} 1 entry O(n)

all entries O(n log n)

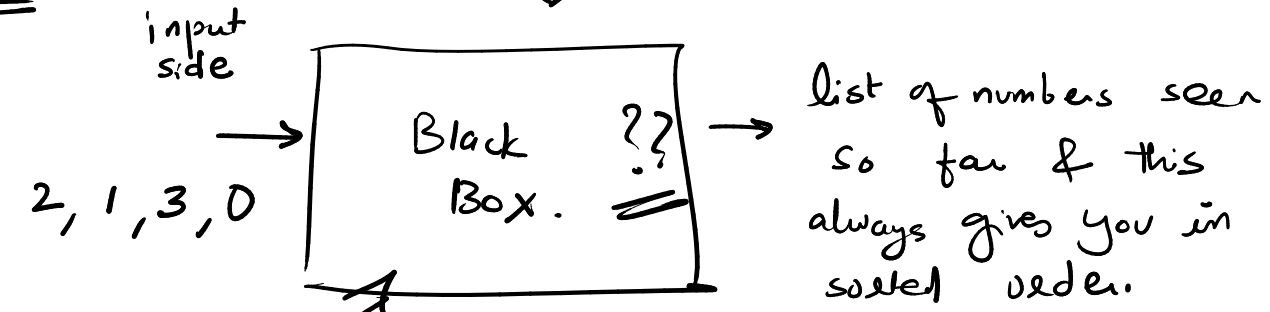
{ (c, 3), (a, 2), (b, 2), ... }

ans += 'cba'

① Data structure | Abstract Data Type

SortedList

Imagine



[2, 1] ⇒ [1, 2] sort( ) ↳ [1, 2, 3] ↳ [0, 1, 2, 3]

[2, 1, 3) ⇒ [1, 2, 3] O(n log n)

[2, 1, 3, 0] ⇒ [0, 1, 2, 3) ⇒ (n log n)

6, 5, 4, 3, 2, 1 n . n log n

[6]
↳ sort
[5, 6)
↳ sort
[4, 5, 6)

— pause —

vectors — arrays

arrays → Arraylists

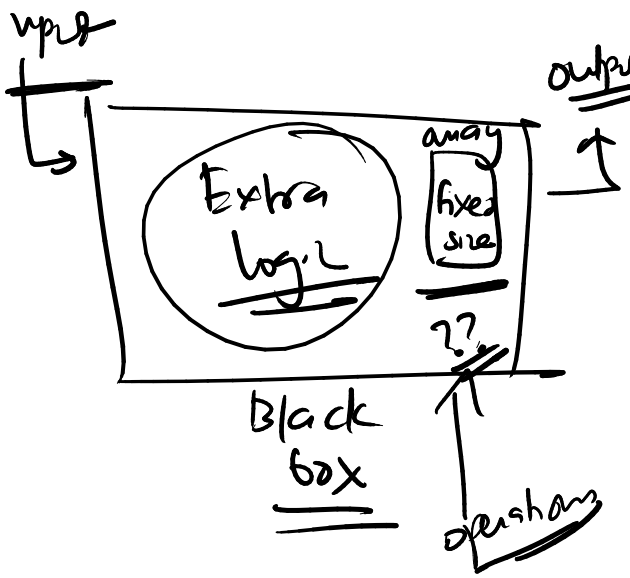
What advantage?

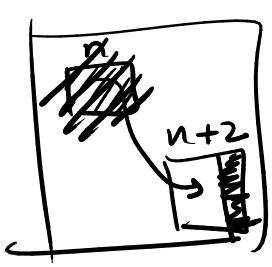
In array, size beforehand

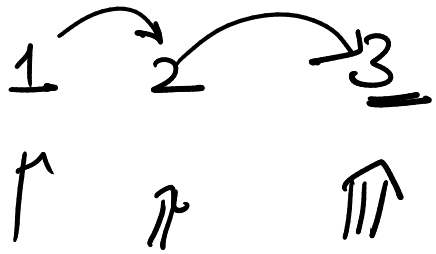
In vectors, size is dynamic!!

n+2

primitive data type





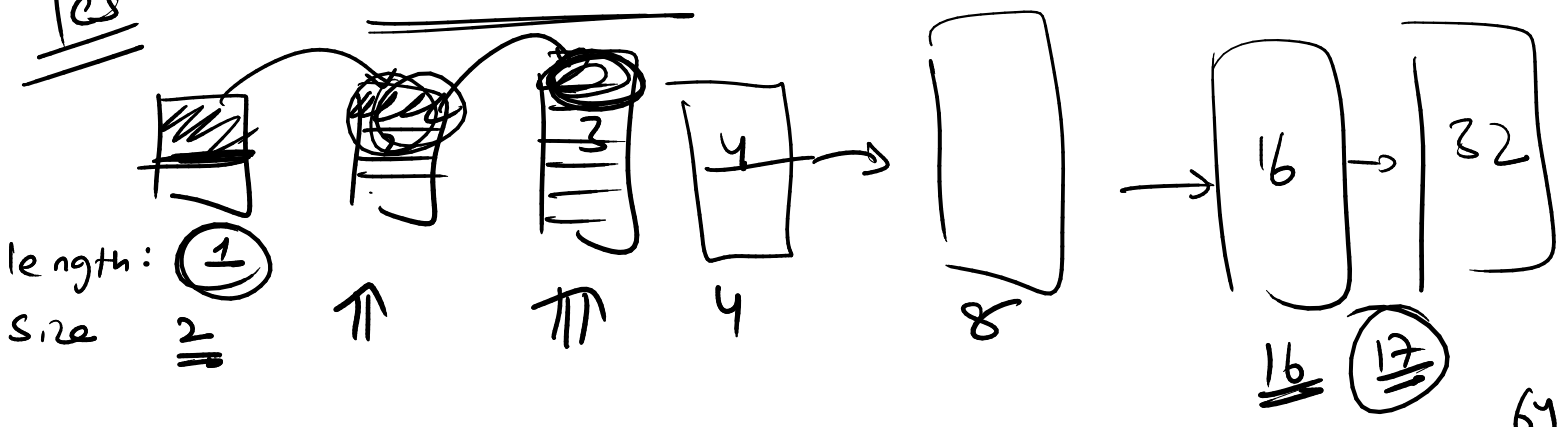


n

$O(n^2)$

$1 + 2 + 3 + \dots + n-1 = \frac{n(n-1)}{2}$

Yes: Amortization



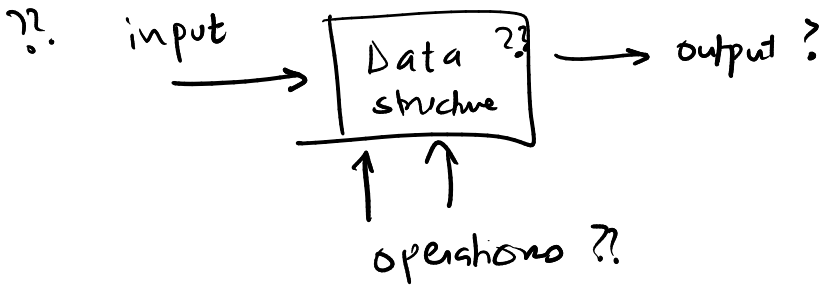
64

$1 + 2 + 3 + 4 + 5 + 6 + 8 + \dots + n$ grows

$O(n)$

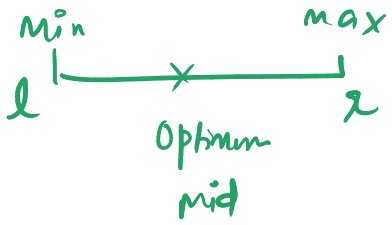
$1 + 2 + 4 + 8 + 16 + \dots + n$

$n\left(1 + \frac{1}{2} + \frac{1}{4} + \dots + \frac{4}{n} + \frac{2}{n} + \frac{1}{n}\right) = 2n$



how is it better than using primitive data type ?

min max



while (l ≤ r) :

mid = (l + r) / 2

if (optimal sol^n found) ← arr[mid] = key

return mid

else if (cond^n to go left)

r = mid - 1

else l = mid + 1

l = 1 l = 0, r = 1

l = 0 r = 0

mid = 0

l = 0 r = 1

mid = 0

Recap:

Part 1

a) Time complexity

b) Arrays :

(i) Sorting → n^2 sorts : Bubble

→ nlogn sorts : quick, merge

(ii) Searching → linear & binary

(iii) Hashing → O(1) search

(iv) Hashsets & Hashmaps.

(v) Techniques : Array rotation

2 pointer

sliding window

prefix sum & difference arrays

(vi) Recursion.

Backtracking

c) Couple of leetcode problems.

Part 2

Auxiliary data structures. (Helping DS)

↳ Stacks & Queues.

90% there are no standalone questions.

DS:

1D { Arrays, Linkedlists }

2D { Trees, Graphs. } ← × unless you have

auxiliary DS { stacks, queues } *

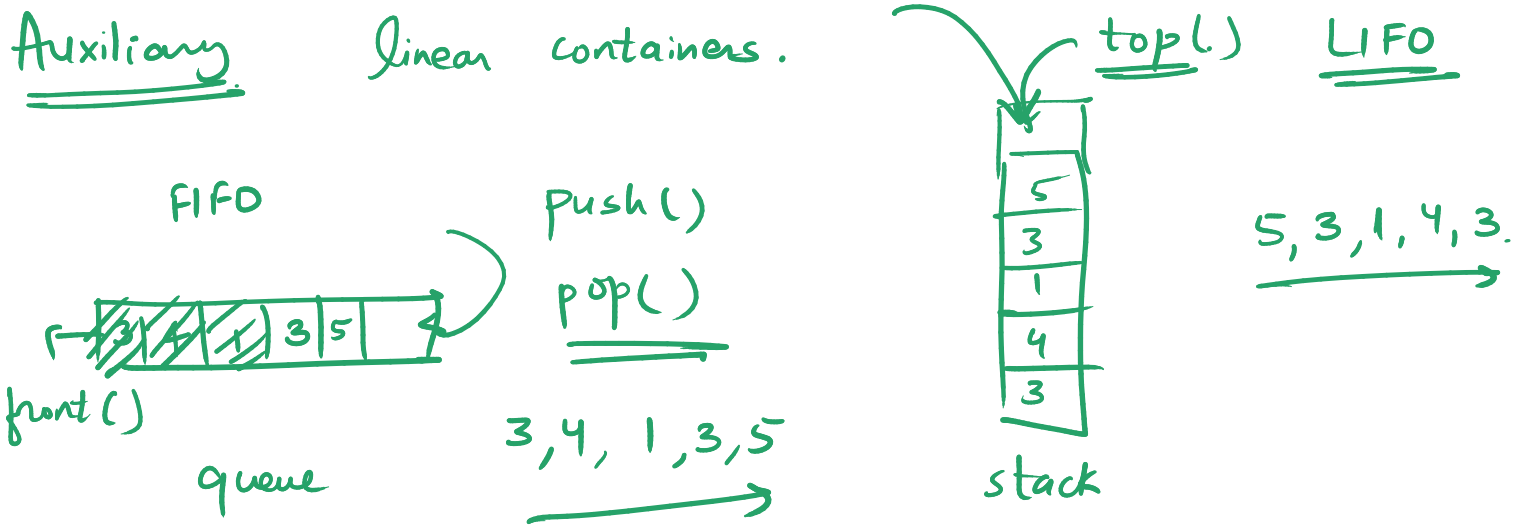
Recursion & backtracking

DP

Algo: searching-sorting, divide n conquer, greedy

Number Theory.

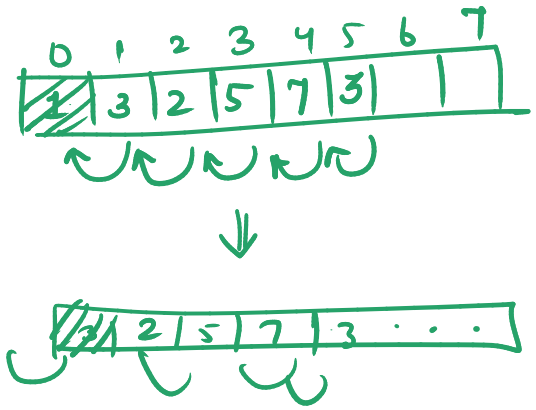
Auxiliary: linear containers.



array:

Queue

1) Push elements at the end
2) Pop from the start.



Insert(3) O(1)

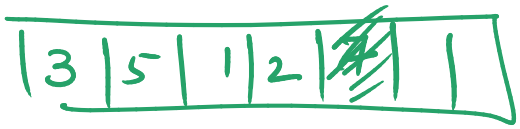
pop() → 1

pop() → 3 O(n)

Stack:

1) Push at the end
2) Pop from the end

O(1) Pulling



pop() → 7 O(1)

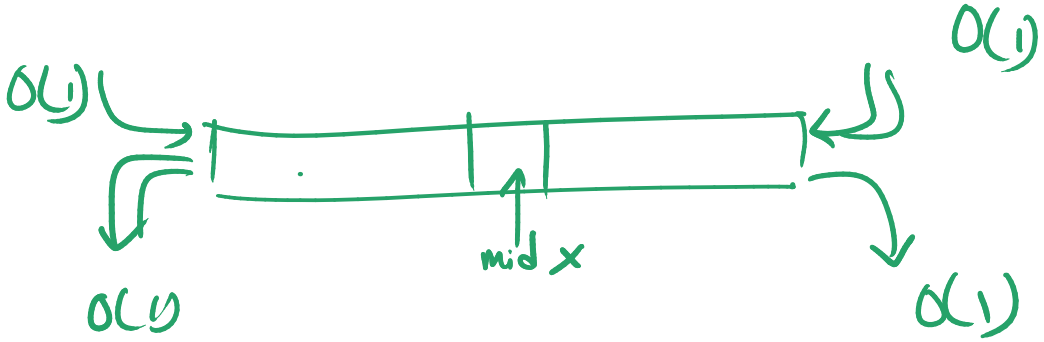
Ideally: You can use arrays as stacks !!

queues():
1) push
2) pop from start
} O(1)

C++/Java: ~~deque~~ Stacks Queues in STL / Collections

Py: ① Use normal arrays as stacks only.

② deque ⇒ Doubly ended queue ← deque (collections)



---

key value pair

map<string, int> m

m[s]++

for (auto x : m)
x.first x.second

Py: dictionary values

m = dict() enumerate

m[s] = 1

for a key in m.keys():

else: m[s] += 1
m[s] = 1

Java :

Hashmap

Thought Concept → Equivalent concept ??

m[s]

- You Don't Know JS Yet

ps

npm x

plain JS

node js

node index.js

imports ✗

Template in stacks : Next greater element (to right)

input: [1 3 2 2 4 1 2 1 6]

output: ans [3 4 4 4 6 2 6 6 -1]

How?

$O(n^2)$

$O(n)$

ans

```
for (int i = size-1 ; i >= 0 ; i--) {
    × if (s.empty())        ans.push_back(-1);
    × if (s.top() > arr[i])  ans.push_back(s.top())
      while (!s.empty() && s.top() < arr[i]) s.pop()
      if (s.empty())  ans.push_back(-1)
      else  ans.push_back(arr[i]);
    s.push(arr[i]);
}
```

logic: looping backwards :

1. If my stack is empty, push (-1)
2. If top of stack > current element
   push (top of stack)

NGR

3. While s is not empty & top is lesser
   keep removing elements from stack
4. if s.empty push (-1)
   else push (top of stack)
5. Push the current ele in stack.

Variations #1 : NGL (Next greater ⇒ to left).

input :	1	2	1	1	[3]	2	1	4
output :	-1	-1	2	2	-1	3	3	-1

changes : 2) (1) Instead of n-1 to 0,
0 to n-1 →

(2) result.reverse() X

#2 : NSR (Next smaller element on right)

Same as NGR only comparison signs will flip. //

#3 : NSL (Next smaller on left)

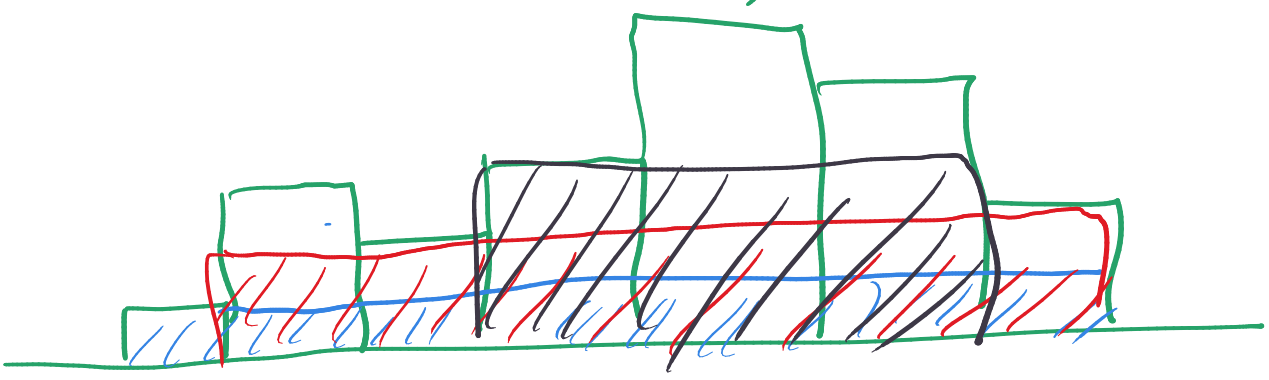
4 codes : direction of traversal | Comparison sign with top

NGR NGL NSR NSL

2 codes : Interview Content

- Max. area of Rectangle on histogram
- ~~Max~~ Max. stock span problem

arr = 1, 3, 2, 3, 5, 4, 2



Recap: Queues (FIFO) + Stacks (LIFO) ⇒ ready made {stacks & queues} in C++/Py/Java.

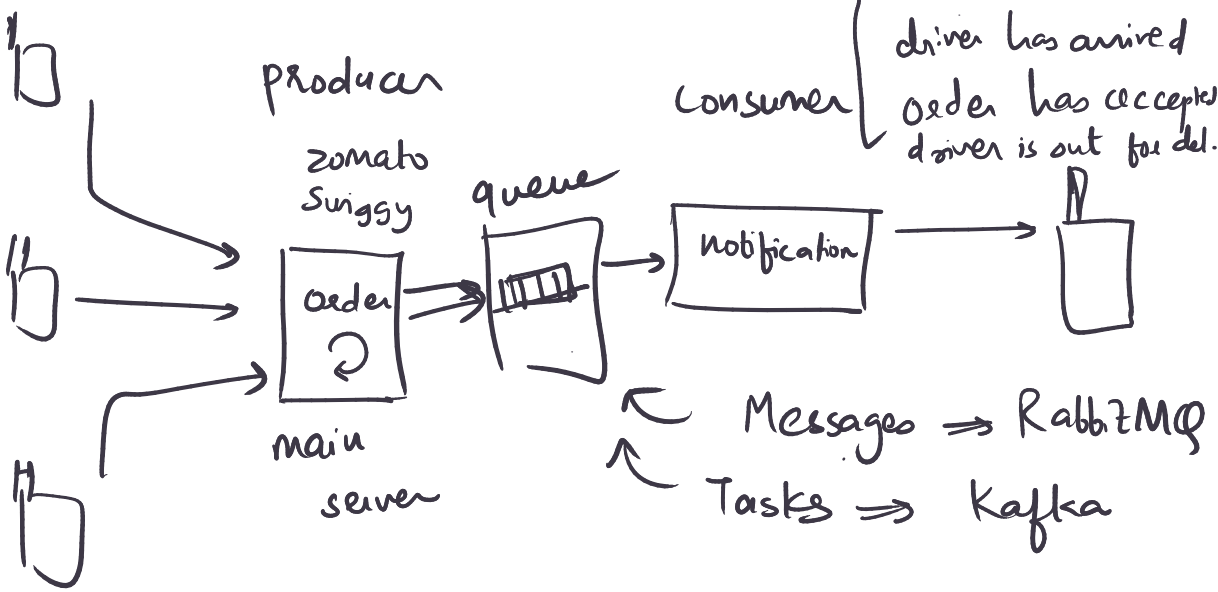
Level order traversal
- Trees
- Graphs

Template:
- NGR
- NGL
- NSR
- NSL

Recall: Hashset by our own. In C++.
Set<string> ||
set<string>

Classic
1. Stock span
2. Max area under histogram.

Usecase of queues: Zomato/Swiggy



greater than

Classic. (1) Stock span problem.

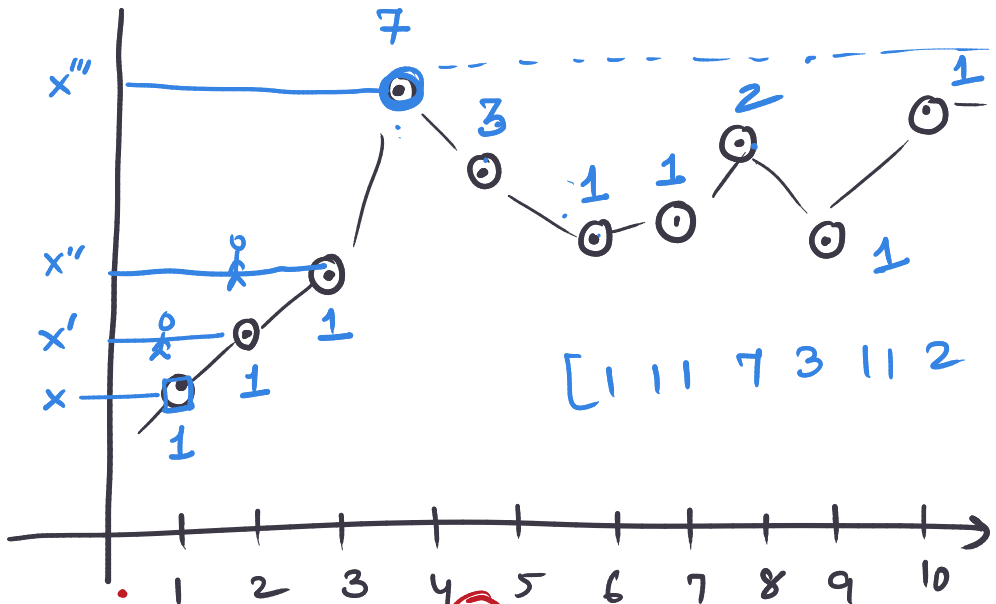
1	2	3	4	5	6	7	8	9	10

stock-price = [10, 12, 14, 12, 10, 8, 10, 12, 14, 11]

o/p: span = [1 1 8 5 3 1 1 1 2 1]

start today ⇒ how many consecutive days, I remain maximum.

stock



Stock span

[1 1 1 7 3 1 1 2 1 1]

How?

$O(n^2)$

↓

$O(n)$

Soln.

i	1	2	3	4	5	6	7	8	9	10
	0	1	2	3	4	5	6	7	8	9
arr:	8	10	12	16	13	11	12	14	10	15
ngr:	10	12	16	-1	14	12	14	15	15	-1
ngr_i:	1	2	3	10	7	6	7	9	9	10
span = ngr_i - i	1	1	1	7	3	1	1	2	1	1

if ngr = -1, ngr_i = size()

⇒ span

- Stack< ↓ >

i

{int, int}

pair

```
for (int i = n-1 ; i >= 0 ; i--) {
    while ( !s.empty() && s.top().first < arr[i])
        s.pop()

    if (s.empty())
        ans.push ( n - i)
    else if ( s.top().first > arr[i])
        ans.push (s.top().second - i)

    s.push( { arr[i] , i } )
}
```

STOCK SPAN

GFG

Q.2 Max area under histogram.

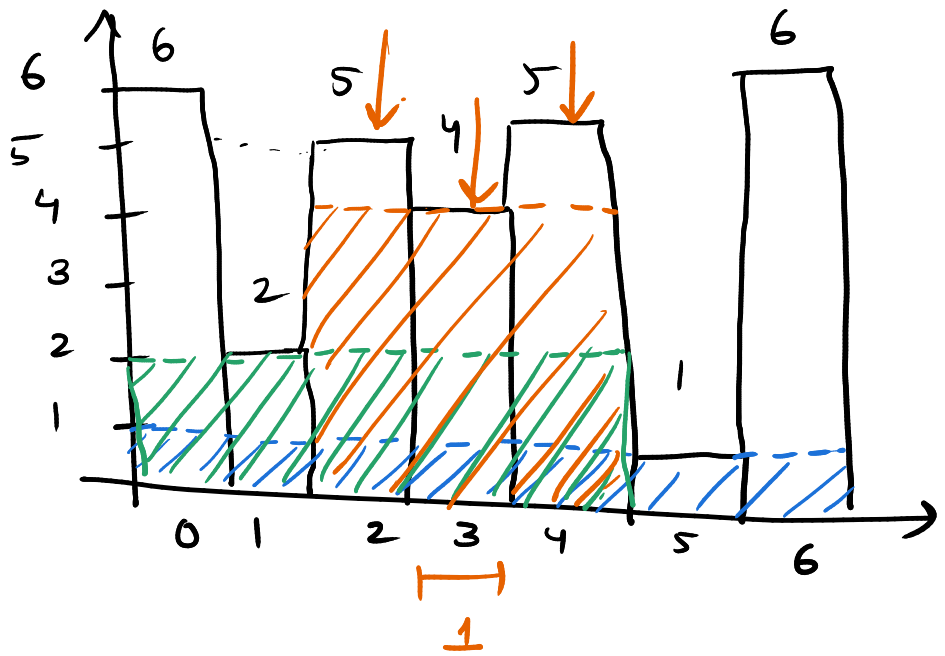
arr[] = [6, 2, 5, 4, 5, 1, 6]

▨ ⇒ 7

▨ ⇒ 10

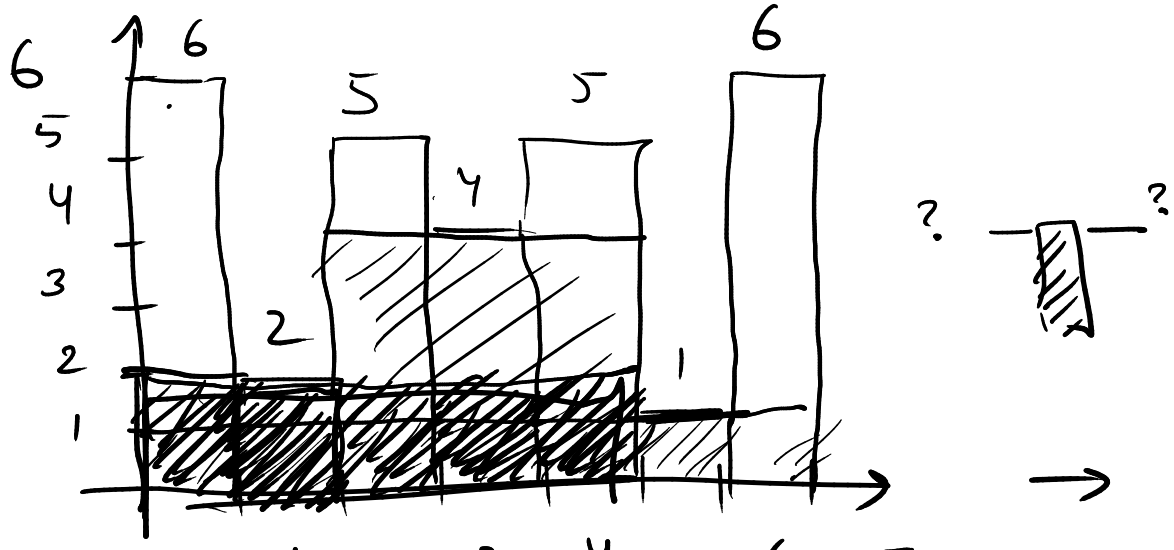
▨ ⇒ 12

Classic



output

12



i	0	1	2	3	4	5	6
height → arr[i]	6	2	5	4	5	1	6
nsr	2	1	4	1	1	-1	-1
nsl	-1	-1	2	2	4	-1	1
nsr_i	1	5	3	5	5	7	7
nsl_i	-1	-1	1	1	3	-1	5
width	1	5	1	3	1	7	1
areas:	6	10	5	12	5	7	6

0 to n-1

-1, n

⇒

→ -1, 0 to n-1

⇒ Max 12

$$width[i] = \underbrace{(i - nsl\_i)}_{\text{left}} + \underbrace{(nsr\_i - i)}_{\text{right}} - 1$$

$$\boxed{width[i] = nsr\_i - nsl\_i - 1}$$

$$area[i] = width[i] * arr[i] \Rightarrow \underline{max}$$

Famous problem : Min no of notes required to make X amount.

notes = { 5, 1, 2, 10, 50, 100, 200, 20, 500, 2000 }

amount = 786

Sort: notes = { 1, 2, 5, 10, 20, 50, 100, 200, 500, 2000 }

786 = 500 + 200 + 50 + 20 + 10 + 5 + 1

(7) notes. || Minimum

7 steps

GREEDY !! Every step is best choice.

↳ If you make n steps & all steps you make best decision, overall ans should be best

$X = 6$ denominations $= \{1, 3, 4\}$

greedy :

6	4
2	1
1	1

} 3 notes

Best ans: $6 = 3+3$ || 2 notes

greedy works ; $arr[i+1] \geq 2 * arr[i]$

Best ans : Sometimes you make a suboptimal decision now, so that it ~~can lead to~~ better decisions later. || Foresight

6 , Min notes

4 ×
3

Technique : 6-7 sessions

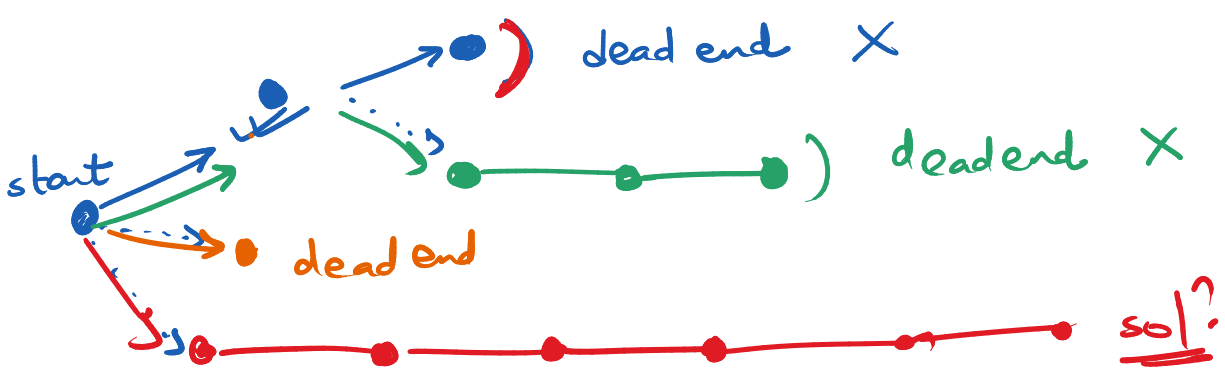
↳ 1 compulsory Test Q.
Interview

Structure to a sol^n → template.

multiple problems → variants

New template

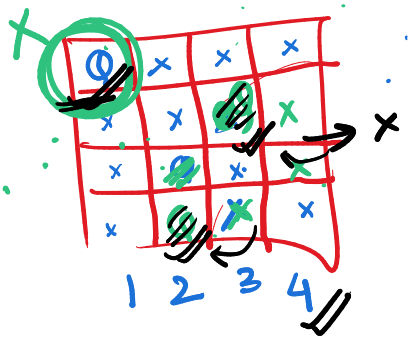
Certain problems where only possibility to solve them is to perform an exhaustive search. Hard problems.
(exponential time complexity)



exhaustive search

X

n-queens

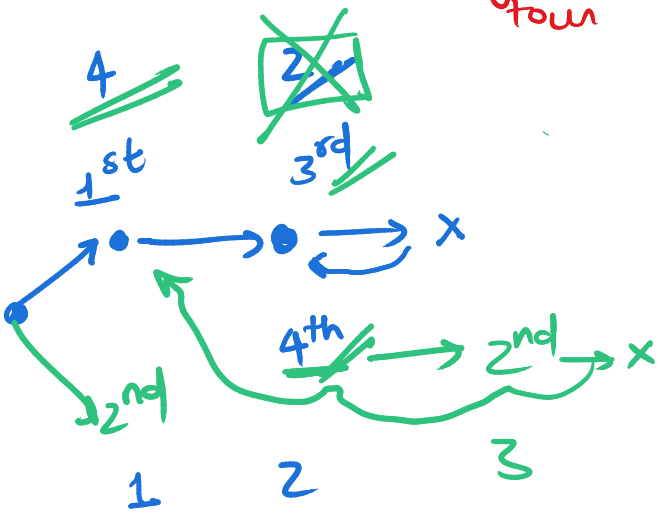


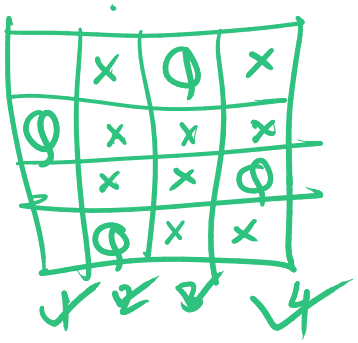
levels.

4 possibilities.

constraints

9×9 Knights tour

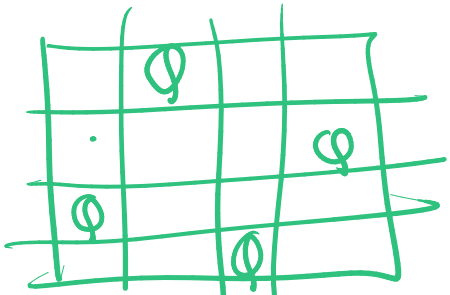




STOP?

sol^n

→ 2 → 3 → 4
1 1 1

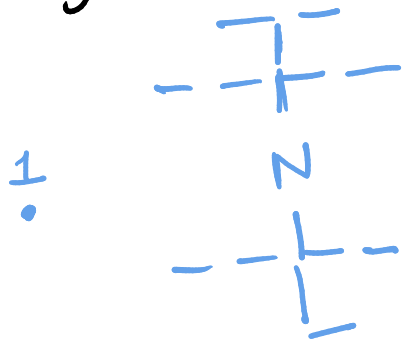


2 sol's

4×4

8×8
92

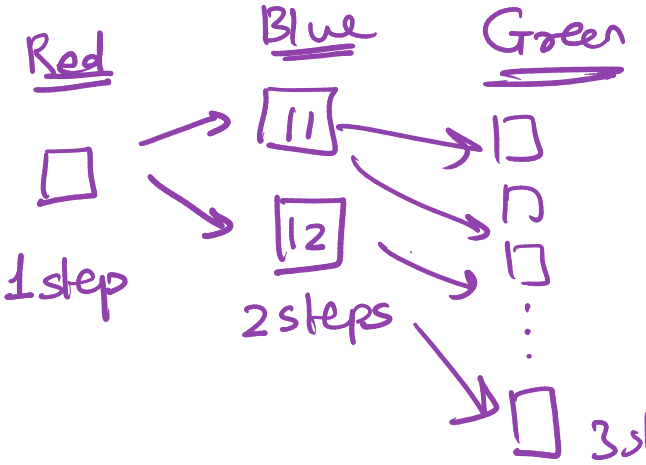
```
bool find_solution ( level , other params ... ) :
    if (found_a_solution) :
        print / save the soln
        return true ;

    bool is_valid = false ;

    for all possibilities at this level :
        if (constraints are satisfied) :
            save current possibility
            if ( find_solution (level + 1 , other params ...)) :
                is_valid = true ;
                return is_valid ;

            remove current possibility

    return is_valid .
```

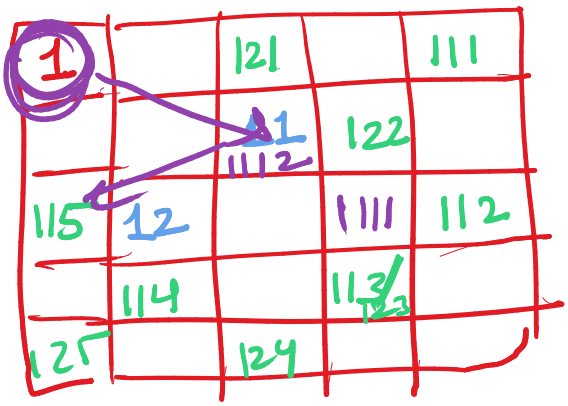


① N-queens.

② Knights tour problem.

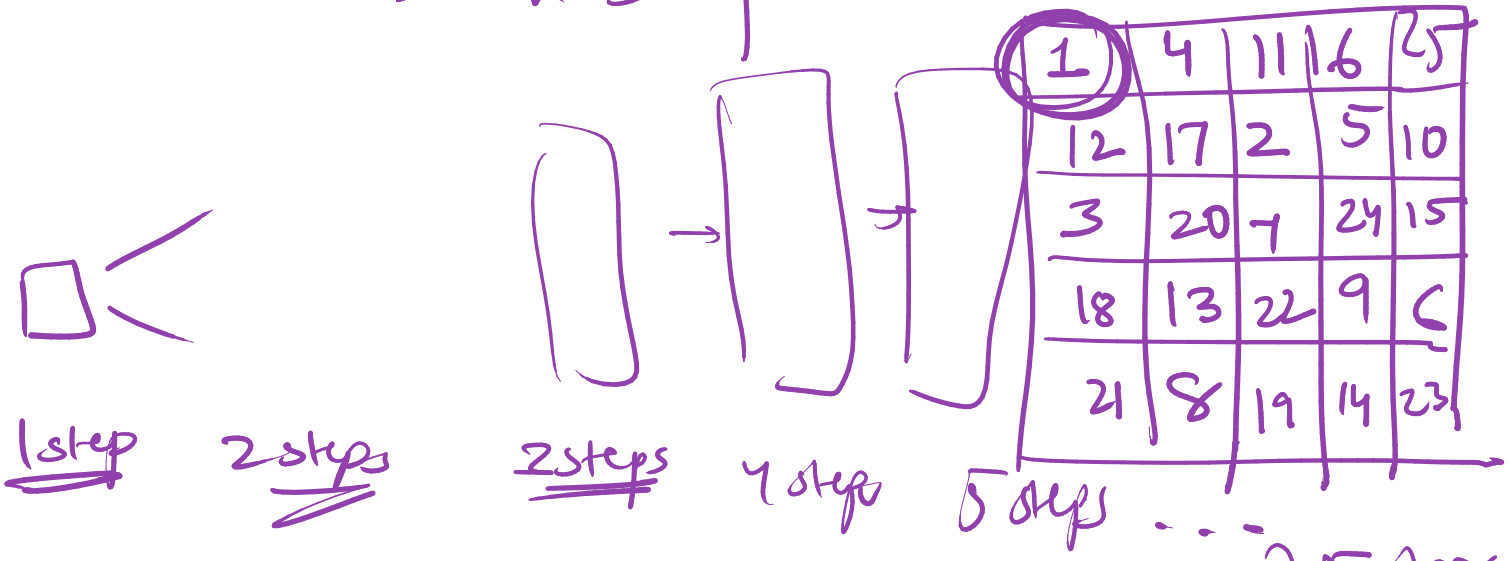
levels ⇒ imagine !!





Q. circuit. You have touch all squares once without repeat.

n cells ⇒ n steps



1 step 2 steps 3 steps 4 steps 5 steps ... 25 steps

Level ⇒ no of steps

no. of col^n.s

Stacks : ① Max area under histogram ② Stock span problem.

Interesting : min notes that give me change | greedy approach.

Counter case : n = 6 am = [1, 3, 4]

↳ You take a suboptimal decision right now, so that you can take a better decision later.

6 → 4 (best now) → 1 → 1 ⇒ 3
6 → 3 (suboptimal) → 3 (better later) ⇒ 2 | foresight.

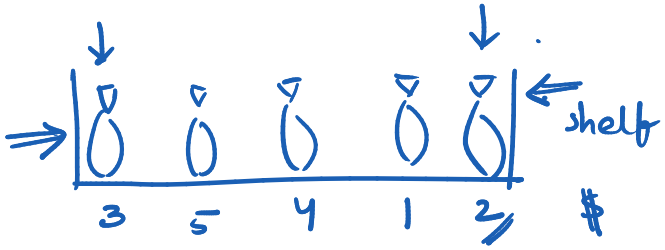
---

1. Kind of problems → Combinatorial problem (# counting, # no. of ways)
→ Optimization problem (extreme values)

C/O problem → greedy approach ? ⇒ Searching & Sorting.

1 wine bottle / year

price of wine doubles every year.



Consume max value wine.

In what order should I consume ? # take out of shelf from extremes.

min

2 2 12 32 80 ⇒ maximum

6 10 8 ~~2~~ ✗
~~12~~ 20 16
40 32

2 problems. : Online test → BS

n chefs d dishes.

am = [a_1 a_2 a_3 ...] → speed of chefs.

Min. time in which d dishes could be made

Min time = 1

max = min a_i * d

~~1~~ ~~2~~ ~~3~~ ~~4~~ 5 6 . . .. — — ~~46~~

1. 0 → greedy ? × → optimal choice at every step isn't optimal.

adhoc ? Eg. max area under histogram (stacks) ✓

dp ↳ cannot come up with intuitive recursion. (then adhoc)

??

Thm : Every recursion → loops.

loops → recursion ✓

} max area under histogram.

n bars ⇒ hist.

Recall Recursion $f^n(n)$:

base cond^n

a = $f^n(< n)$

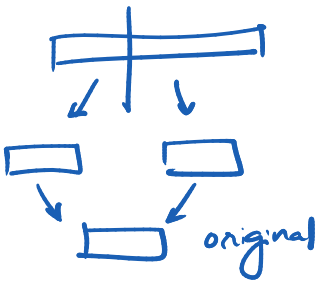
b = $f^n(< n)$ } decomposition

ans of n = a+b } Recomposition

fib(n) = fib(n-1) + fib(n-2)

fact(n) = n * fact(n-1)

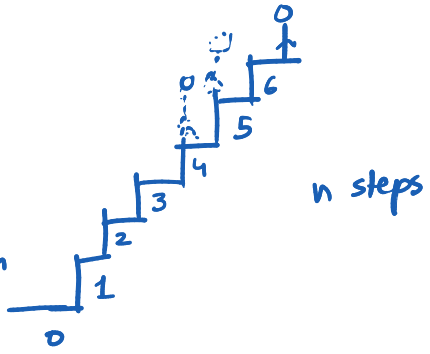
Mergesort.



Q.1 climb staircase → 1 step / 2 steps

if the steps you take in order forms the way/pattern

how many different ways you can climb ?



i/p : 3	i/p 4
o/p: 3	o/p 5

W_n ⇒ # ways staircase with n steps.

W_n ⇒ ans.

W_1 = 1

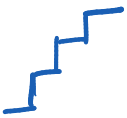
W_2 = 2

W_k ⇒ # ways stair with k steps.

n=4

5 ways { (1,1,1,1), (2,1,1), (1,1,2), (2,2), (1,2,1) }

n=3



3 ways { (1,1,1), (2,1), (1,2) }

step

W_n = W_{n-1} . W_{n-2}

fibonacci

$W_n = W_{n-1} + W_{n-2}$

n = 4

3 ways

n = 3 + 1 step

(1,1,1) → (1,1,1,1)

(1,2) → (1,2,1)

(2,1) → (2,1,1)

+

2 ways

n = 2 + 2 step

(1,1) → (1,1,2)

(2) → (2,2)

Intuitive Recursion : If I want ans for n — ans(n)

Can I derive it from smaller values of n but same f^n. ↳ subproblem

$f(n) = g\{f(n-1)\}$ ⇒ fact.

$f(n) = g\{f(n-1), f(n-2)\}$ ⇒ fibonacci

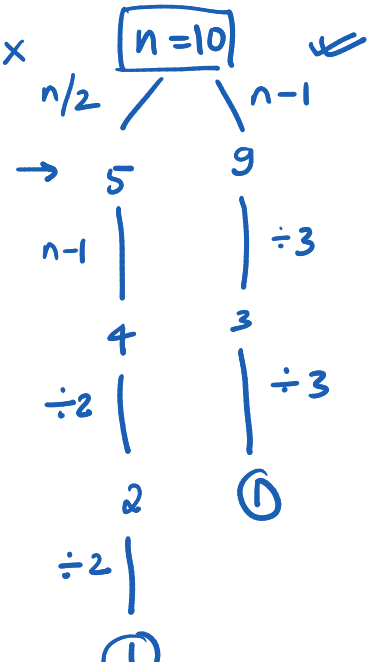
$f(n) \Rightarrow g\{f(n/2)\}$ → Mergesort, BS.

Sorting → x if order of elements matter !!

Q.2 Min ops to 1. — n

3 ops:
a) If $n/3$ ⇒ $n \to n/3$
b) If $n/2$ ⇒ $n \to n/2$
c) $n \to n-1$

} Compulsion — 4



min operations to reduce n to 1.

Optimization → 1. Greedy (Searching / Sorting / Min-max)

↳ x Prove: Optimal every step isn't optimal overall. || Suboptimal decision now can lead to better decisions later.

2. Adhoc → not an adhoc if you can write intuitive recursion.

3. Recursion / may fall in DP !!

Combinatorial → 1. Mathematically, come up with pattern or formula ⇒ PnC !! ✗ fails

4 couples M_1W_1 M_2W_2 M_3W_3 M_4W_4 — n couples $\{M, W\}$

3 seats $\underline{M_1}$ $\underline{W_2}$ $\underline{W_3}$ — fact → O(n) — m seats m<n

#ways we can form committees ? ⇒ O(n)

Elect m presidents from 2n people.

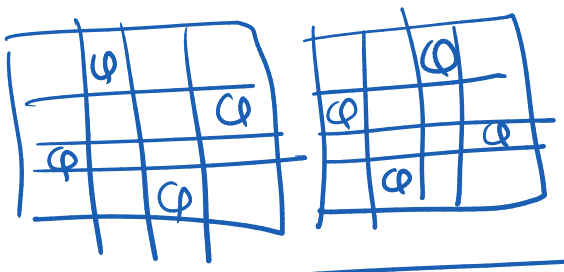
2. Recursion (Intuitive) / may be DP.

Cond^n Total couple shouldn't be present.

↳ Given n, tell me no. of ways I can keep N queens on chessboard no queen attacking ?

i/p : n = 4

o/p : 2



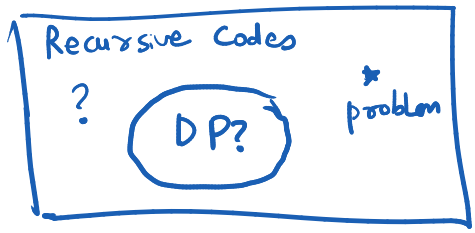
Problem → O → Can I write recursion? → No ? Adhoc

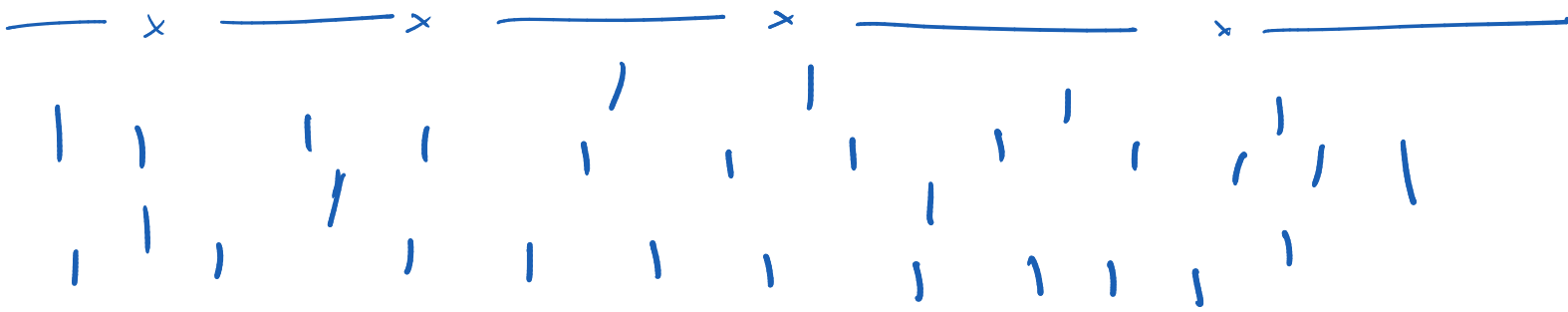
→ Yes? Write ! ★

→ C → Pattern / PnC ✗

→ Can I write recursion? → Yes. ★

★ Recursion → some of them DP.





Recursion → end up making similar calls → computed again & again

→ Cache { memo table } → if you later need same recursion call → fetch ans directly !!

DP

Recursion → no repetitive calls. | all f^n calls are unique → not a DP.

Recursion → repetitive calls → memoize it → DP

↓

Putting it in memo

Recursion + Memoization → DP.

1. Write a recursive code
2. Identify if common/identical recursive calls are made.
3. If yes, create memo table. DP
   If no, that's it. Can't improve further.

Any Recursion with identical calls → fixed technique to memoize → DP !!

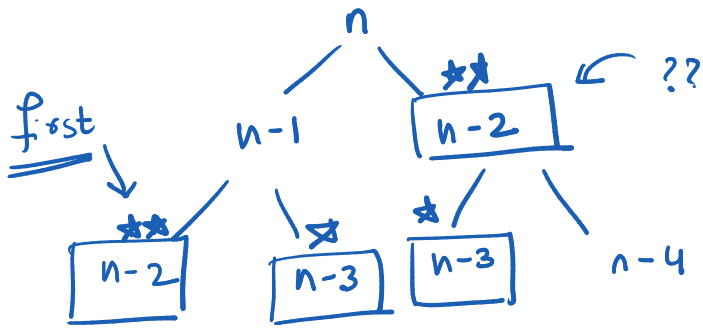
```
f(n):
  // Base
  if(n==0 || n==1): return

  // Problem decomposition
  a = f(n-1)
  b = f(n-3)

  // Problem recomposition
  return a*b/10
```

DP. →

```
f(n):
  // Base. — as it is —
  if (memo[n] != -1)  // cached
      return memo[n]

  a = f(n-1)  // Decomposition
  b = f(n-3)  — as — it — is —

  memo[n] = a*b/10  // store it for future calls
  return memo[n]
```

Why performance boost ?

$fib(n) = fib(n-1) + fib(n-2)$



T.C

$O(1.681^n) \sim O(2^n)$

```
memo[n+1] = {-1}  // Initialize

fib[n] = { }                    O(n)
fib[0] = 1   fib[1] = 1
for ( int i=2, i<n-1, i++)
    fib[i] = fib[i-1] + fib[i-2]
```

loop. X

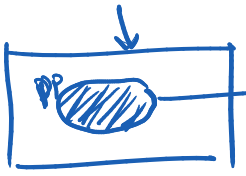
memo — n unique calls.

for others direct ans.

→ O(n)

DP : Introduction

2 problems → Optimization → Greedy → optimal every step isn't optimal overall.
→ Adhoc technique → can't come up with intuitive recursion.
→ Combinatorial → Recursion. (Imp!!) → Mathematics / patterns / formulae



Recursive codes where you are making duplicate calls.

Recursive

Recursion + Memoization = DP

memo[n+1] = {-1}

f(n):
// Base cond^n

// Problem decomposition

// Problem Recomposition

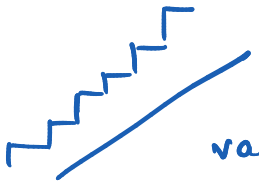
DP →

f(n):
// Base cond^n
if memo[n] != -1
return memo[n]

// Problem decomposition

// Problem recomposition
memo[n] = ans
return ans;

① Fibonacci ⟹ (template)



variation.

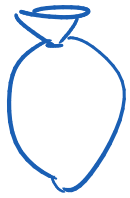
Eg: Min steps to 1. 3 ops ÷3, ÷2, n-1

Template #2:

Knapsack Problem

n items

	0	1	2	3	4
w	2	1	7	4	5
p	10	6	18	12	20



C = 12 kg

bag.

Q. What's the max price you can make in this bag?

o/p : 48 | max.

⇒ this problem is too deep!

{ i_1 i_2 i_3 i_4 i_5 i_6 }

any subset is possible.

{ i_1, i_2 } { i_2, i_3, i_6 }

12 kg. 5kg 7kg

Greedy → 20 + 18 Rs ⇒ 38

5 1 2 4
20 6 10 12
⇒ 48

X

{ price / weight }

BF. Go and calculate all possible subsets. [3] i_1 i_2 i_3

$0 \leftarrow \{\}$ $\{i_1, i_2\}$ $\{i_1, i_2, i_3\} \rightarrow$ sum(prices)

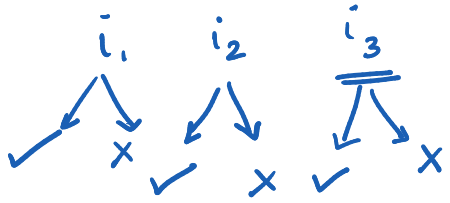
$\{i_1\}$ $\{i_1, i_3\}$

$\{i_2\}$ $\{i_2, i_3\}$

$\{i_3\}$

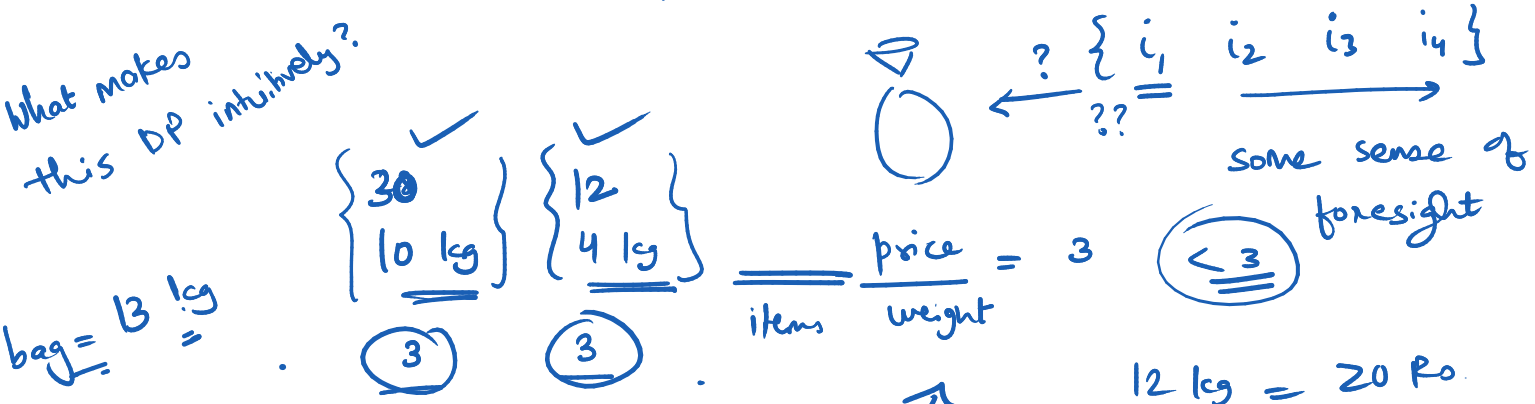
$2^3 = 8$.

bag $\{\quad\}$



Super: Whenever in a question you are given N items, and each item has skip/pick possibility ⇒ Knapsack variant !!

What makes this DP intuitively?



9kg C

12

{ 3kg C

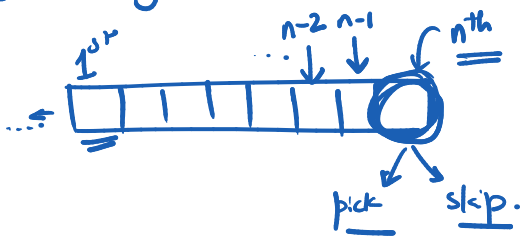
30 }

Highlight

you cannot decide unless seen all the items

① Greedy X ② for loop

Can I go recursion?




```
int knapsack (w[], p[], n, C):
    if ( C <= 0 || n == 0)
        return 0;
    if w[n] > C :
        return knapsack (w, p, n-1, C)
    else:
        return max { knapsack(w, p, n-1, C),
                     p[n] + knapsack(w, p, n-1, C - w[n]) }
```

skip

pick

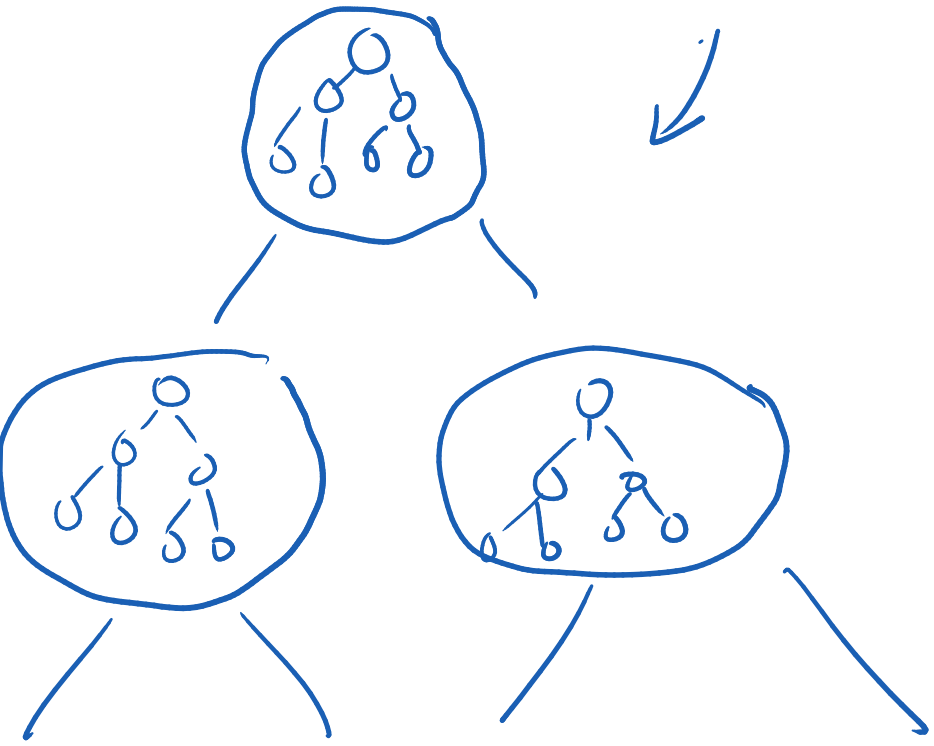
2^n calls

Test case : C = 10 items = w[] = {5, 4, 6, 3}
n = 4 p[] = {10, 40, 30, 50}

pick skip

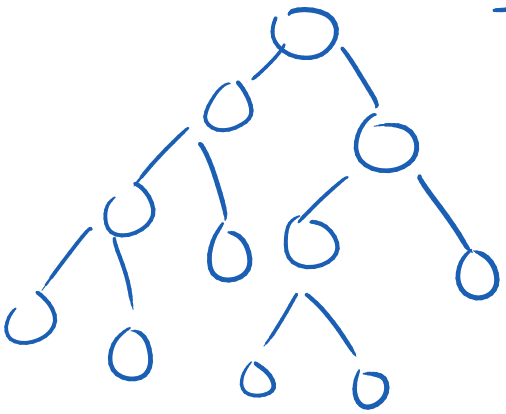
Multiverse

remember

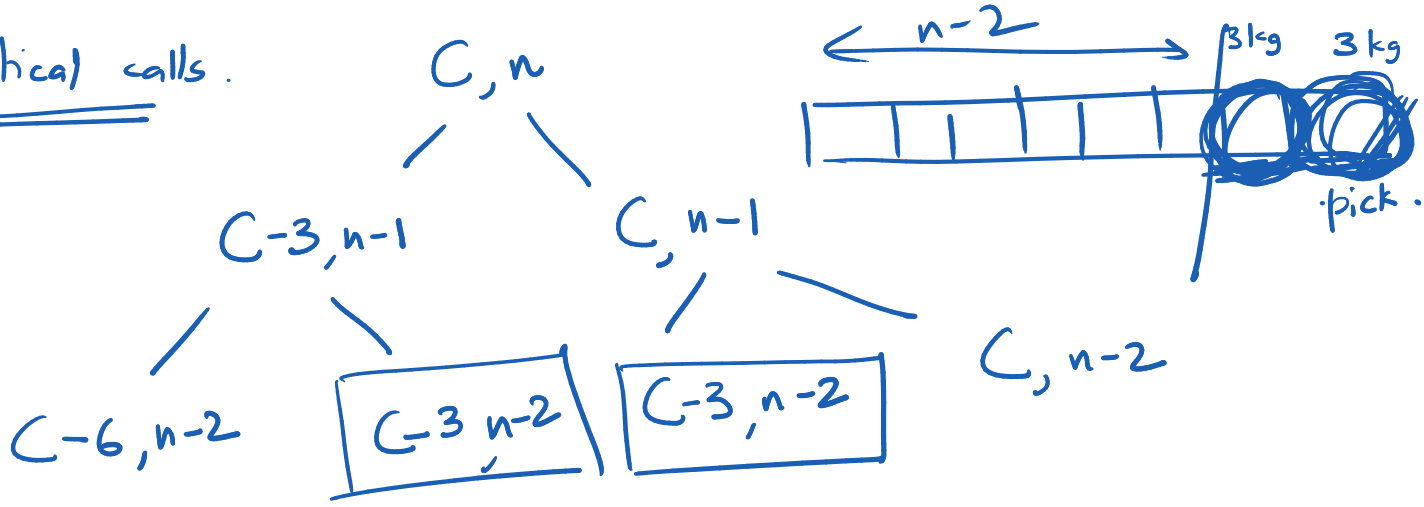


universe

flat



Identical calls.



memo[n+1][C+1]
= {-1}

// Base

if memo[n][c] != -1
    return memo[n][c]

int ans = 0

if w[n] > C : ← skip
    ans = knapsack (..)

else:
    ans = max (skip , pick)

DP ✓

memo[n][c] = ans

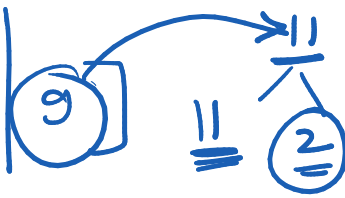
return ans ;

Similar problem

#variation 1

sum = 11

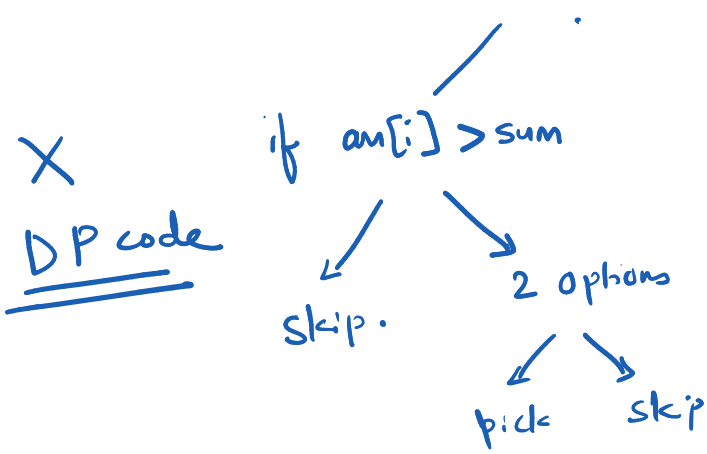
arr = [1, 3, 4, 5, 7, 9, 11, 2]



Q. Is there a subset that sums to the given sum?

```
bool subset_sum (arr, n, sum):
    if (sum == 0)
        return true
    if (n == 0)
        return false
    if arr[n] > sum:
        return subset_sum (arr, n-1, sum)
    else:
        return subset_sum (arr, n-1, sum) ||
               subset_sum (arr, n-1, sum - arr[n])
```

$O(2^n)$



X DP code

How much improvement?

$O(2^n)$ —memo./DP→ $O(n \cdot sum)$ / $O(n^2 a_{max})$

memo[n][sum]

n elements

$n \sim 10^5$

T.C.

T.C.

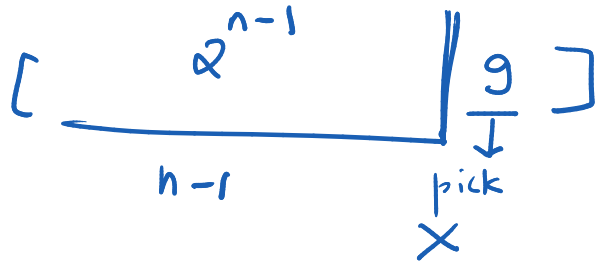
Recursion

Intuition ⇒ How?

DP

Redundant work —skipping→ Optimized code

all possible subsets are not checked → memoize → Yes.



Yes/No

How am I ensuring I will get an answer if 1 subset actually can produce ans?

What did you eat?

Did you eat? Yes/No.

I did not ask you what elements make sum.

Yes/No ? → DP works

Tell me all possible numbers that make sum. → {7,1,3} → $O(2^n)$

Backtracking

↳ exhaustive search 2^n

vs.

Recursion → Memoization → DP

↙ ↘

sometimes you may/may not exhaustive search

DUPLICATE CALLS.

↑ Eliminate this!!

#ways.

$A = [1, 2, 3, 4]$

↳ all possible subsets.

$n(P(A)) = 2^4 = 16$

Recursion | no duplicate calls.

* All calls are worthy!

Recap.

Recursion → memoization → DP

Template 1: Knapsack problem.

n items $w[] = \{$ $\}$ C

Optimization $p[] = \{$ $\}$ n^{th} item

Recursion. || you only care for 1 instance.

for n-1 items recursion will do the job.

∼ Math. induction.

Return: the max price you can make with C capacity & n items.

```
int knapsack(w[], p[], n, C):
    if C==0 || n==0      return 0
    if w[n] > C :
        return knapsack(w, p, n-1, C)
    else:
        return max { knapsack(w, p, n-1, C),
                     p[n] + knapsack(w, p, n-1, C - w[n]) }
```

(n-1 → n)

item n^{th}

$w[n] > C$: skip

$w[n] < C$: skip, pick

Variation.

① Subset sum problem.

arr = [1, 2, 7, 5, 9, 6] Sum = 13

[_, _, _, _, _, _]

pick skip

✓✓ $O(n \cdot sum)$ ← can I make it ? → Yes/No.

$O(2^n)$ ← which set makes 13 ? → Backtracking !!

```
bool subset_sum(arr, n, sum):
    if sum==0 : return true      if n==0 : return false
    if arr[n] > sum: return subset_sum(arr, n-1, sum)
    else: return subset_sum(arr, n-1, sum) || subset_sum(arr, n-1, sum-arr[n])
```

② Equal partition problem arr = [1, 3, 7, 9, 11, 6] ⇒ sum has to be even.

Whether it is possible to split the array into 2 parts?

[ ] [ ]
set1 set2
↓ ↓
sum sum

subset_sum(arr, n, $\frac{\sum arr[i]}{2}$)

③ Assign signs +ve or -ve to numbers. Target sum = target sum

arr = [1, 3, 4, 5] target sum = 3

-1 +3 -4 +5 = 3

sum = 0

sum → sum += arr[n]
sum -= arr[n]

+ve { 3, 5 } set1 -ve { 1, 4 } set 2

if sum == target_sum | done

produce the output where signs are given.

↳ backtracking. | print all possibilities

o/p: [1 1 1 -1], [1 1 -1 1], [1 -1 1 1], [-1 1 1 1]

[1, 1, 1, 1] Sum = 2

Count: 4

#ways

#possibles

Just the count ⇒ dp.

ts

```
if (target_sum == sum)
    count += 1

if n == arr.length
    return 0

ts(arr, n+1, sum + arr[n])
ts(arr, n+1, sum - arr[n])
```

n == 0

n == arr.length

(4) minimum absolute difference.

↳ variation | equal sum partition.

arr = [a1 a2 a3 a4 .. an]

{ } - { }

set 1 | set 2

|sum(set1) − sum(set2)|

→ minimum.

How ??

{ }

(set1) − (set2) | ==0 or not.

S = Σ arr[i]

S1 = sum(set1)

S2 = S − S1

[1, 9, 2, 2, 5]

{9} − {1, 2, 2, 5}

sum 9 | sum 10

1.

min absolute diff = |S1 − S2|

= |S1 − (S − S1)|

min. absolute diff = |S − 2S1|

S = known constant.

```
int subset_sum (arr, n, current_sum, min_seen_so_far) :
    if n == 0    return 0;
```

nth number

```
    int a = subset_sum (arr, n-1, current_sum + arr[n], min_seen_so_far)    // pick
    int b = subset_sum (arr, n-1, current_sum, min_seen_so_far)             // skip.
    return min (|S - 2*a|, |S - 2*b|)
```

Knapsack. : → n items C capacity max price. (Template)

↳ subset sum

↳ equal sum partition

↳ target sum

↳ min absolute diff.

Bounded Knapsack.

Assumption : You can pick one item only once.

What if each item can be picked multiple times !!

change = 6

denominations = {1, 3, 4}

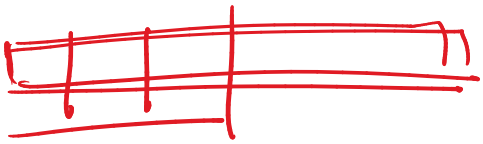
min coins that form the change.

{3, 3}

Unbounded knapsack.

Offically knapsack is over !!

rod l = l



length: 1 2 3 . . . l

price: p_1 p_2 p_3 . . . p_l

n ==

pick { rod_cut (length, price, l - length[n], n) + p[n]

skip { rod_cut (length, price, l, n-1) }

max

DP Recursion + Memoization → DP. Template 0: Fibonacci
1: Knapsack.

Template 2 : Longest Common Subsequence (LCS)

Q. $s_1 = aggtab$ (0, 4, 5) ⇒ subset of chars of s1 that is also present in s2 in same order.

$s_2 = gctaayb$ (3, 4, 6)

{a a b} ⇒ length = 3. longest ? ✗

{g t a b} ⇒ length = 4 longest

O/p : 4

aba
0 2 5
4 3 5

s_1 = a b b t a → {a, a} ⇒ 2

s_2 = e d c b a a → {b, a} ⇒ 2

a - g
a b c d e f g

length (LCS) = 2

$i_1\ i_2\ i_3 \dots i_n$ chars from s_1

$j_1\ j_2\ j_3 \dots j_n$ chars from s_2.

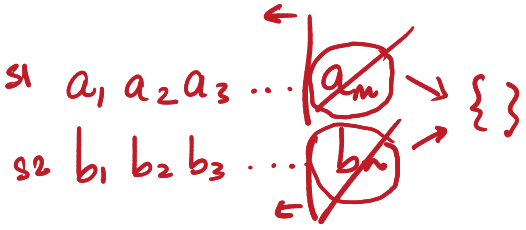
$j_1 < j_2 < j_3 \dots < j_n$

```
int lcs(s1, s2, m, n):
    if m == -1 || n == -1:
        return 0
    if s1[m] == s2[n]:
        return 1 + lcs(s1, s2, m-1, n-1)
    else:
        return max{lcs(s1, s2, m-1, n),
                   lcs(s1, s2, m, n-1)}
```

→ DP.

s_1 ← length m
s_2 ← length n

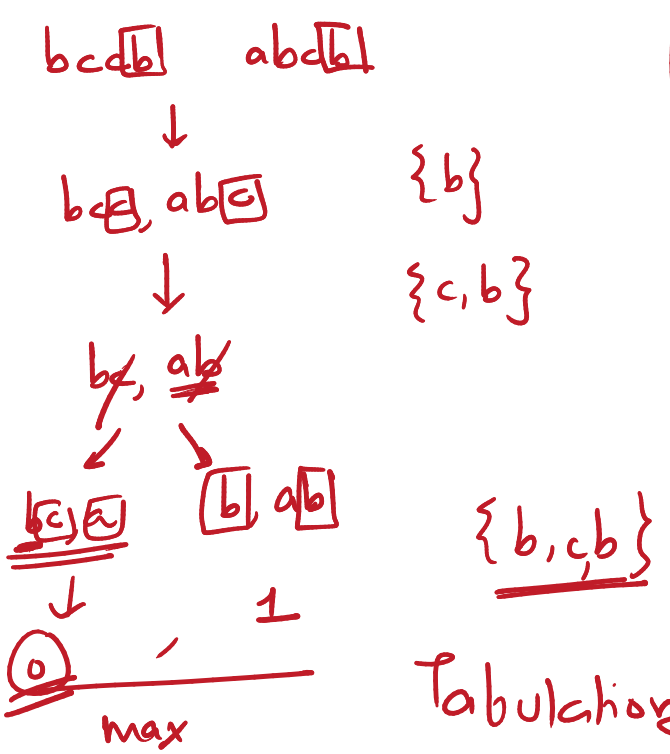
str: 0 to m-1
0 to n-1

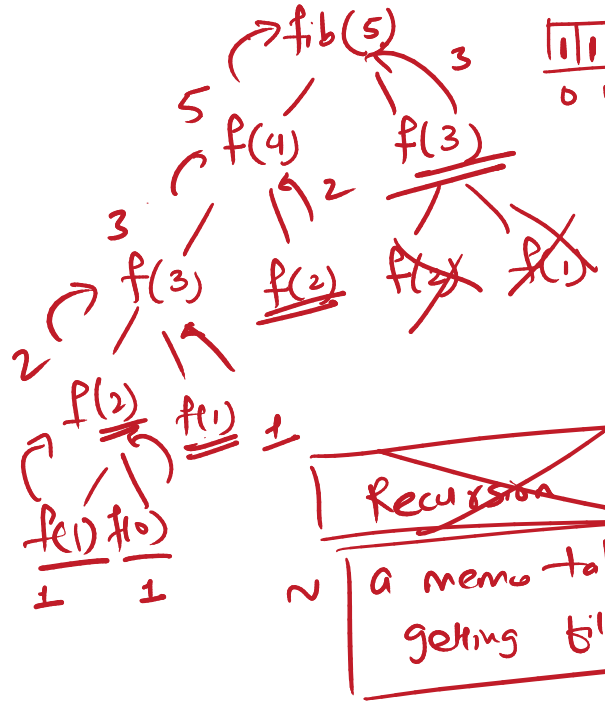


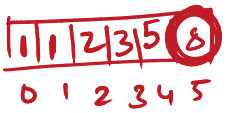
choice.

= pick the char

!= s_1 char







Recursion happens

a memo table is getting filled.

Tabulation ← Can I fill the memo table?

Instead of making recursion you go for just lookup.

fib[n+1]

fib[0]=1 fib[1]=1

for (int i=2; i<=n; i++)

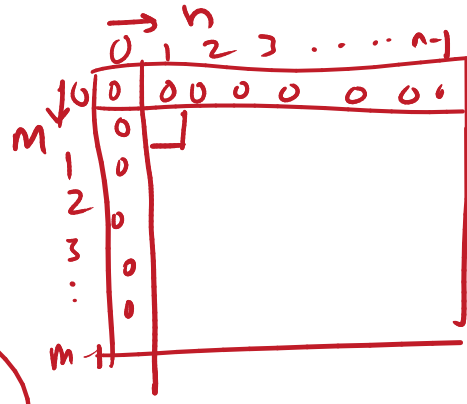
fib[i] = fib[i-1] + fib[i-2]

Cond^n:

memo(n+1)[m+1] is still recursive

return fib[n]

lcs (s1, s2, m, n) if m==0 || n==0 return 0



if (s1[m] == s2[n])

~~ans = 1 + lcs(s1, s2, m+1, n+1)~~

else

ans = max(lcs(s1, s2, m+1, n), lcs(s1, s2, m, n-1))

memo[m][n] = ans.

for i=1, m+1

for j=1, n-1

memo[i][j] = 1 + memo[i-1][j-1]

else:

memo[i][j] = max(memo[i+1][j], memo[j][i-1])

Ideally :

AC

{ Recursion / Relation } → memo table → { Memorization } → fill it memo table yourself → Tabulation

This is not bad too!

Most efficient DP soon

Theoretically TC ↔ logically

Variations

① s_1 = a b c d e f

s_2 = d b c b c d e a

length = 4.

longest common substring

sliding window technique.

```
lcsubstring (s1, s2, m, n, ~~max len~~) :
    if m==0 || n==0          return 0
    if s1[m] == s2[n] :
        max_len = max (max_len, 1 + lcsubstring (s1, s2, m-1, n-1))
        return max_len.
    else
        return 0
```

Longest common subsequence (LCS) {abda} {bada}

s1 = abbacdeaf
s2 = babeda
} given 2 strings find length of LCS.

Recursion

LCS

```
if (m==0 || n==0)
    return 0
if (s1[m]==s2[n]) and m!=n
    return 1 + lcs(m-1, n-1, s1, s2)
else
    return max(
        lcs(m, n-1, s1, s2),
        lcs(m-1, n, s1, s2)
    )
```

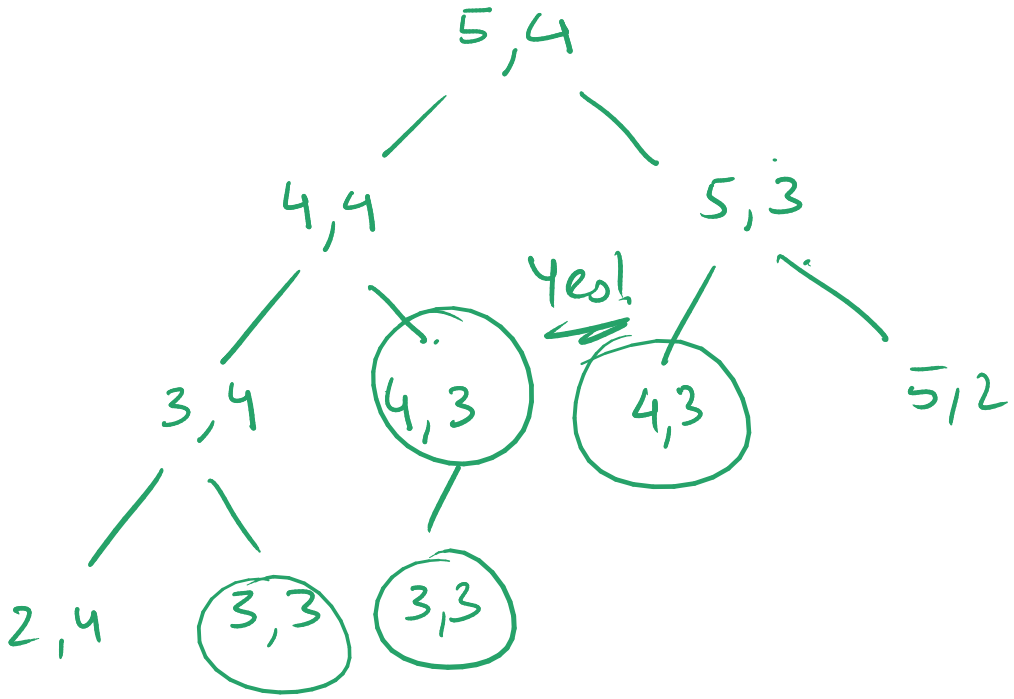
abbacdeaf
babeda

abbacdea
babeda

abbacdea
babed

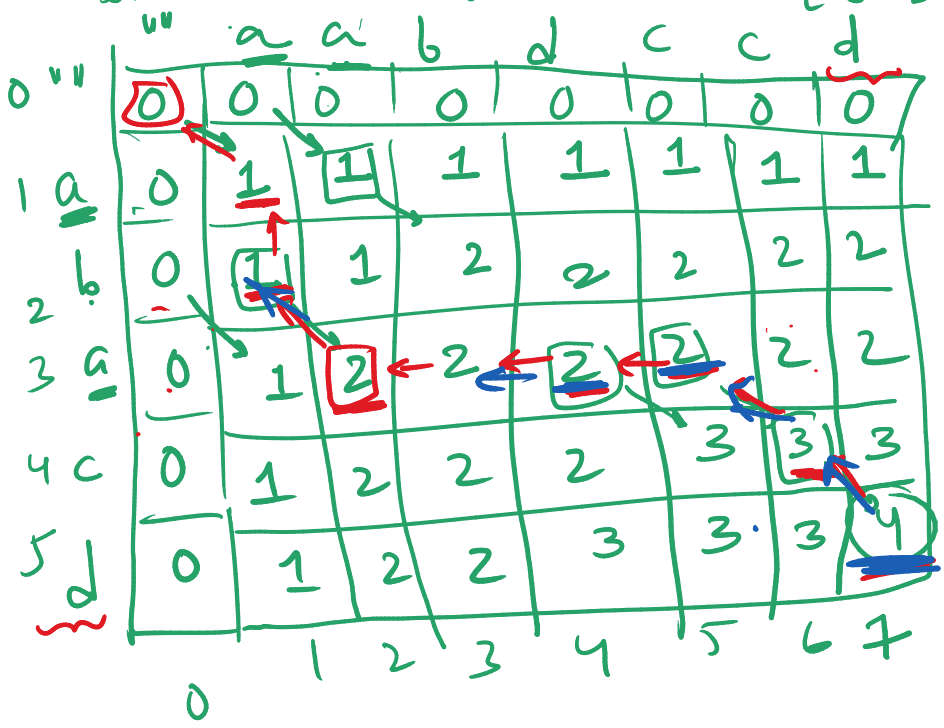
Am I doing any redundant work?

int memo[m][n]
= {-1}



Variations s1[m] == s2[n]

Entries in the memo table. memo[m][n]



#1: Print lcs.

5 s1 : a b a c d (5)

7 s2 : a a b d c c d (7)

4

memo[5][7]

ans = 4

length of lcs

aacd

7+5-4 = 8

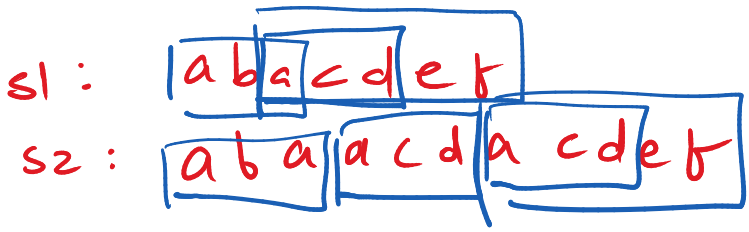
a a b d a c c d

lcs( ) ⟶ memo[ ][ ]

```
int i = m, j = n;
ans = ""
while (i > 0 || j > 0):
    if s1[i] == s2[j]:
        ans += s1[i]
        i--; j--;
    else:
        if j > 0:
            j--
        else i--
ans.reverse()
```

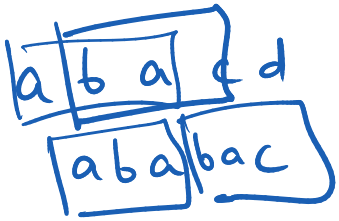
} put unequal chars as well.

Variant 2 : Longest common substring.



ans = acdef

length = 5



Longest common substring

```
if m==0 || n==0 :
    return 0
if s1[m] == s2[n]
    return 1 + lcs(m-1, n-1, s1, s2)
else
    return 0
```

```
lcs(m, n, length, s1, s2):
if (m==0 || n==0)  return length
```

Tabulation

```
if (s1[m] == s2[n])
    return lcs(m-1, n-1, length+1, s1, s2)
```

Recursion

```
else
    return max( length, lcs(m-1, n, length, s1, s2)
              , lcs(m, n-1, length, s1, s2) )
```

Variant 3: shortest common supersequence (SCS)

5 s1: a b a c d
4 s2: b c d e

lcs ⇒ b c d ⇒ 3

s = a b a c d e

subsequence of s

len(t)

length of SCS

5 + 4 − 3 = 6

all characters from s1 & s2 but lcs only once.

$$|s1 + s2 - lcs|$$

Variant. #4

Min. number of insertions + deletions to convert s1 to s2

s1: heap
s2: pea

ans = 3

h p
ea

heap → hea → pea

s1 —(all chars from s1 that are not in lcs)→ lcs —(all chars unique to s2 → s2 insertions)→ s2

$$|4 + 3 - 2(2)| = 3$$

$$|s1 + s2 - 2\,lcs|$$

variant 5 Longest palindromic subsequence

s = a g b c b a

length = 5

a b c b a → palindrome

s1 : s agbcba } LCS abcba

s2 : s.reverse() abcbga

variant 6 : s → min. insertions/deletions to make it a palindrome

s = a g b c b a

g

a g b c b g a

SCS of s and s.reverse() (1)

s1 : agbcba

s2 : abcbga

s:

s 6 → ans SCS 7

2(6) - 5

(7)

variant 7

Maximum Repeating subsequence
Longest (LRS)

s = a a b e b c d d

LRS = 3

a b c d

s1 : s
s2 : s
⇒

a a b e b c (m)

a a b e b c (n)

↙ ↘ —

Recap: DP → what it is ?
→ Templates: 1) Knapsack 2) LCS. 3) MCM. } Variations

CPU : addition is easy multiplication takes time.

Matrix Chain Multiplication.

Matrix multiplication.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}_{2\times 2} \begin{bmatrix} a \\ b \end{bmatrix}_{2\times 1} = \begin{bmatrix} 1a+2b \\ 3a+4b \end{bmatrix}_{2\times 1}$$

multiplications = 4

facts: $a\times b \cdot b\times c \Rightarrow a\times c$ final size

ops : multiplications

$a\times b \cdot b\times c \Rightarrow a\times b\times c$ no. of multiplications

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}_{2\times 2} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}_{2\times 2} = \begin{bmatrix} 1a+3b & 2a+4b \\ 1c+3d & 2c+4d \end{bmatrix}$$

3 matrices. $\underbrace{2\times 2}_{A}$ $\underbrace{2\times 1}_{B}$ $\underbrace{1\times 4}_{C}$ ⇒ output 2×4

Question: $(AB).C$ (AB: 2×1) OR $A.(B.C)$ ($B.C$: 2×4)

Size: Both cases 2×4

ops.

$(AB).C$: $2.2.1 + 2.1.4$

$= 4 + 8$

$= 12$ ✓

$A.(B.C)$: $2.1.4 + 2.2.4$

$= 8 + 16$

$= 24$ ✓

We know only multiply 2 matrices at a time!!

Question: $n = 4$ matrices

Order is imp!!

$(AB)C \neq (AC)B$

A B C D ⇒ an arrangement that gives minimum # multiplications to get the final output.

$((AB)C)D$ OR $(A(BC))D$ OR $A((BC)D)$ OR $((AB)(CD))$

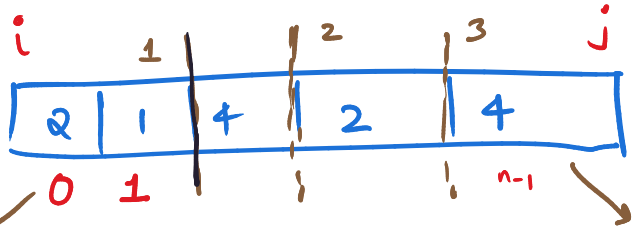
i/p: 2x1 1x4 4x2 2x4 — n — n-1 matrices

arr: 2 1 4 2 4 | 2 | 1 | 4 | 2 | 4 | arr dimensions.

o/p: # min no of multiplications required to do chain multiplication

multiply {n-1 matrices} ⟹ 1 ans matrix.

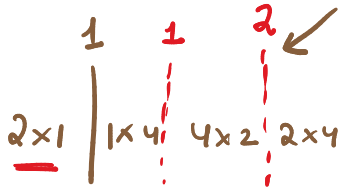
Dry run 0,4



k = 1 to n-1

start your splitting

end your splitting



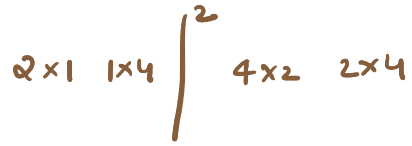
A {B C D}

1 ↓

C_1 A {B (C D)}

C_2 A {(B C) D}

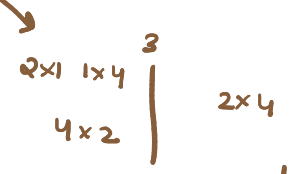
{min C_i}



{A B} | {C D}

C_3 (AB)(CD)

4 matrices
5 ways



{A B C} | D

1 2

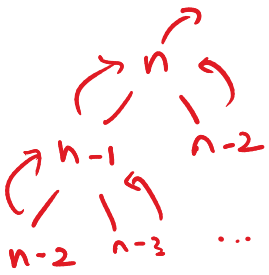
C_4 {A (BC)} D

C_5 {(AB) C} D

fib (n):

fib(n-1)

fib (n-2)

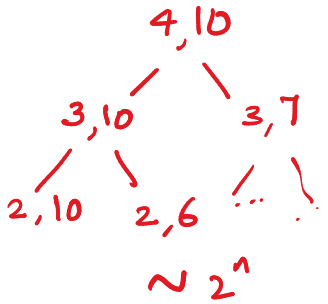


~ 2^n

knap(n, C):

knap(n-1, C)

knap (n-1, C-w)



~ 2^n

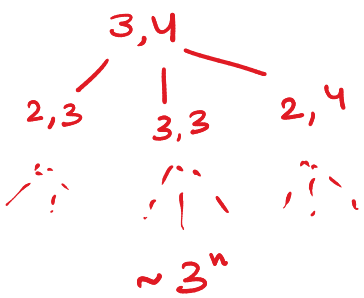
lcs (m, n)

lcs (m-1, n-1)

lcs (m, n-1)

lcs (m-1, n)

} if else control these recursive calls.



~ 3^n

2D Recursion

Plane Recursion

reducing the input size in recursive calls arithmetically

mergesort (n)
mergesort (0, n/2)
mergesort (n/2, n)

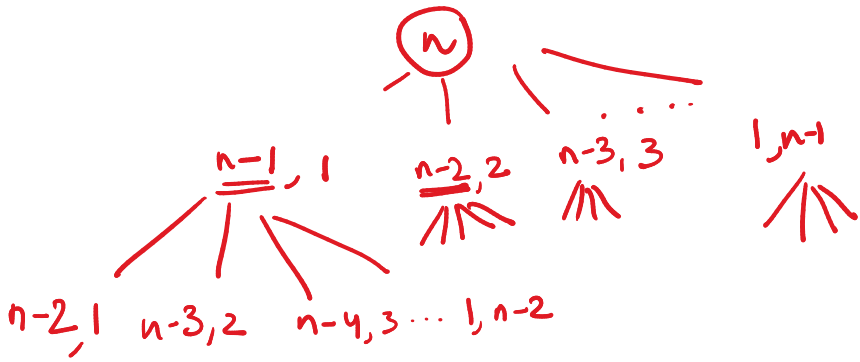
quicksort (n)
quicksort (0, i)
quicksort (i+1, n)
i ⇒ partition index

These are plane recursion.

Size reduces geometrically,
TC is logarithmic.
$\log_2 n$

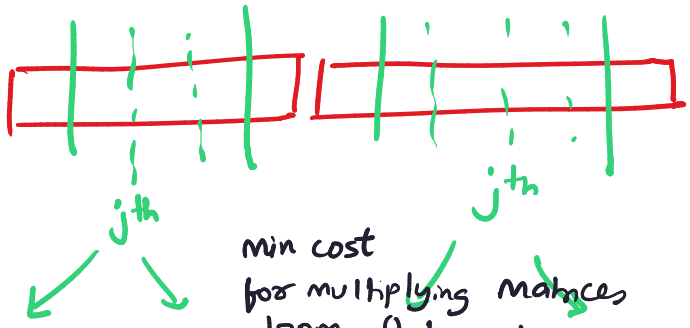
$\sim \log_2 n$

f(n)
for loop of i.
f(n-i)
f(i)



Highlight :

f(5)
f(1) f(4)
f(4) f(1)



min cost for multiplying matrices from 0 to n-1
mcm (arr, 1, n-1)

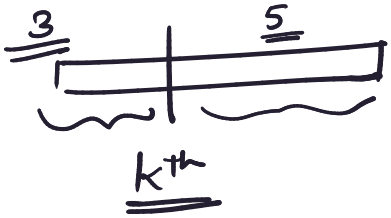
mcm(0, n-1):
k : {1, n-1}
a = mcm(0, k)
b = mcm(k+1, n-1)

Think INDUCTION !!

Structure / template of MCM :

```
int mcm (arr, i, j):
    if (i >= j) : return 0      // If single matrix, no cost to mult
    int ans = 10e8
    for k = i , k < j , k++ :
        int temp = mcm(arr, i, k) + mcm(arr, k+1, j) +
                   arr[i-1] * arr[k] * arr[j]
        ans = min (ans, temp)
    return ans
```

DONE



8

7 matrices

A B C D E

ABC DE

n = 4

↓

5 possibilities

f(k) f(n-1-k)

n = 5

↓

14 possibilities

(1,6) (2,5) (3,4) (4,3) (5,2)

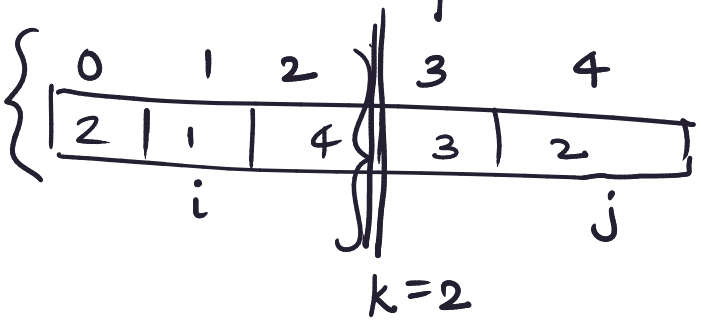
↓

min cost

$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ | $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$

0

{ABC} | {DEFG}

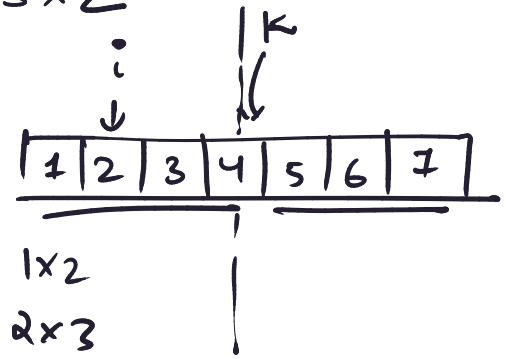


2×1

1×4

4×3

3×2



1×2

2×3

3×4

4×5

5×6

6×7

1×4 4×7

↙ ↘

mcm mcm

mcm (i, k)

2×1 1×4

i-1 k

mcm (k+1, j)

4×3 3×2

k j

arr[i-1] × arr[k]

a b

↳ abc

arr[k] × arr[j]

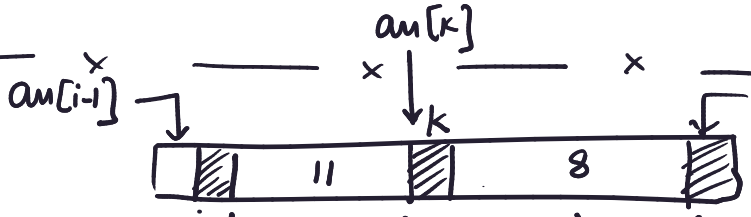
b c

— × — × — × — × — × — × —.

1×4 4×2 2×3 3×1

↓

1×1



mcm (i, k)

mcm (k+1, j)

↓

arr[i-1] × arr[k] arr[k] × arr[j]

↳ arr[i-1] * arr[k] * arr[j]

2 variations : ① Palindrome Partitioning ② Egg Drop problem

#1 s = "sagar" ⇒ min no of partitions such that each component/part is a palindrome

o/p: 2

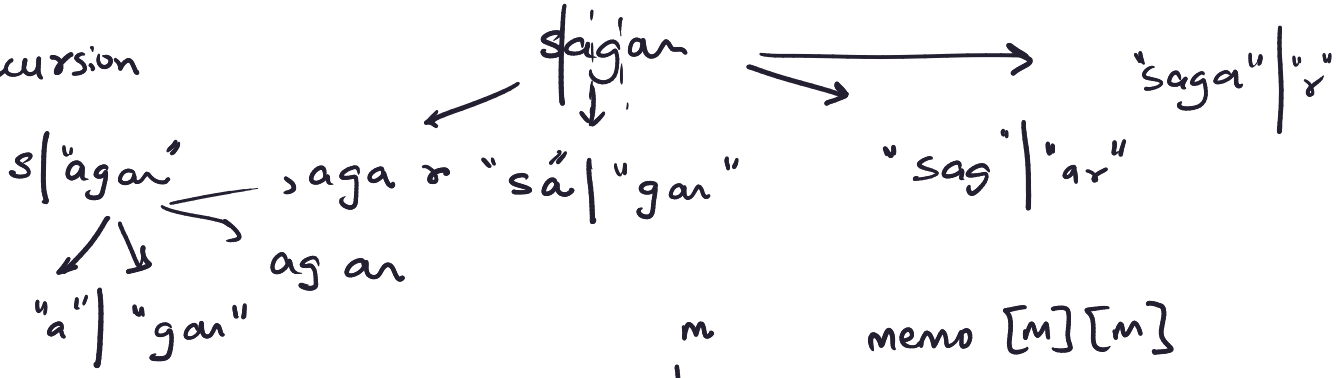
s | aga | r

s = "abcde" → a | b | c | d | e ⇒ all single chars are palindromes.

n max: n-1 partition

Q. Can I do better?

B.F. Recursion



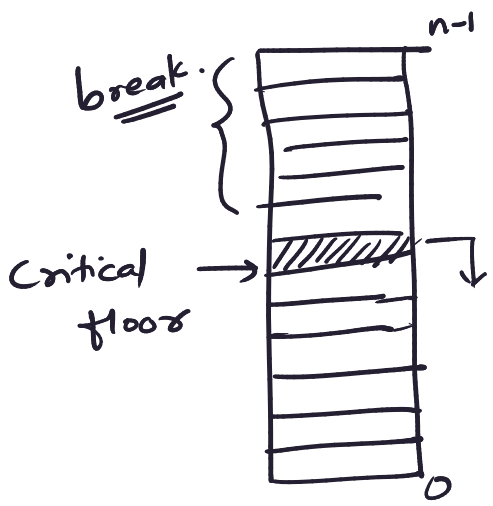
memo [n][n]

pseudo: int palindrome_partitioning (string s, int i, int j):

```
if (i >= j)  return 0;
if is_palindrome ( s.substring(i,j))  return 0;
int ans = 10e8
for (int k = i; k < j , k++):
    int temp = pp(s, i, k) + pp(s, k+1, j) + 1
    ans = min (ans, temp)
return ans;
```

#2. Egg drop problem.

n floors m eggs



critical floor ⇒ last floor where egg will survive the drop.

You dont know the critical floor.

In the worst case : min drops required to find the critical floor.

Eg. n = 10 m = 1

o/p : 10

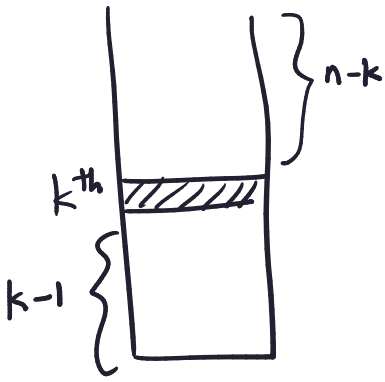
n = 10 M = 2

o/p : 5

n = 10 m = 3

o/p. I dont want to think.

```
int egg_drop ( int m, int n ) :
    if m==1          return n
    if n == 0 || n==1    return n
    int ans = 10e8
    for k = 1 , k <= n , k++
        int temp = max { egg_drop (m-1, k-1), } + 1
                       { egg_drop (m, n-k)   }
        ans = min (ans, temp)
    Return ans
```



① Fibonacci ② Knapsack ③ LCS ④ MCM — Classics.

PS1 : Min cost path on a grid
PS2 : Word break problem
PS3 : Weighted Job scheduling problem.

} ad-hoc DP problem

1. Understand the PS
2. Reason – why recursion & lets write bruteforce recursion
3. Understand why DP ? Overlapping problems.
4. Memoize it to make it a DP sol?

PS1: Min cost path on the grid.

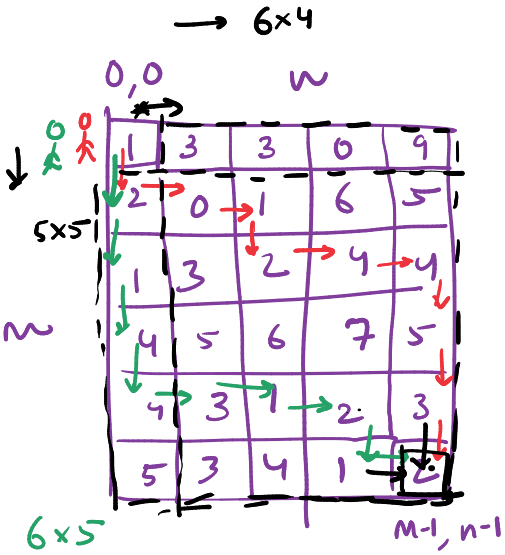
* 1 2 0 1 2 4 4 5 3 2

cost : 24 min. cost ?

greedy ✗

1 2 1 4 4 3 1 2 1 2.

cost : 21 min cost = 21



② 0,0 → m-1, n-1 order ✗

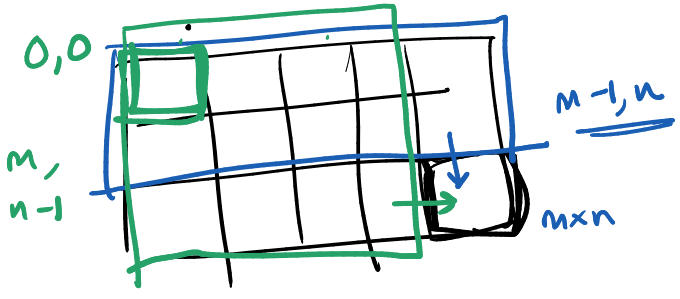
m-1 down steps
n-1 right steps.

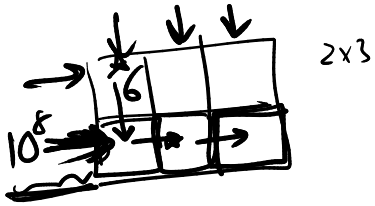
break down problem into smaller chunks.

fact(n) = n . fact(n-1)

fib(n) = fib(n-1) + fib(n-2)

ans (6×5) = min(R + ans (6×4) , D + ans (5×5))





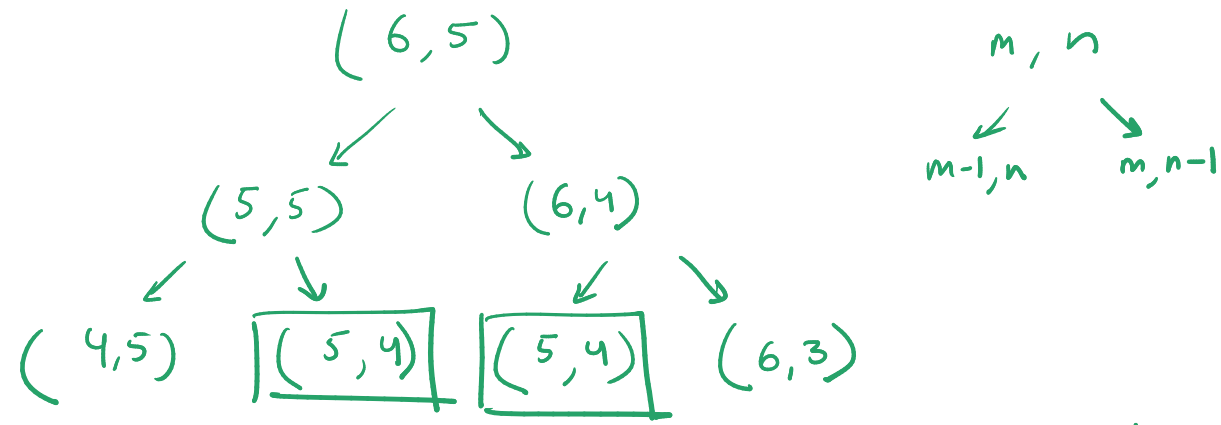
```
int min_cost ( vector <vector <int> > grid , int m, int n) {
    if (m < 0 || n < 0)                    // Boundary cond?
        return 10e8 ;
    if (m == 0 and n == 0)                 // reached start cell.
        return grid [m] [n] ;              // grid[0][0]
    else
        return grid [m][n] +               // m, n
        min ( min_cost ( grid, m-1, n) ,
              min_cost (grid, m, n-1) ) ;
}
```

if memo[m][n] != -1 return memo[m][n]

ans = grid [m][n] + min (min_cost (grid, m-1, n) , min_cost (grid, m, n-1)) ;

memo [m][n] = ans
return ans

③



DP.

memoize

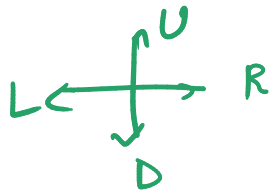
same recursion calls ∴ redundant work !!

int memo [rows] [cols] = {-1}

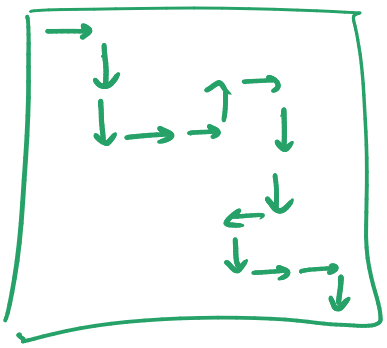
④ . Memoize ⟶ ✓ DP problem

Key : There is no backward possibility here !!

If you can : its not a DP problem.



Graph problems.



2nd Word Break Problem

m ✗ my ← (n ... t) True("")

no overlapping words !!

str = "mynameissagarandiamadatascientist"

list_of_words = ["my", "name", "is", "a", "sagar", "and", "i", "am", "data", "scientist", "soft", "dev", "mad", "ent"]

① Yes/No.

Q. Can you break the string such that each part of it is a valid word ? ⟹ looping → ✗ single O(n) ✗

② ans

s_1 s_2 s_3 s_4 s_5 s_6 ..... s_{n-1} s_n.

s_1 | s_2 . ans($s_2 \dots s_n$)

no

$s_1 s_2$ $s_3 s_4 s_5 \dots s_n$

no yes.

```
is_valid(str)
{ if this str is in list of words }
```

Recursive

Valid word

if recursively can I break the rest of string.

false

carpet

{can ⇐ carpet}

True

carpet | ""

1. question ✗ wrong method picked.

{Intuition overlooked, premature building} NOTE IT DOWN !!

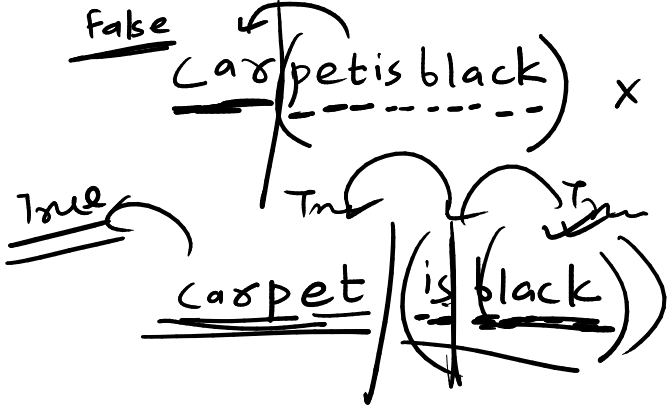
2. Approach {Theory}

Recursion {Logic}

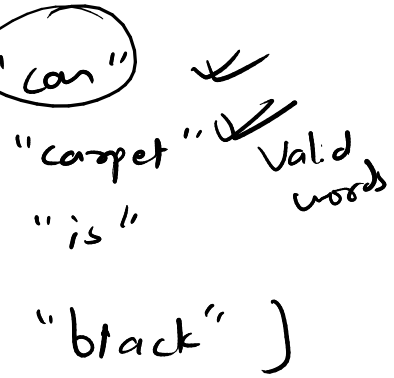
Unable to code it.

Strengthen language → {"Practice"}

Question. I thought Y approach because of XYZ logic. However XYZ is not right because of ABC. ∴ X approach



premature breaking




```
bool is_valid ( string word) {
    list of words = ["", ".", " ", "  ", ]
    for (auto x : list_of_words) if x == word return True;
    return False;
}
```

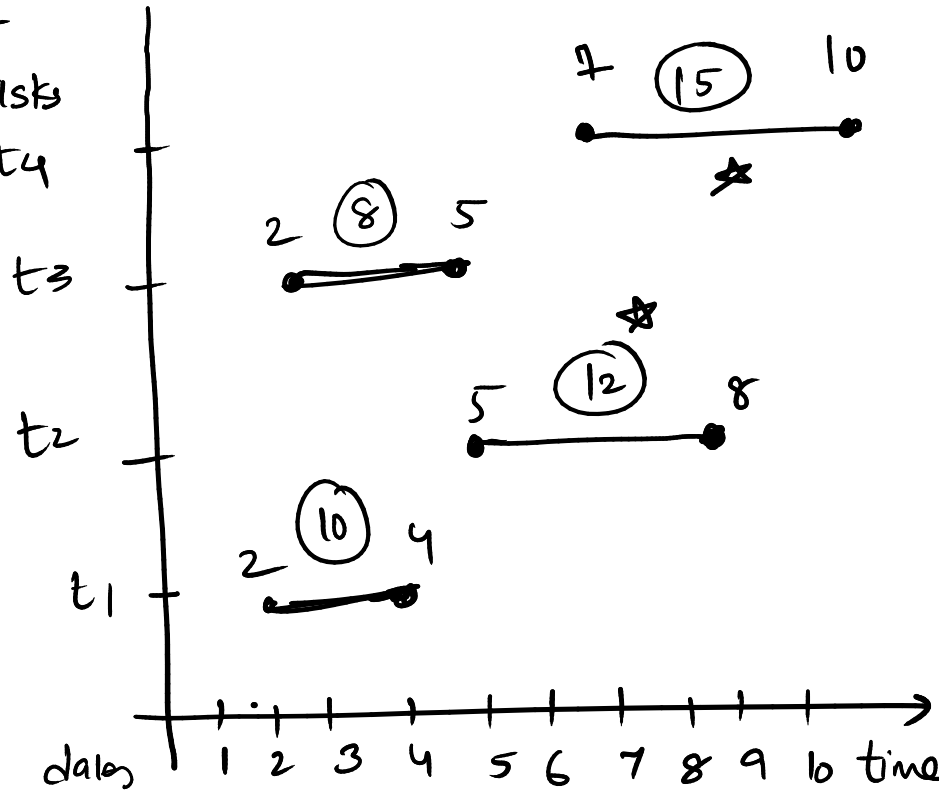
set can be better

substr(start, no.of chars)

```
bool word_break (string str) {
    int size = str.size();
    if (size == 0)  return true ;   // base cond?
    for ( int i=1 ; i <= size; i++)
        if ( is_valid ( str.substr (0, i) and word_break(
                                    str.substr(i, size-i))
            return True
    return False
}
```

Gantt charts

t: start date
end date
value



Q. What tasks should I pick to max. value impact?

$[(2,4,10)(5,8,12),(2,5,8),(7,10,15)]$

max value $\Rightarrow$ $10+15$ = 25

Q.1 n piles of coins ⇒ arr. 2 questions 75 mins.

a = [1, 3, 2, 2, 5] ans = 5

1 move: +1 coin to any pile
-1 coin from any pile

OR

$n = 10^5$ $a_i = 10^6$

In how many minimum moves, you can make all piles equal?

Soln: ① mode ⇒ most frequent one & match all elements to that.

unique → ??

X [1, 3, 2, 2, 5] mode = 2

+1 -1 0 -3 ← 5

② median ⇒ sort, find mid & bring everything to mid.

[1, 2, 2, 3, 5] median = 2.

③ mean ⇒ sum of all = 13 piles = 5

why floor? on avg = 2.6 coins $\frac{13}{5} = 2$

why not ciel? 2 or 3

a_i = [1, 3, 2, 2, 5] range of numbers = {1, 5}

[1, 100]

x_i	1	2	3	4	5
c_i	8	5	6	9	12

a_i = ____ _ _ _ _ $n = 10^5$ sort (a) → min = a[0]

max = a[n-1]

nlogn

l = min, r = max 1, 10^6

mid = $\frac{l+r}{2}$

while (l <= r)

$O(3n \log(a_{max}))$

Ideal

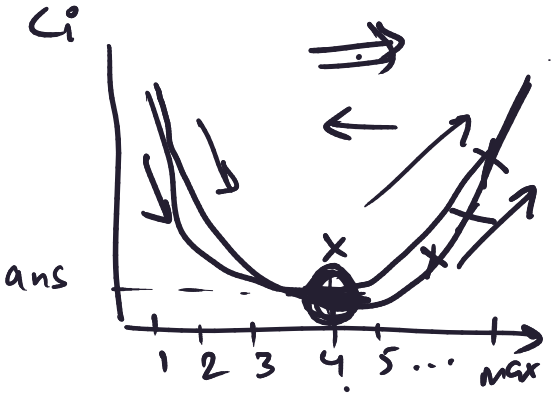
```
cost(arr, value){
  for (auto x : arr)
    cost += abs(x - value);
  return cost;
}
```

{

cost >= cost(mid) <= cost(mid+1)

[m-1]

return mid.



cost(mid-1) >= cost(mid) => cost(mid+1)

l = mid + 1

cost(mid-1) <= cost(mid) <= cost(mid+1)

r = mid - 1 } ans = mid

B.F.

min_cost = INT_MAX;

for (i min, max of arr)

cost_i = cost(arr, i)

~~cost_x~~ = min(cost_i, min_cost)

min_cost

cout << minCost << endl;

15 t

75 min → 30 min

→ 45 min

T.C.

$O(n \cdot a_{max})$

T.C. Actual

$O(n \log a_{max})$

~1000s

$n \to 10^5$

$a_{max} \to 10^6$ 10^8 10^{11}

AC

Q.2 n chefs d dishes

time[i] = i^{th} chef takes to cook 1 dish.

min time required to get all d dishes prepared.

Eg: 3 (5)

time per dish: 2 3 5

o/p: 6

5 mins? ✓

A: 2
B: 1
C: 1

4 dishes ✓

6 mins:

A: 3
B: 2
C: 1

6

only fastest chef works

Soln

min_time, l = 1

max_time, r = $d * a_{min}$

times { mid = $\frac{l+r}{2}$

while (l <= r)

if (count(mid, times) > d)

r = mid - 1

else

l = mid + 1

ans = mid;

}

count(t, times) {

dishes = 0 O(n)

for auto x : times

dishes += t/x

return dishes;

}

AC

T.C.

$O(n \log(d \cdot a_{min}))$

BS (but BF worked)

2 questions → Both AC → BS

Idea: OA : 2 questions : Both arrays !! (sort + search)

APPLY !!

SDE1 ↙ DSA

0-3 yoe → ?? ↓ Sys design ✗ 12-18

3-7 yoe SDE2 ↓ DSA + Sys Design "18-28"

?? 7+ yoe SDE3 28+

Connection 100 companies

ABC → careers each = each team

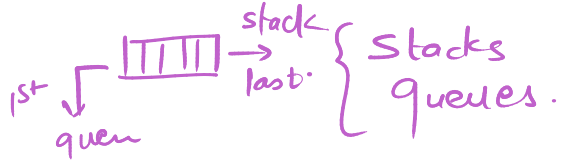
Microsoft

SDE → 12-14 teams

Opp

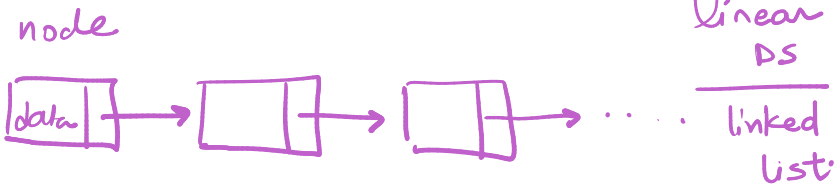
12 teams No !!

① Applications. Classic ones.

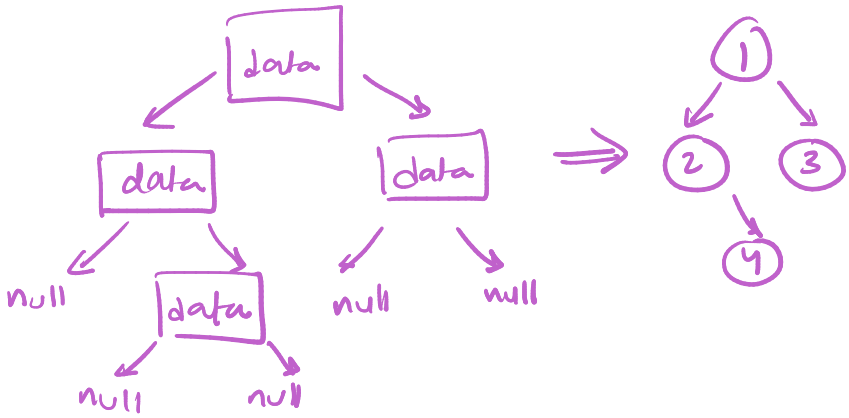


Concept of trees (binary) ← stacks & queues.

```
node {
    int data ;
    node* next ;
}
```



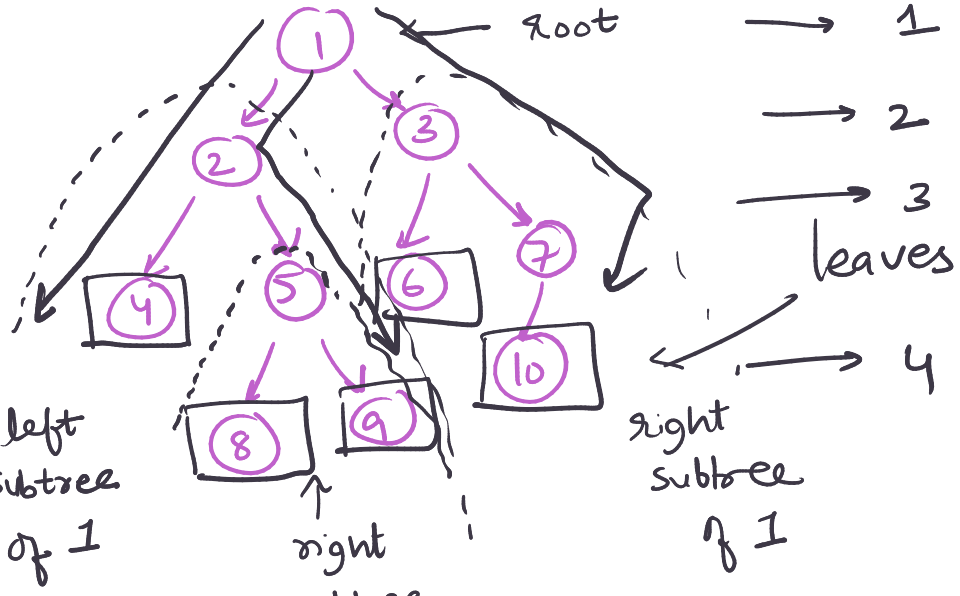

```
node {
    int data;
    node* left
    node* right
}
```



Terminology.

Binary Tree

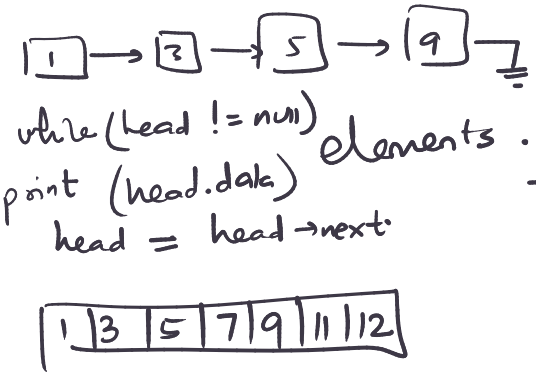
every node ↓ 2 nodes left right



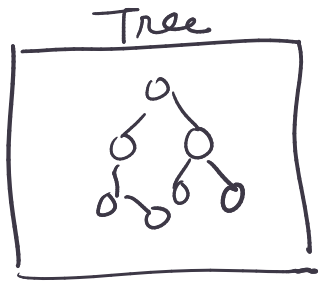
4 levels

height of tree is 4.

root to leaf path ↳ branches



while (head != null) elements.
print (head.data)
head = head → next



You should be able to touch every element.

traversal

↓

whatever i/p I gave you, tell me all of them.

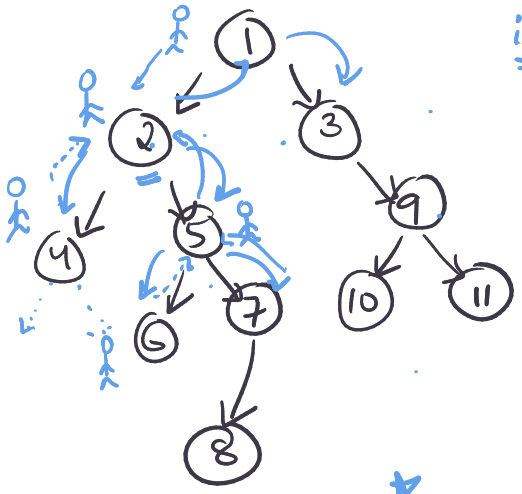
i ⟶ Traverse

```
print()   for(i in 0,len(arr)):
              print(arr[i])
```

Print all elements.

non linear. How?

- breadth first traversal (BFT) →
  - Level order traversal
- depth first traversal (DFT)
  - Pre order
  - In order
  - Post order



in: 4 2 6 5 8 1 3 10 9 11 (left, root, right; left, root; right)

post: 4 6 8 7 5 2 10 11 9 3 1

pre: 1 2 4 5 6 7 8 3 9 10 11 (root, left, right; root, left, right)

★ level: 1 2 3 4 5 9 6 7 10 11 8

★ pre: root, left subtree, right subtree
in: left subtree, root, right subtree
post: left st, right st, root.

Stacks. We follow BODMAS rule

a + b × c + (d + e) / f

How2 exp = "3 + 4 × 5 + (2 + 3) / 5"

integer o/p ⇒ value of this expression.

Traverse: 24

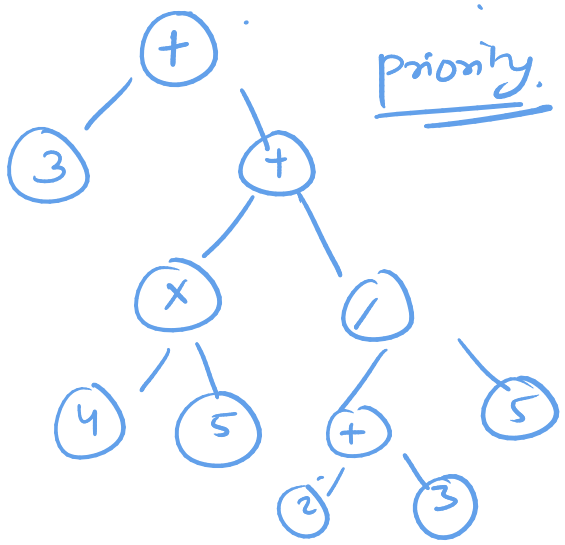
3 + 2 × 3 → 5 × 3

3 + 6 → 9

× 15

logic : Infix way of writing expressions.

all numbers ⇒ leaf nodes
all operators ⇒ internal nodes.



priority.

Inorder : left root right

→ 3 + 4 x 5 + 2 + 3 / 5

infix ⇒ lose the priority of ()
↳ human readable

Postorder. left right root

→ 3 4 5 x 2 3 + 5 / + +

postfix

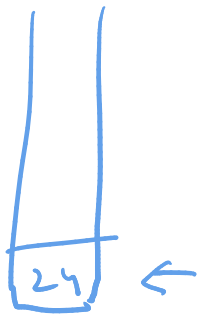
Preorder: root left right

→ + 3 + x 4 5 / + 2 3 5

prefix

logic Using stacks.

1 digit :



logic : Postfix

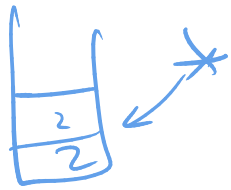
Push numbers in stack

whenever you see operator
↳ last 2 element, operate
push the result back.

postfix ⇒ left right root.

number number operator

stacks



Actionables

1. Given an infix expression (brackets) ⇒ Convert postfix
2. Use a stack, push numbers, see the operator operate on last 2 numbers, put the result back in stack.

String "345"
↳ parse Int.

Online assessment | DSA questions. 90 mins 4 questions.

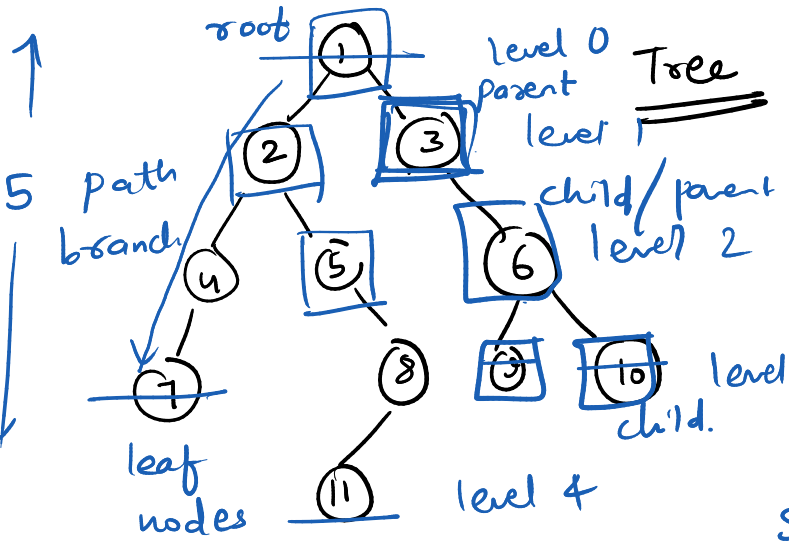
Q.1 Arrays (Searching, sorting, brute force, 2 ptr, SW)

Q.2 ?? Stacks / Strings / NumberTheory

Q.3 Trees / Graphs. If graphs, BFS / DFS on grid.

Q.4 DP problem (SWE)

Trees



Terminologies:

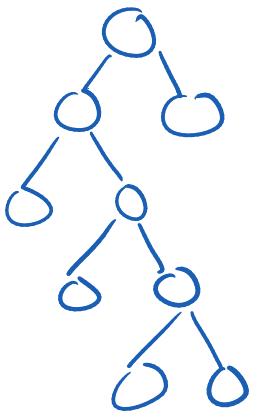
- root
- leaf
- internal nodes
- levels
- height
- branch

```
nodes {
    int data;
    Node left;
    Node right;
}
```

single parent → 0, 1, 2 children

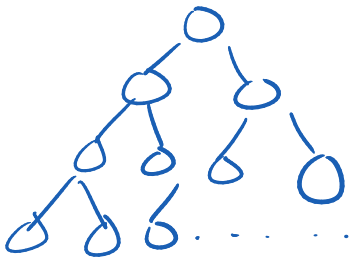
Types of Trees:

① Full tree



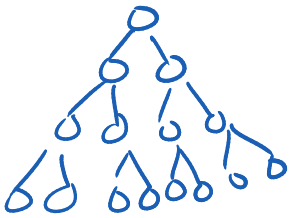
Every node has 0 or 2 children.

② Complete Tree



All levels must be complete except the last one.

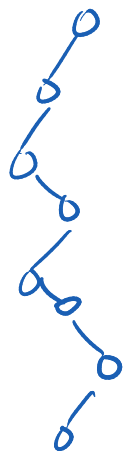
③ Perfect Tree



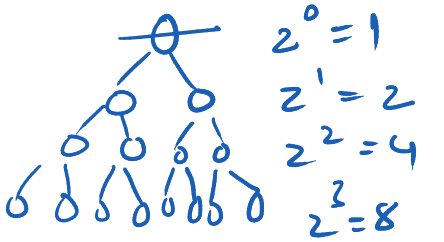
All levels are complete.

Ideal Tree

④ Skewed Tree



Only child allowed to all.



Capacity

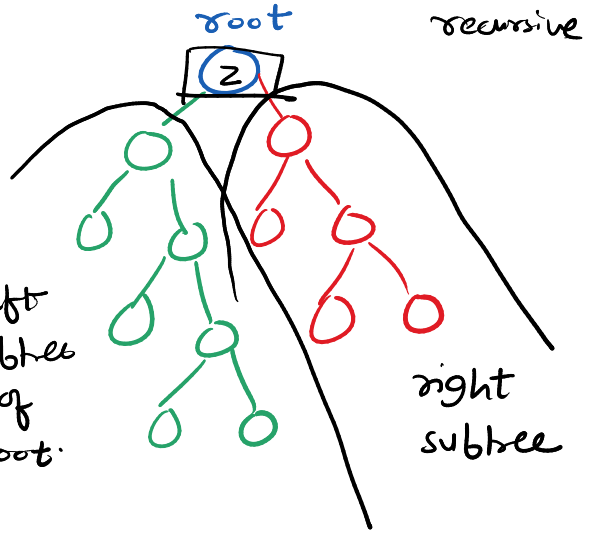
T.C

O(height of tree)

① Recursion on Trees.

Tree is a recursive structure.

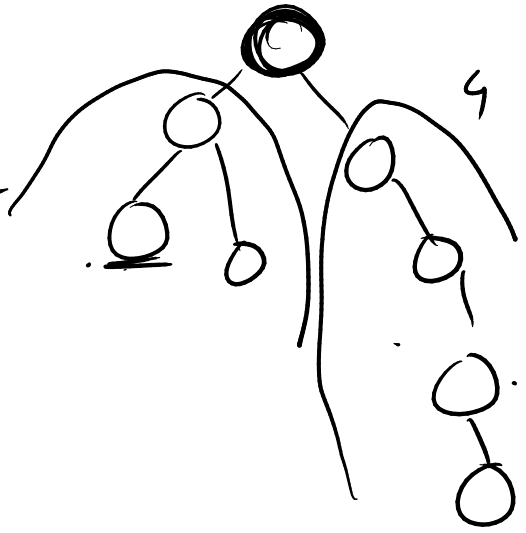
Q.1 Sum of nodes of tree



$$sum(root) = sum(root \to left) + sum(root \to right) + root.data$$

B.C If root is null, return 0

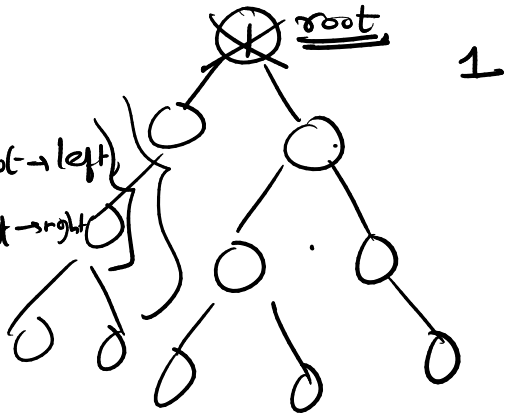
Q.2 Height of a tree



$$height(node) = \max\{height(node \to left), height(node \to right)\} + 1$$

if node == null, return 0;

Q.3 Max element in a tree



$$\max imum(root) = \max\{root, \max\{maximum(root \to left), maximum(root \to right)\}\}$$

if root = null, return 0;

Q.4 Search in a tree | key → exist or not.

search(root, key) =

if root.data == key return true
if root == null return false
search(root → left, key) ||
search(root → right, key)

Q.5 check if tree is full tree. || each child has 2 nodes or zero.

bool isFull (root) ⇒

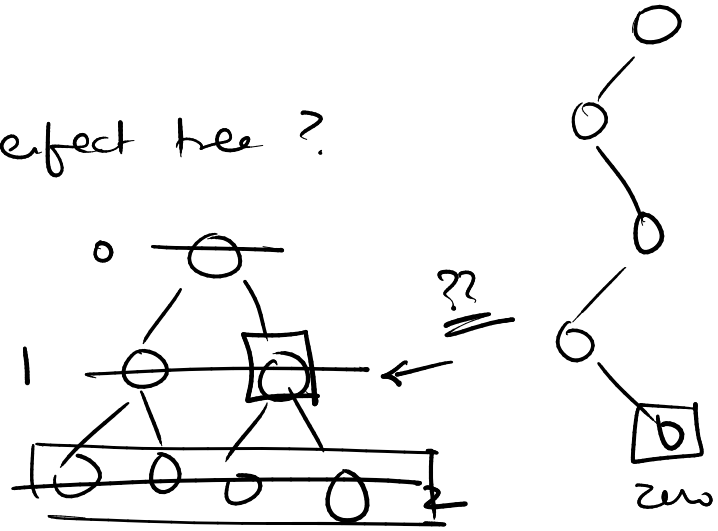
if (root→left and root→right)
return isFull (root→left) and
isFull (root→right)
else False;

Base
if (root == null)
return true
if (root→leaf==null
and root→right ==null)
return true.

Q.6 How to check if tree is skewed ?

Q.7 How to check for perfect tree ?

height ? ⇒ 3



bool isPerfect (root, height, level) {
if root==null return true;
if (root→left == null && root→right == null)
return (level+1) == height;
if (root→left ==null || root→right == null)
return false
return isperfect (root→left, height, level+1) &&
is perfect (root→right, height, level+1)

all leaf nodes are at level same (height-1)

height ⇒ h=3 7 nodes h Total nodes = $2^h - 1$

h=4 15 nodes.

count(root) = 1 + count(root→left) + count(root→right)

if (root == null) return 0;

Except the last level, all levels should be complete

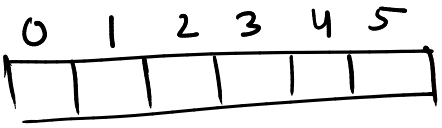
Q.8. check if tree is complete !!

bool isComplete (root, index, total_count)

if root == null return true

if index > total_count

return false.

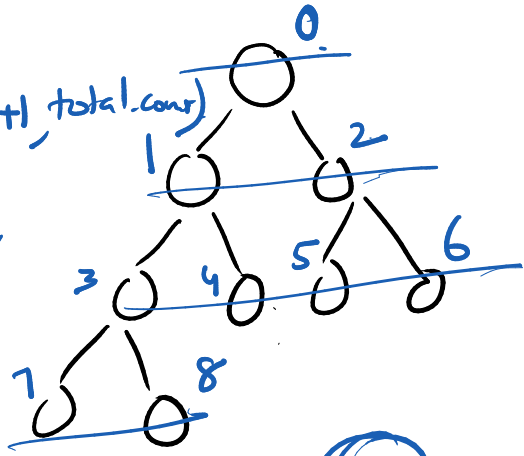


return isComplete (root→left, 2*index+1, total_count)

&& isComplete (root→right,

2*index+2,

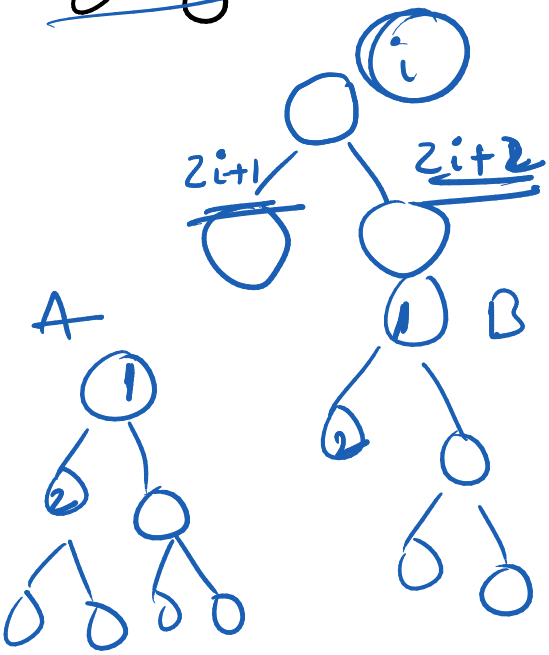
total_count);



Q.9 Is treeA identical to tree B?

bool (Root A, Root B) {

identical ⇒ Structure + values.



}

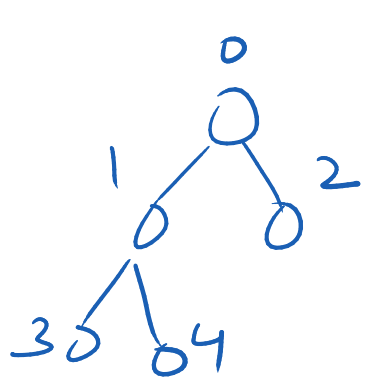
Q. 10 treeA symmetric to treeB?

→ Structure.

isIdentical (rootA, rootB)

symmetric

if rootA == null && rootB == null
return True;



if (rootA → left == null &&
rootA → right == null)
return rootB → left == null && rootB → right == null

5

if (root → left == null && rootB → left != null
&& ___ right) return false;

return isIdentical (rootA → left, RootB → left)
&& isIdentical (rootA → right,
rootB → right);

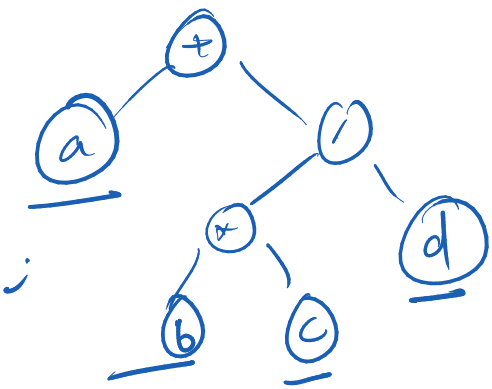
10 Round 1 Recursion codes for Tree.

Stacks: pre, in, post. → a + (b * c) / d

prefix

postfix

infix



traversal(root) {

if (root == null) return;

① traversal (root → left);
② traversal (root → right);
③ cout << root.data << endl;
}

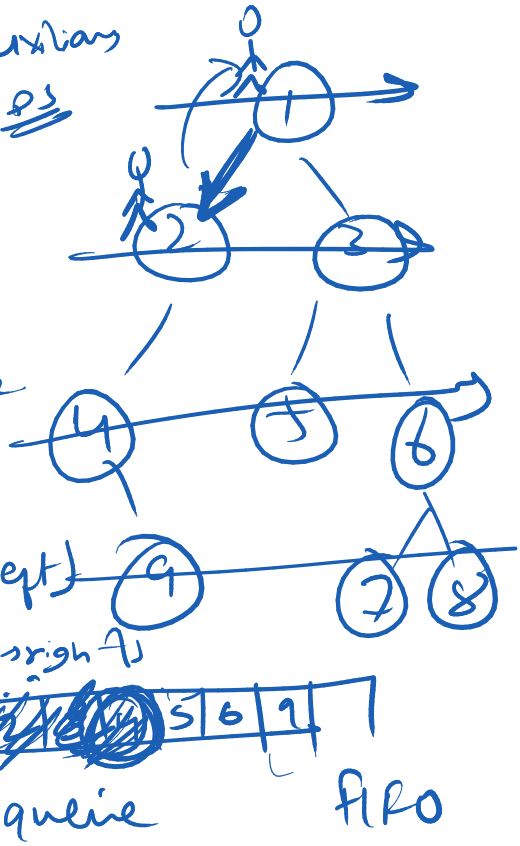
left right root : post
1-2-3

root-left-right : pre
3-1-2

1-3-2
left root right : in

Level Order (Remember it)

Auxiliary DS



print: 1 2 3 4 5 6 9 7 8 22

q

```
while (!q.empty()){
    remove the first element from queue
    // Do it. | print()
    if root->left  : q.insert(root->left)
    if root->right : q.insert(root->right)
}
```


queue FIFO

Tree: non-linear DS + Terminology

Types of trees: 4 types: full, complete, perfect, skewed.

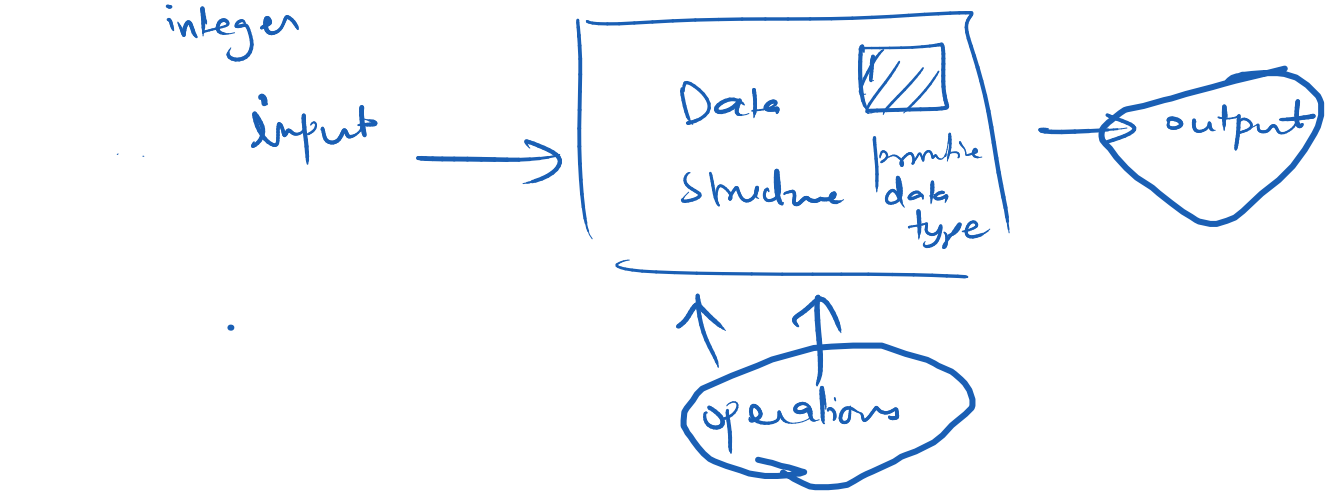
Recursive codes: (10) R1

Traversals → depth: pre_in, post (recursion) — two questions

↳ breadth: level order (queue) Imp template

8 questions

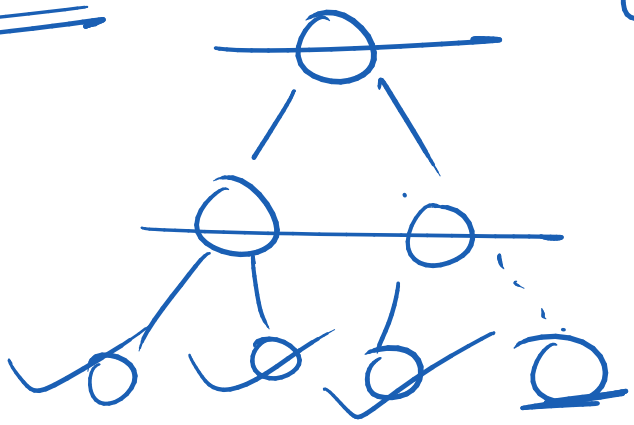
Trees ← CRUD



① Create / Insert

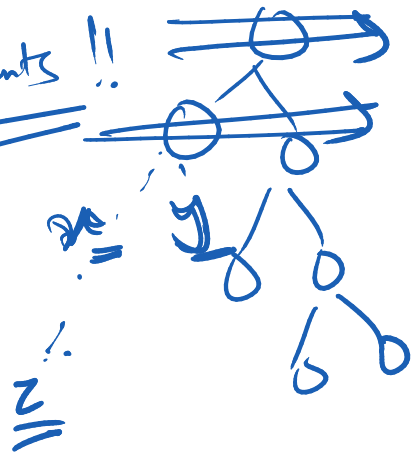
① level order find 1st empty leaf position. put the integer there

Making a tree.



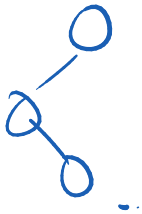
Complete Tree!!

6 elements !!



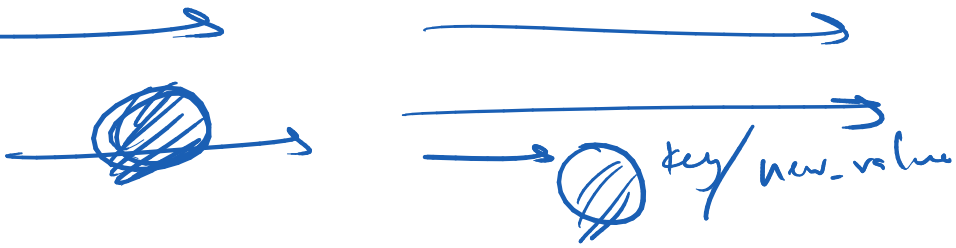
7th element.

② Read → pre/in/post
└ level.



③ Update : update (key, new_value)

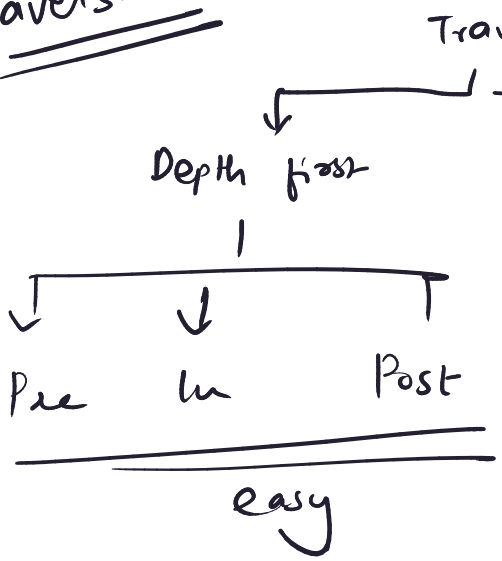
④ Delete:



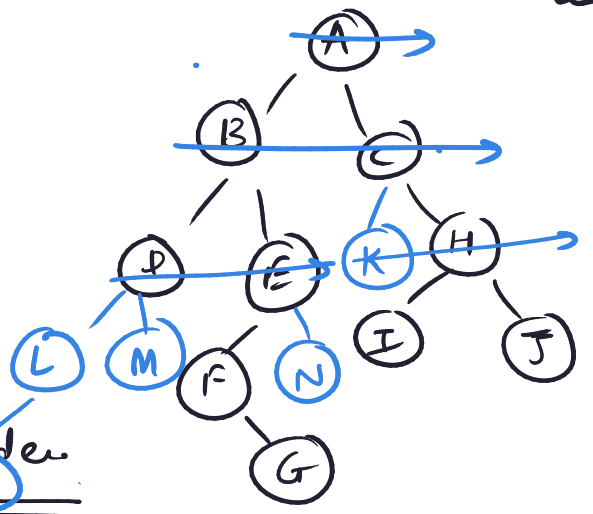
2 patterns : Level order traversal

??

Traversals:



Insert 'K', L



Level:

A B C D E H

F I J G

```
level_order ( root ) :
    if root == null return ;

    queue<Node*> q ;
    q.push (root)

    while ( !q.empty()) :          current = q.front()
        // Get the 1st node & process as
           required
        if (current → left)        q.push (current→left)
        if (current → right)       q.push (current→right)
```

Variations:

① Insert a node

Variation 1:

CRUD

input → Data structure → access all input

CRUD

Type specific operations

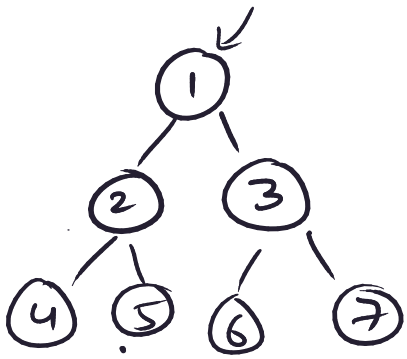
Creation :

1 node
2 node
3 node

} Perform level order traversal and find the 1st null & replace that with this node

Node* root = null

root.



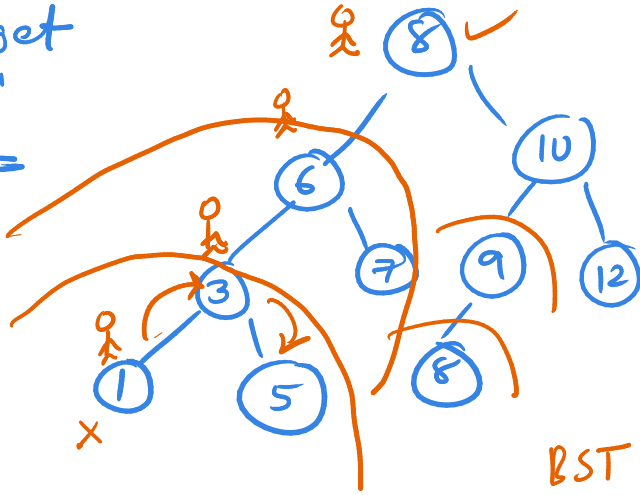
of min height()

② Read level order → array / print.

Q. Goldman Sachs.

I will give you a 'target' sum

If pair of nodes exist that form this sum.



Io x + y = target given x, y ∈ nodes of this tree.

BST

Inorder traversal left root right.

lets 1 3 5 6 7 8 8 9 10 12

left ≤ root ≤ right ⟹ all nodes

for n target - n ⟹ O(n)

multiset ⟹ O(n log n) tree

Why not array? LOT → [⟶ sort 2 pointer

③ Update.

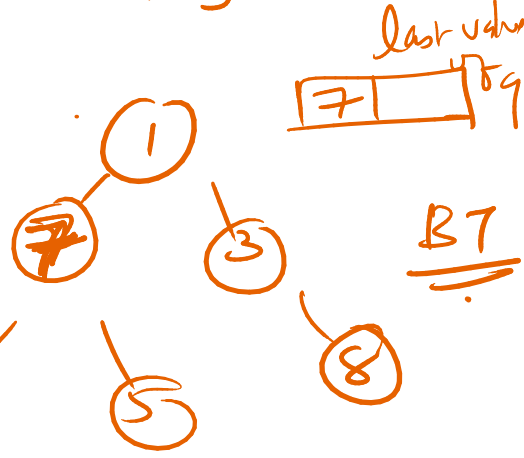
search(root, key) ⇒ Node*

Level ✗ Recursive

r → data = key2

④ Delete.

Level Order



delete (2) ⓐ check O(n)

2 should exist

Ⓑ O(n) find the last element of delete

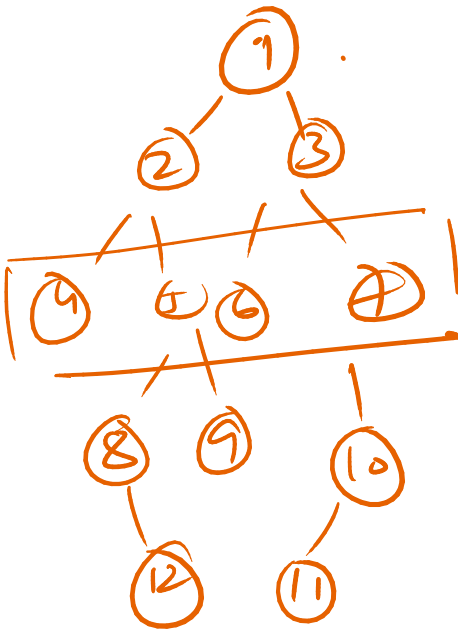
pre post in ⇐ lowermost node

Variation 2

width of a tree

track of q.size()

1 2 4 3



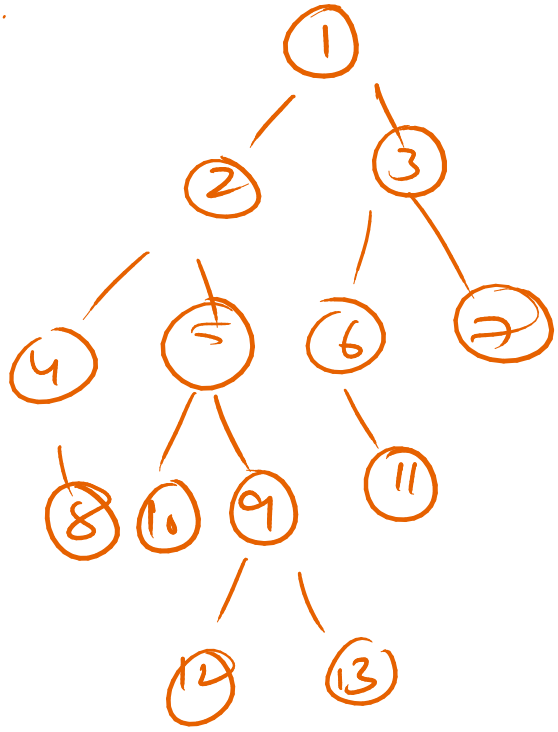
width = 4

max size you observed

↳ max width.

Variation 3

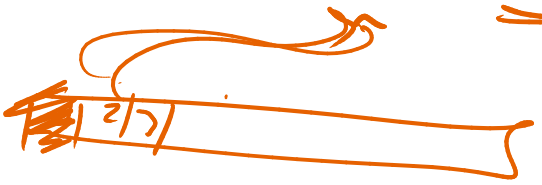
print corner nodes.



1 2 3 4 7 8 11

12 13

Vector<int> level; q.size()

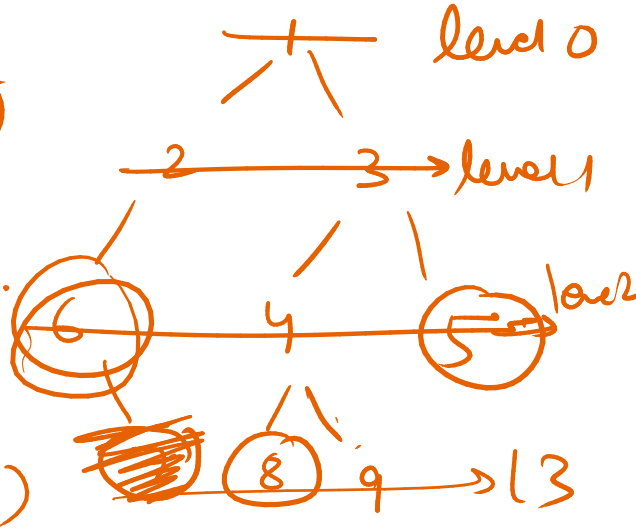


level[0]

level[level.size() - 1]

Variation 4

Q. find sum of leaf nodes at minimum level.



ans = 5

q.size()

when level switch is happening.

ans = [6, 5] ⇒ 11 Samsung

Variations

reverse level order Traversal

Variation 6

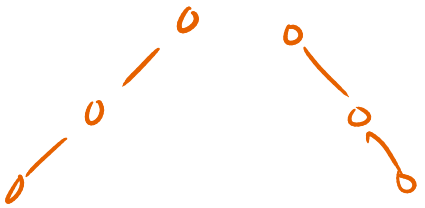
Spiral order traversal

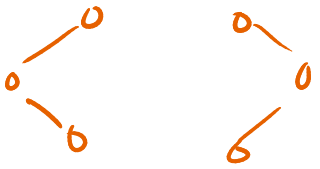
2nd template

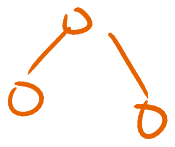
One of my fav templates.

Samsung

Question $n = 3$. Count the no. of structurally different BTs with n nodes.





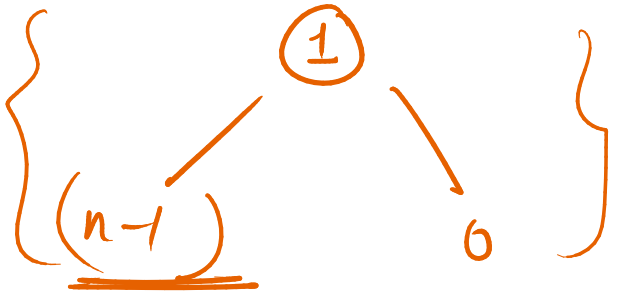


Output: 5

H1 : When given n, 1 node will always be at root. You have to actually think for rest of n-1 nodes.

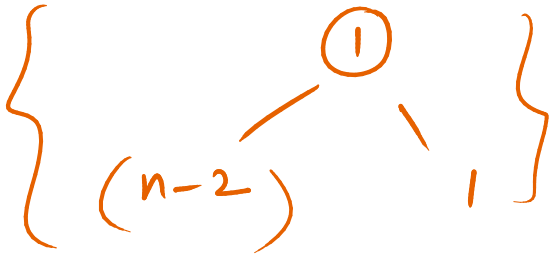
H2 : Of n-1, some of them go in left subtree, some of them go in right subtree

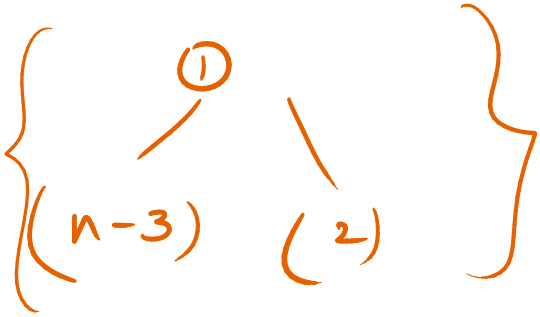
C_n ⇒ Count of BTs with n nodes.

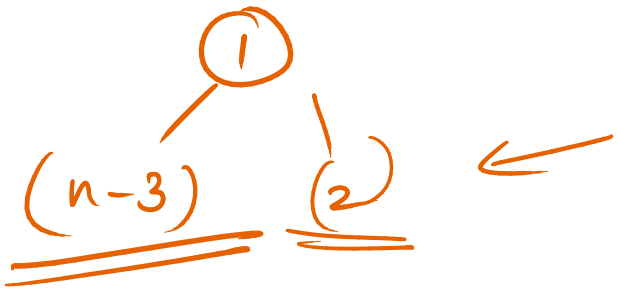


n-1 nodes

Sum

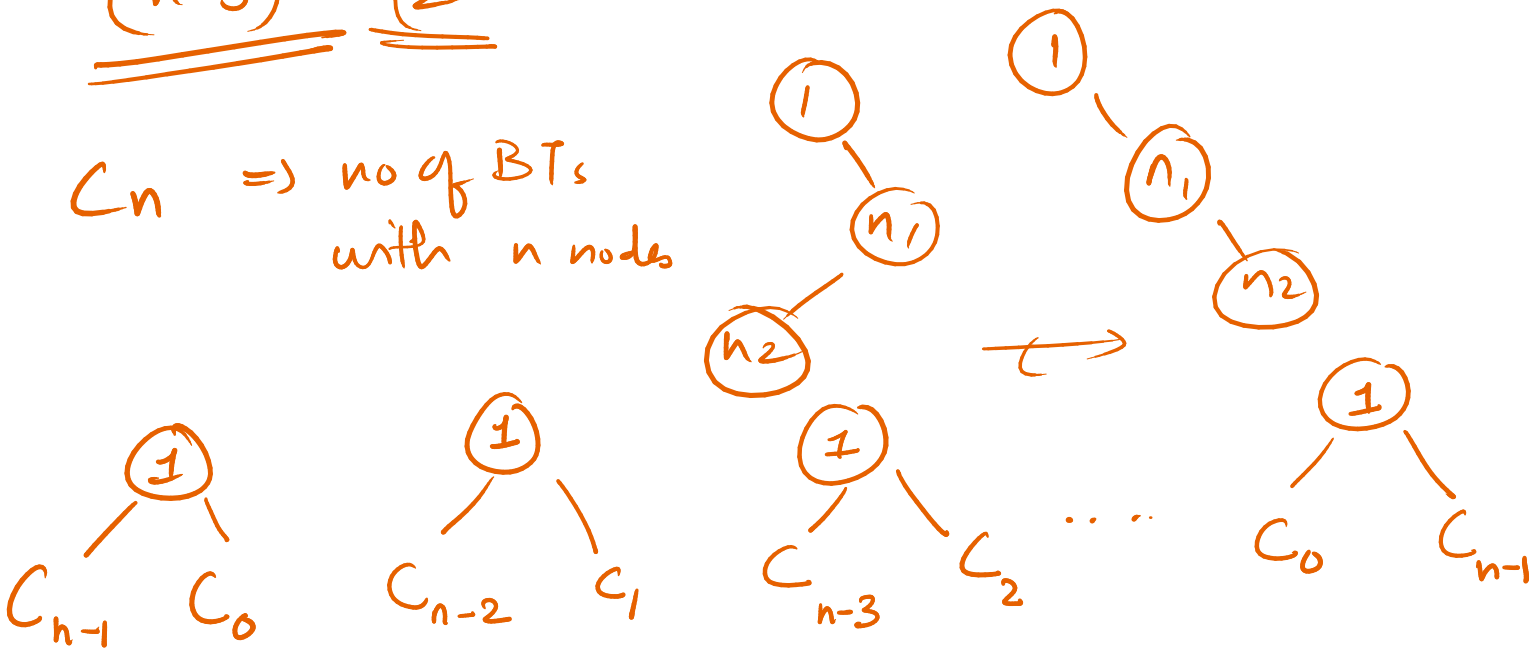






How are these 2 arranged

C_n => no of BTs with n nodes



Q. How many dresses you have ?

3 tshirts x 4 jeans = 12

$$C_n = C_{n-1} \cdot C_0 + C_{n-2} \cdot C_1 + C_{n-3} \cdot C_2 + \ldots + C_1 \cdot C_{n-2} + C_0 C_{n-1}$$

$$f(n) = f(n-1) + f(n-2)$$

$$C_n = \sum_{i=0}^{n-1} C_{n-1-i} \cdot C_i$$

Catalan numbers

func. C (int n)

if (n==0 || n==1) return 1;

C[n+1]

int ans = 0

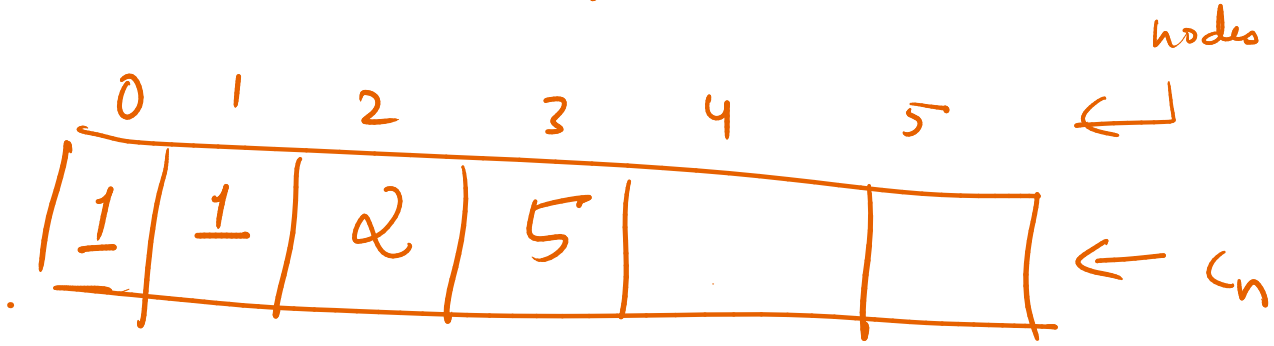
memo table

for (int i=0; i<n; i++)

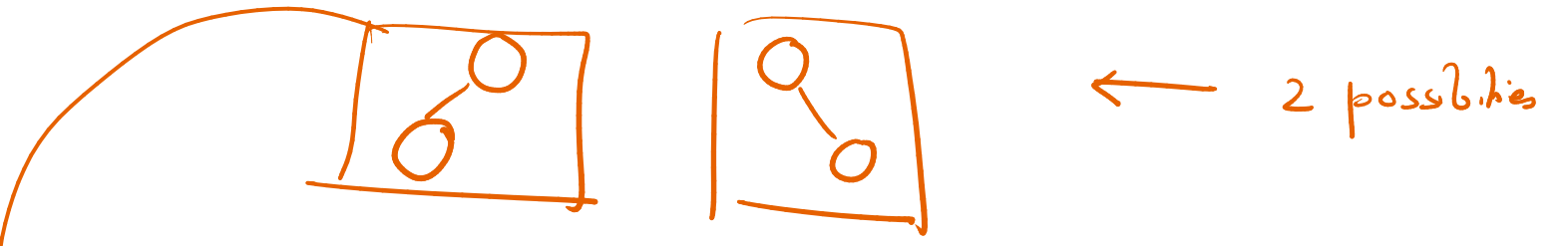
DP

ans += C(n-1-i) · C(i)

return ans;

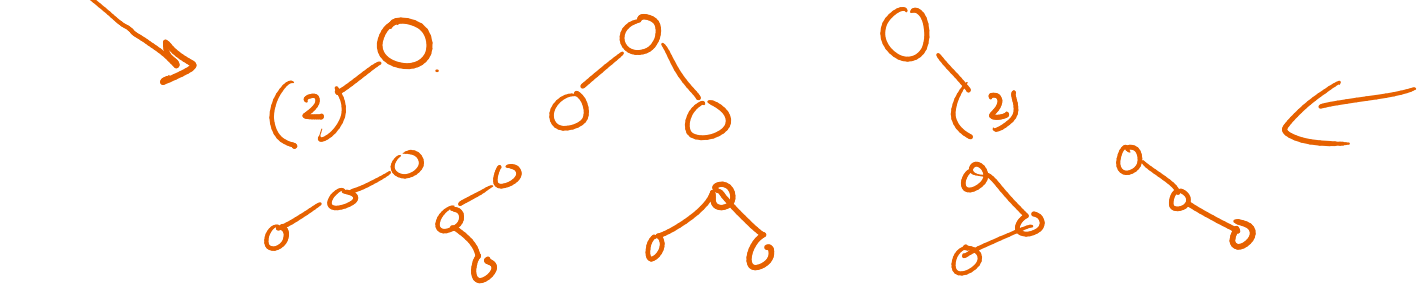


$C_2 = C_1 \cdot C_0 + C_0 \cdot C_1 = 1 \cdot 1 + 1 \cdot 1 = 2$



$C_3 = C_2 \cdot C_0 + C_1 \cdot C_1 + C_0 C_2 = 5$

$= 2 \cdot 1 + 1 \cdot 1 + 1 \cdot 2$



$$C_4 = \underset{\text{left}}{\overset{3 \text{ nodes}}{C_3}} \cdot \underset{\text{right}}{C_0} + C_2 C_1 + C_1 C_2 + C_0 C_3$$

$$= 5.1 + 2.1 + 1.2 + 1.5$$

$$= 14.$$

$$C_5 = 42$$

Variations

Q.1 Given n pairs of parentheses, find the count of balanced parentheses arrangements.

$n = 3$ — 3 open 3 closing

((())) (()()) (())() } output 5

()(()) ()()()

$n-1$

n pairs — $\_(\_)$ — left right

$n-1$ ( ) $n-2$ (1) $n-3$ (2)

.... $-(n-1) \Rightarrow C_n$

$C_3 = \underline{C_2 C_0} + \underline{C_1 C_1} + \underline{C_0 C_2}$

$2 \qquad 1 \qquad 2$

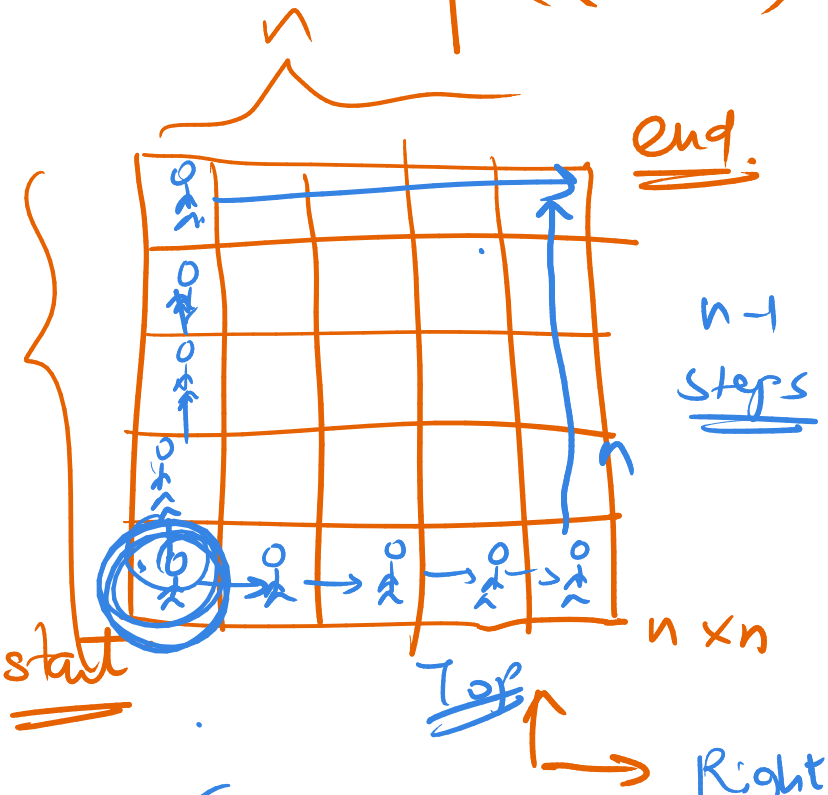
$-(-)$

5

$()()|() \quad ()|(()) \quad |(()())$

$(())|() \qquad\qquad |((()))$

Variation Q2.



no of ways this grid could be traversed such that you reach end.

C_n

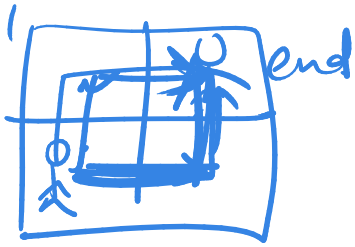
$\rightarrow \underline{C_{n-1} \cdot C_0} + \underline{C_{n-2} \cdot C_1} + \cdots + \underline{C_0 C_{n-1}}$

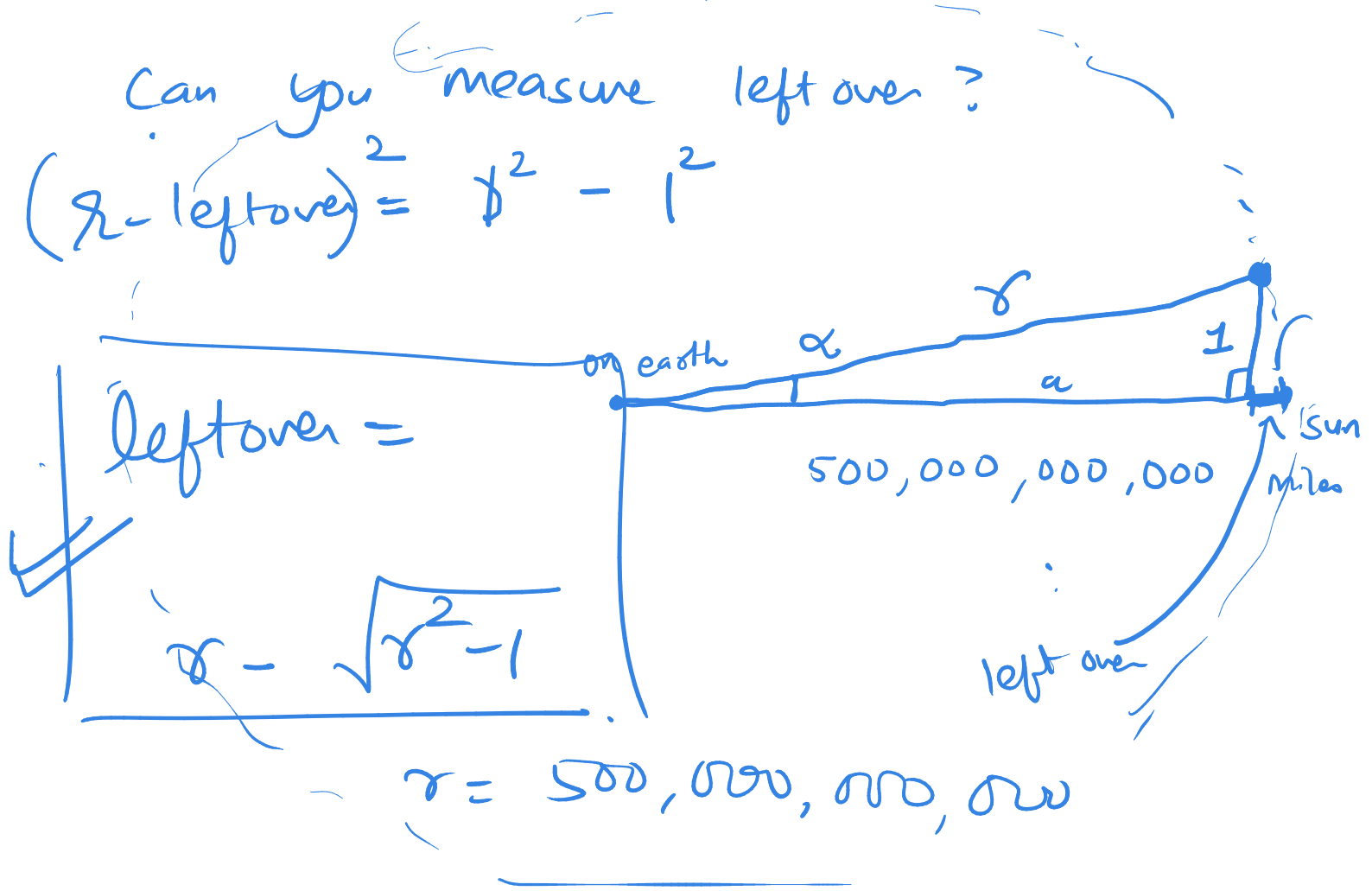
$1 \qquad\qquad 1$

$C_n \Rightarrow n \times n$

$C_n = \sum C_i \, C_{n-i-1}$ (2)

$C_2 = \underline{C_1 C_0} + \underline{C_0 C_1} = 2$





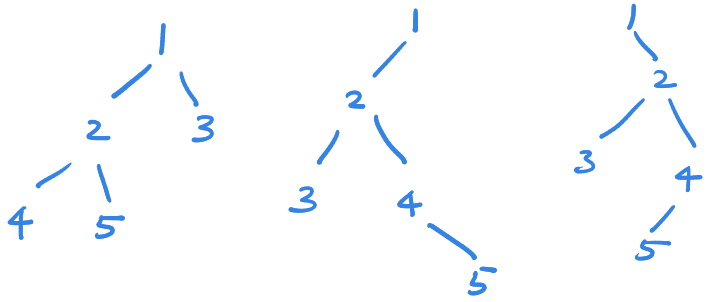
1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786,

① Binary tree → Traversals. → sequence of nodes.

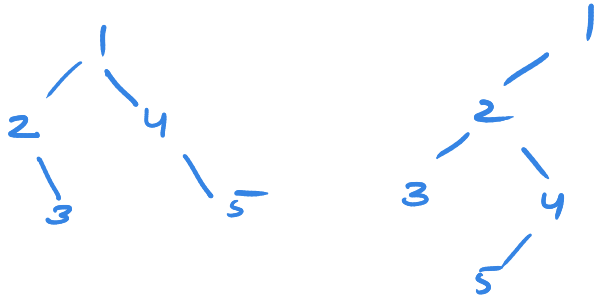
Q. Sequence → Binary Tree?

1 traversal ⟹ No. unique tree × [1 2 3 4 5]

Level

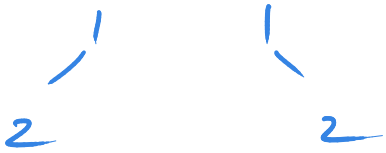


preorder



root left right

2 traversals → unique BT.



left right root

preorder: [1 2] [1 2]

postorder: [2 1] [2 1]

levelorder: [1, 2] [1, 2]

preorder
postorder → No !!

preorder :→ No !!

levelorder :

2 traversals ⟹ Yes !! unique tree given that one of them is inorder traversal.

root l r

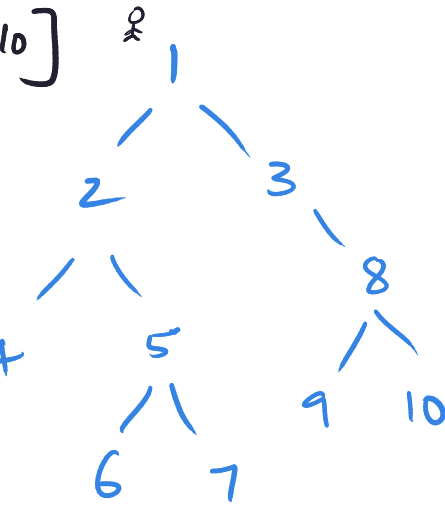
pre: [1 2 4 5 6 7 3 8 9 10]

l root r

in: [4 2 6 5 7 1 3 9 8 10]

l r root

post: [4 6 7 5 2 9 10 8 3 1]



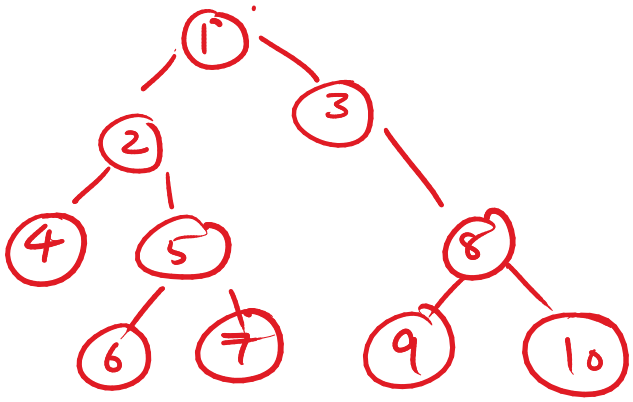
Think recursively

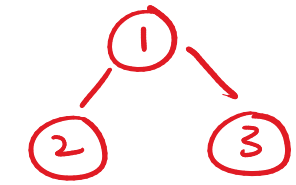
in 4 2 6 5 7 1 3 9 8 10

pre 1 2 4 5 6 7 3 8 9 10

in . . .

post 4 6 7 5 2 9 10 8 3 1





Recap

BT Theory

1. Trees → 2D DS | Types of trees & Terms
2. Recursive codes
3. Traversals → Depth (Pre In Post)
   → Breadth (level) → CRUD + some other codes where level switch was imp.
4. Structure of tree
   → Catalan number
   → Traversals → BT
5. Hierarchical structure } LCA, leaf to leaf diameter
6. All branches. }

Hierarchy: 2 & 12-13 ⇒ 2 is the ancestor

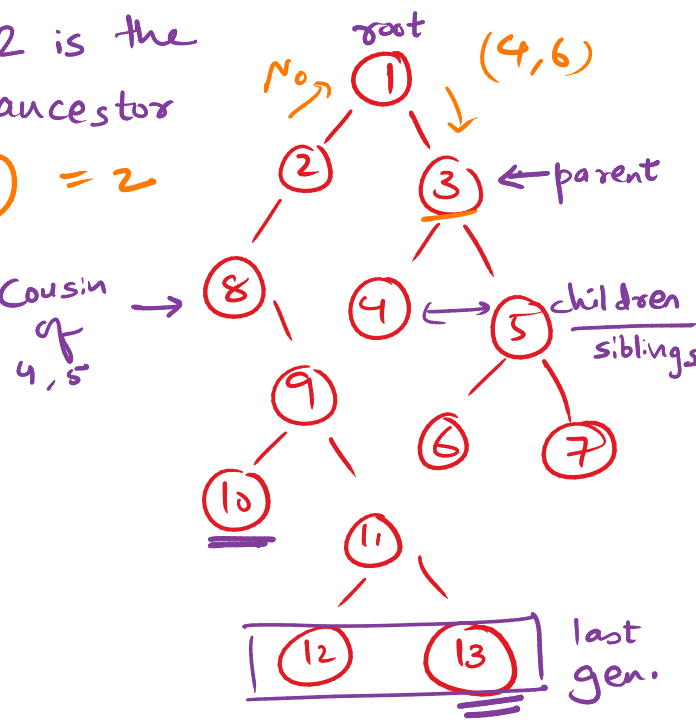
LCA(6,7) ⇒ 5

LCA(10,13) ⇒ 9

LCA(4,6) ⇒ 3

LCA(9,7) ⇒ 1

LCA(2,10) = 2



1st: LCA } How ??

2nd: list of ancestors (10,13) ⇒ [9,8,2,1]

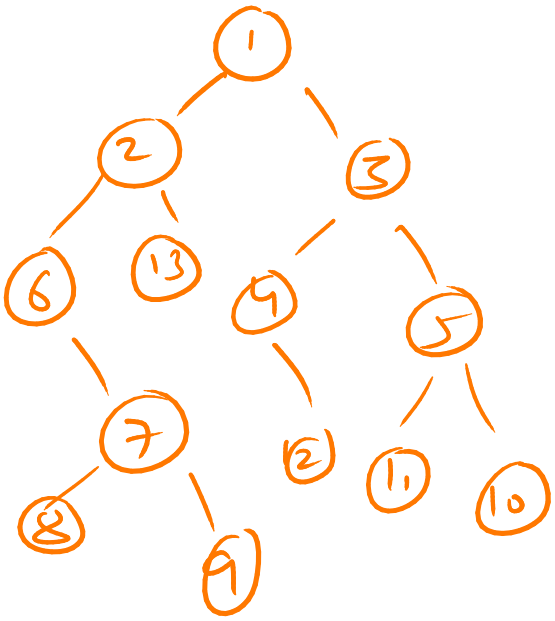
[1] [-,-,-1]

```
Node* find_lca(Node* root, int n1, int n2) {
    if (root == null) return null;
    if (root->data == n1 || root->data == n2)
        return root;
    Node* left_lca = find_lca(root->left, n1, n2);
    Node* right_lca = find_lca(root->right, n1, n2);
    if (left_lca && right_lca) return root;
    if (left_lca) return left_lca;
    return right_lca;
}
```

Templates. : Root node → particular node
I want all nodes on that branch.

(7) [1,2,6,7]

12 [1, 3, 4, 12]



root-to-node branch

root-to-leaf branch.

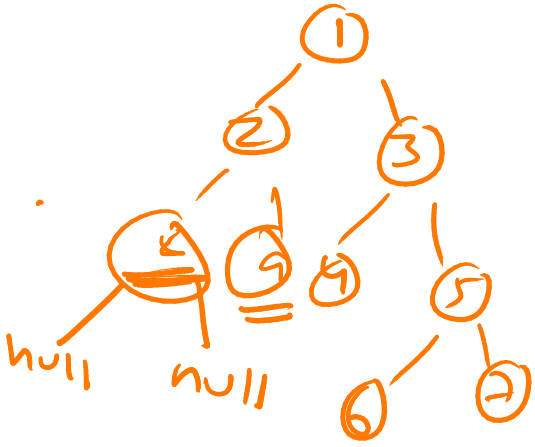
[1, 2, 6, 7, 8]

[1, 2, 6, 7, 9]

[1, 2, 13] [1, 3, 4, 12] [1, 3, 5, 11] [1, 3, 5, 10]

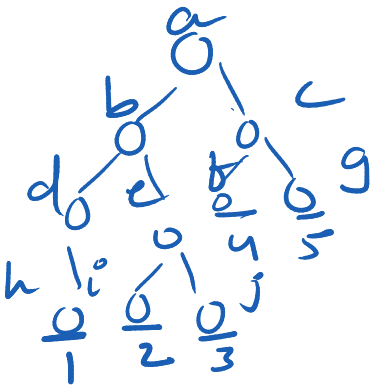
```
vector v;

bool root_to_node(Node* root, int key) {
    if (root == NULL) return false;
    if (root->data == key) return true;
    v.push_back(root->data);
    if (root_to_node(root->left, key) ||
        root_to_node(root->right, key))
        return true;
    v.pop_back();
    return false;
}
```

[1 2 ~~5~~ 9]



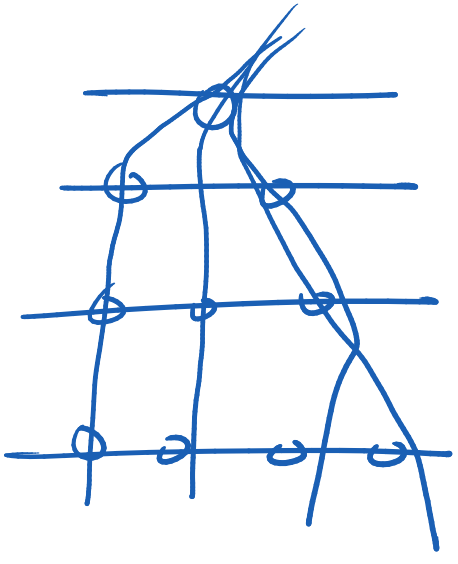
vector <vector <int> > paths ; ← 1D vector

root *




```
void root_to_leaves (root, path):
    if (root == null) return;
    path.push_back (root->data);
    if (root->left == null &&
        root->right == null) {
        paths.push_back (path)
    }
    root_to_leaves (root->left, path)*;
    root_to_leaves (root->right, path);
    path.pop_back ();
};
```

Imagine



Level order

root → leaf all paths

O(n)

Variations :

Q.1 find all ancestors.

1. find LCA
2. find root to LCA
3. print reverse

Interview

k^{th} ancestor of (a, b)

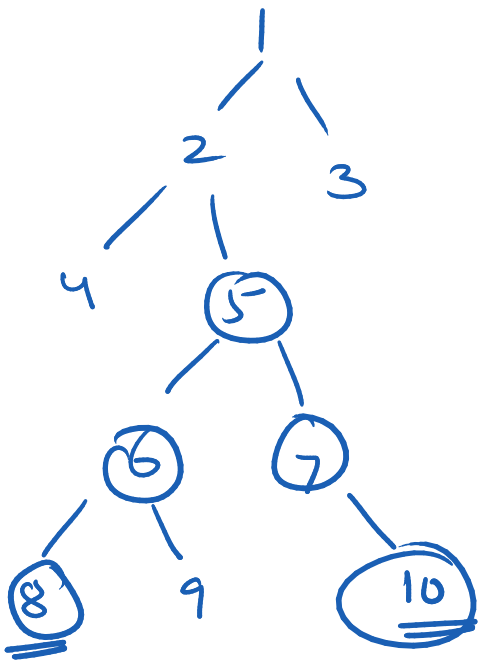
1st ancestor => LCA

9 5 4 1

Q.2 leaf to leaf path.

8 and 10

8 → 6 → 5 → 7 → 10



1. LCA [1 2 5 6 8]
2. root to leafs. [1 2 5 7 10]

Variation 3 Diameter of a tree

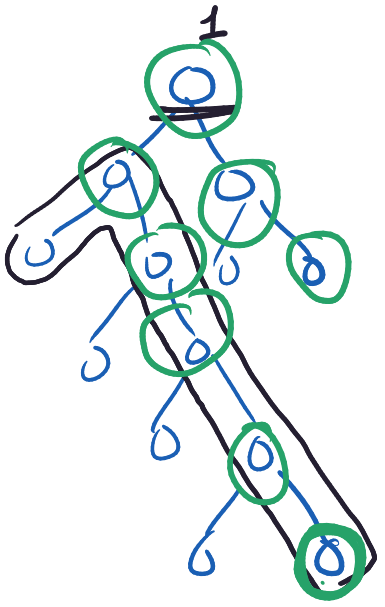
longest leaf to leaf path.

1. find all leaves 2. Make pairs
3. find leaf to leaf path 4. give me largest.

path ✗ length ✓

diameter = 0

diameter = Max { diameter, 1 + leftheight + right_height }



6 X

8

Recap: Trees → ① Structure of tree | types of trees ① ⇒ log n

② Recursive codes → CRUD BST ⇒ O(h) ②

③ Traversals → Depth (Pre, In, Post)
↳ Breadth (level) → CRUD BT + Templates

④ Catalan number

⑤ Construction of BT from 2 traversals
↓ must be inorder

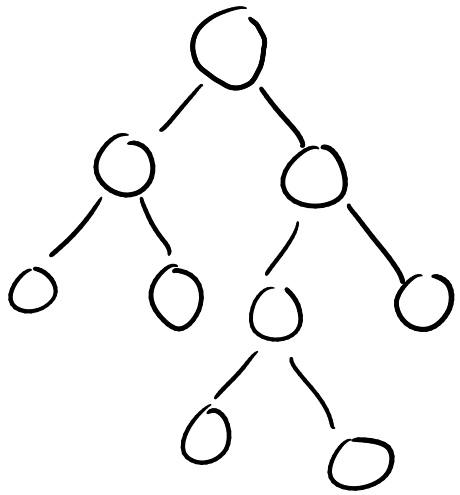
⑥ Hierarchy of tree → siblings, parents, cousins
↳ ancestors. → LCA ⇒ ③

⑦ Root to node paths. | All ancestors
↳ all leaves.

GENERIC BINARY TREES.

* Binary Search Tree (BST).

all nodes in left subtree	<	root node	<	all nodes in right subtree



Q.1 given a BT, tell if it is a BST or not.

✓ f < e < g

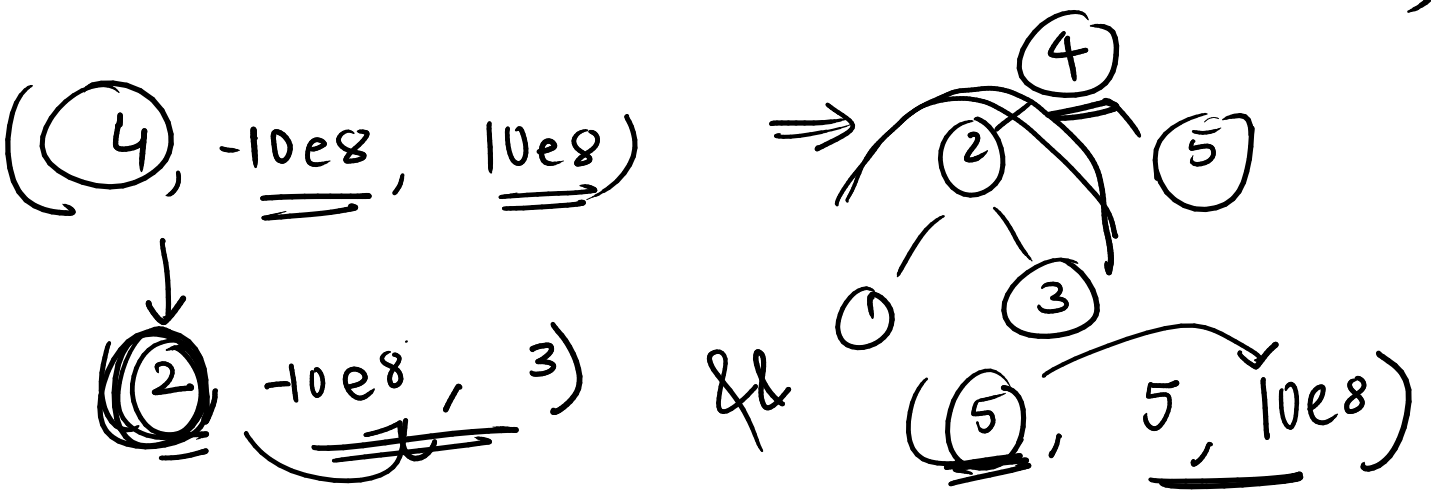
d < b < Min{f, e, g}
b ↑ f

Max{b, d, e, f, g} < a < Min{c, h, i, j}
g

Pseudocode :

```
bool isBST (Node* root):
    return isBSTUtil (root, INT_MIN, INT_MAX)

bool isBSTUtil (Node* root, int min_left, int max_right):
    if (root == NULL)  return True
    if (root.data < min_left || root.data > max_right)
        return False
    return isBSTUtil (root->left, min_left, node.data-1)
        && isBSTUtil (root->right, node.data+1, max_right);
```

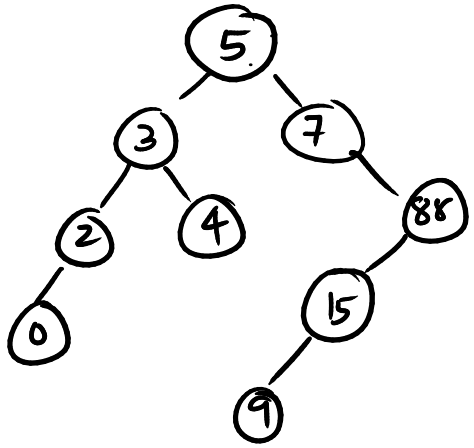
3 params
root
min_right
max_left



CRUD :

Creating : 5 3 7

C ⟵ { if next_num is smaller, go left.
if next_num is larger, go right. }



R inorder, levelorder } → sorted sequence interpretable

U key1 ⟶ key2

smaller — bigger
left root right

```
update (root, key1, key2) :
    if (root.data == key1)
        root.data = key2.
    else if (root.data < key1)
        update (root→right, key1, key2)
    else
        update (root→left, key1, key2)
```

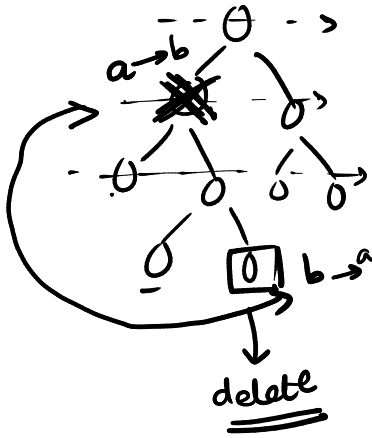
Search

O(height of tree)

D: Trickiest. → Recall delete a node in BT?

You can't delete in BT way because

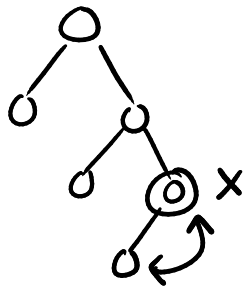
{ nodes on left } < root < { nodes on right. }



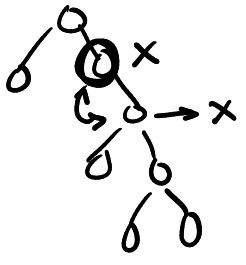
3 possibilities of delete : (BST).

(i) the node to be deleted is leaf node → DELETE

(ii) the node to be deleted has 1 child.
Swap the value with the child & delete the child node.



(iii) node has 2 children.
↳ swap it with its predecessor/successor & delete the pre/successor node.



BST → inorder → sorted sequence { a_1 a_2 a_3 a_4 a_5 a_6 }

a_3 : pre(a_4)

a_5 : succ(a_4)

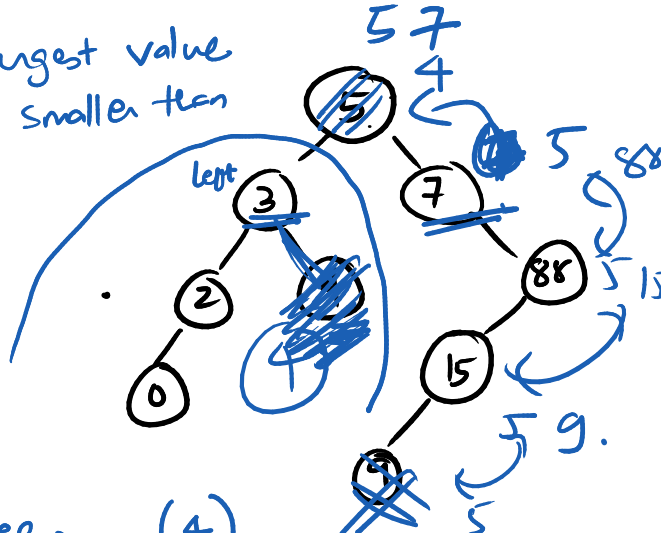
Why is it even working ?!! Recursive

Logic:

1. If root is null, return.
2. If key is found,
   - ✓ pre: rightmost child of the left subtree
     OR left child.
   - ✓ succ: leftmost child of the right subtree
     OR right child
3. search for the key:
   if key < root
   → left subtree
   else
   → right subtree.

Delete (5)

pre(5): largest value which is smaller than 5.



① Use our logic.

(A) swap (5, pre(5))

root == 5.

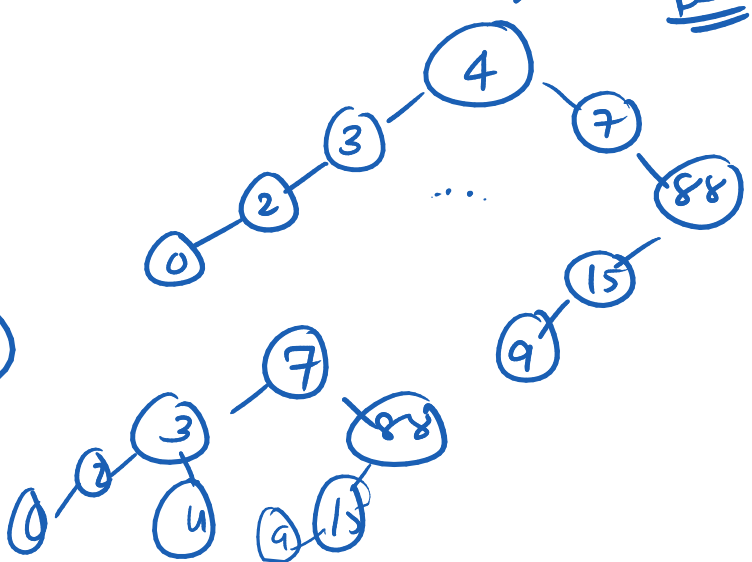
def: rightmost child of left subtree. (4)

(B) swap the values.

(C) Delete {5}.

② swap (5, succ(5))

BST



CRUD	BT	design "BST"
⓪ search (k)	O(n)	O(h(tree))
		always ↘
		O(logn) O(n)
① Create	O(n)	O(n)
② Read	O(n)	O(n)
③ Update	O(n)	O(h(tree))
④ Delete	O(n)	O(h(tree))

When does $O(h(tree)) \sim O(n)$?

↳ keep numbers in sequence.

10 9 7 5 4 2 1

CRUD

If $|h(l\_st) - h(r\_st)| > 1$

balance (tree).

$h(lt) - h(rt) = 0, 1$

if not balance()

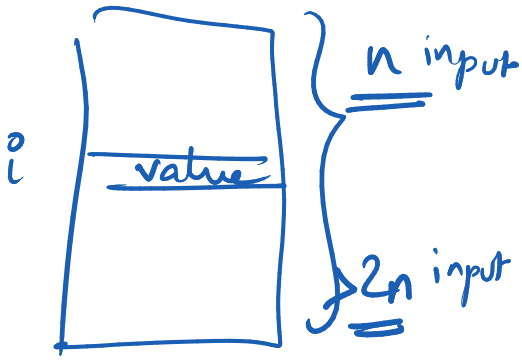
ROTATION. $\Rightarrow O(1)$

self-balancing BST

OR AVL Tree

Hashsets. n inputs.

hash value



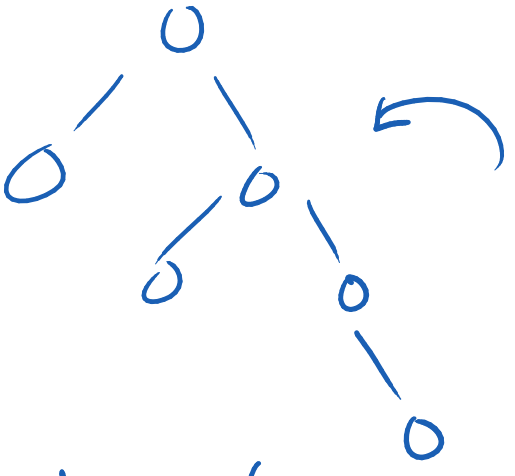
Self balancing BST

hash set

O(log n) insertion

O(log n) search

O(log n) delete.

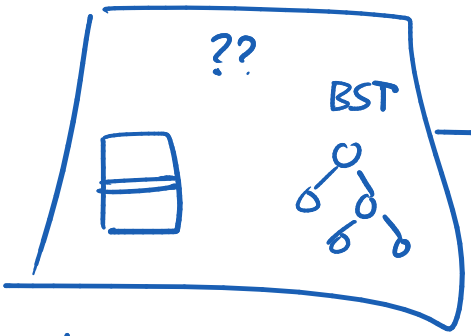


hash("saga")

hash("suraj") → 32

16

Hashset



input

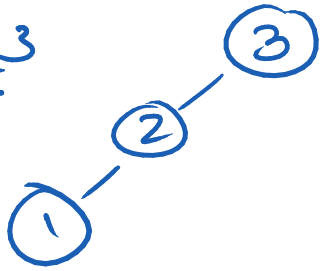
search(key)

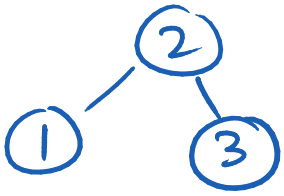
exists OR not.

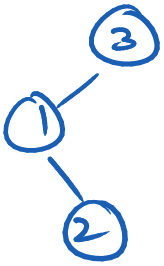
Black Box

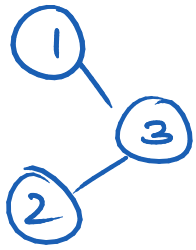
1, 2, 3

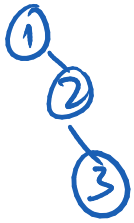
h = 3











3rd LCA of BT (n_1, n_2) ⇒ O(n)

if n_1 is in left ST
if n_2 is in right ST
LCA = Root.

LCA (n_1, n_2) of BST ⇒ node $n_1 < node < n_2$.

O(height) ⟶ Best $O(\log_2 n)$

1 tops more to go in BST.

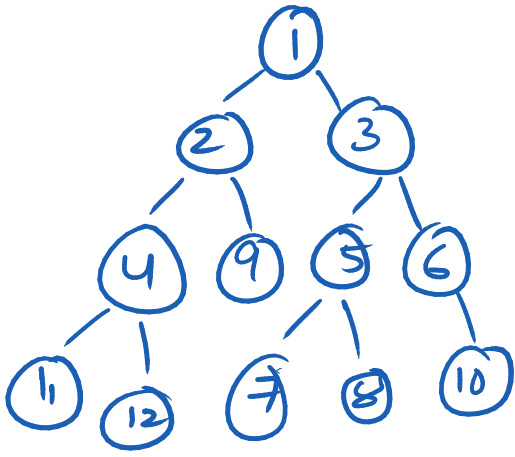
Constraint #1

{left nodes} < Root < {right nodes}

Constraint #2.1

HEAPS

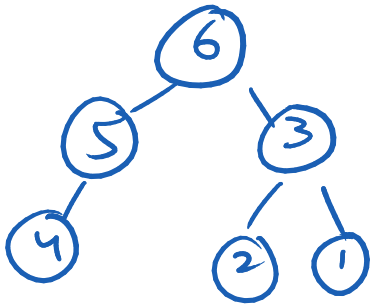
root
∧
{all nodes}



1, 2, 3, 4, 5, 6

2.2

root
∨
{all nodes}

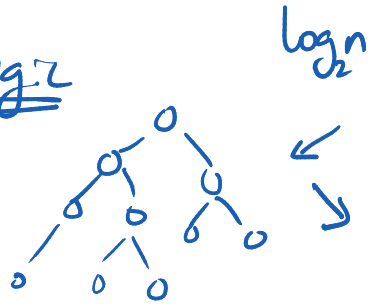


Median of n numbers.

1st : sort $O(n \log_2 n)$

middle $O(1)$

median $O(n \log_2 n)$



median $O(n)$

$\frac{1}{2}$ {smaller than it} < node <

Another logic $\Rightarrow$ median $O(n)$

Making a BST $\Rightarrow$ $O(n \log n)$

Insert : $O(h) * n$

Quickselect algo. k^{th} smallest element.

int partition (arr, left, right) {

return pivot_index.

}

→ O(n)

i compare k

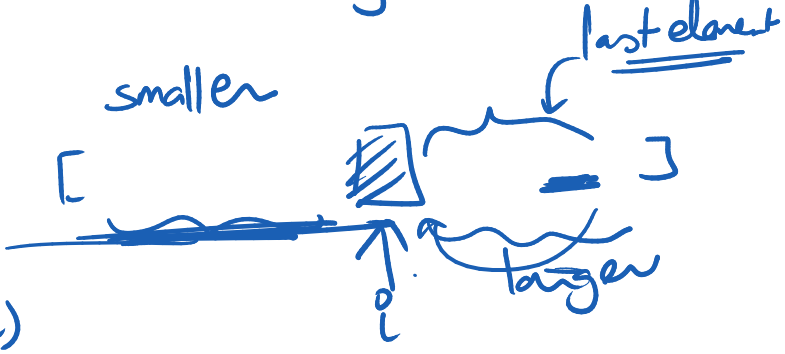
- i == k → return arr[i]
- i < k → search (i+1, right)
- i > k → search (left, i-1)

[1 3 5 7 9 11 13]

k=6

[3 7 9 1 11 13

5]



BST O(n) median

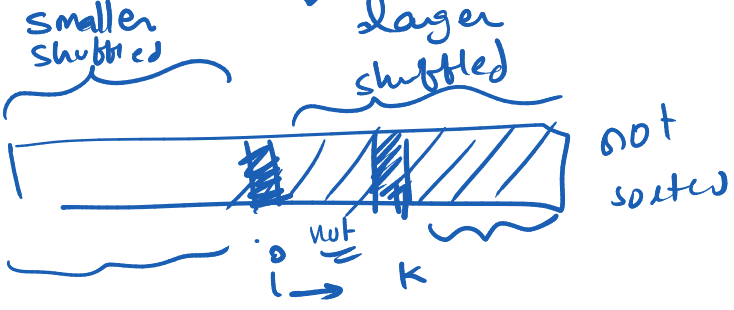
O(nlogn)

↓

~ O(n)

↑ approximate TC

O(n)



$k = n/2$

odd

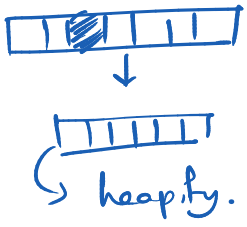
$k_1 = \frac{n-1}{2}$ $k_2 = \frac{n}{2}$

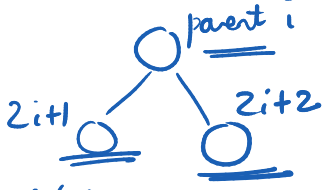
$\frac{k_1 + k_2}{2}$

Recap: last DS. Trees → Binary Trees
→ Binary search Trees | Constraint left < root < right
→ Heap | constraint root < > children nodes.

CRUD. Insert O(log n)

Reading O(n)
Update O(n)
Delete O(n)





Why heaps? ⇒ min (O(1))

Heapsort. ← deletemin O(log n). all numbers in sequence: n log n

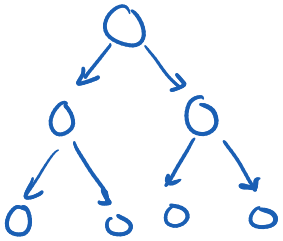
Heap ⇒ priority queue. DS | solved problem: Join ropes in minimum cost. classic

Graphs. So far we are having 1 parent 2 child.

Hierarchy. | Trees

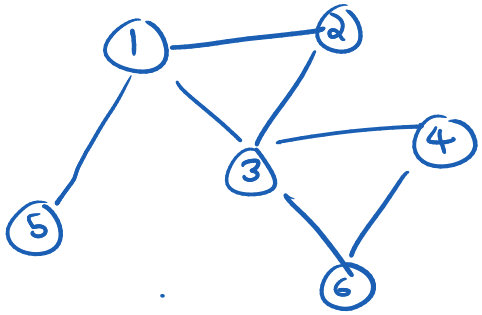
Not always all connections are hierarchical.

Generic model. nodes of data & no restrictions on connections.



Terminology

Graph = Vertices + Edges.
V E



V = 6

e = V-1 | min.

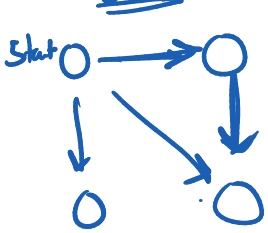
e = V(V-1)/2 | max

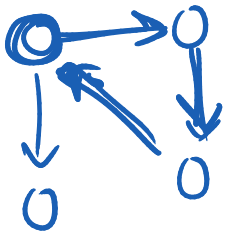
Types of graphs.

① Direction of edges :
- ✓ undirected graph → cyclic, acyclic
- ✓ directed graph → cyclic, acyclic

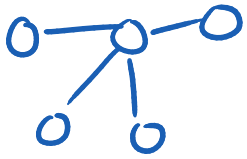
} not used much

cycle



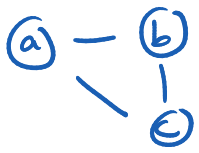


② Information :
- unweighted graph ✓
- weighted graph ✓



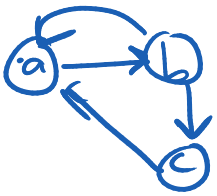
Why graphs? ⇒ most commonly used real-life data structure.
Everything is a graph!!

① undirected unweighted.



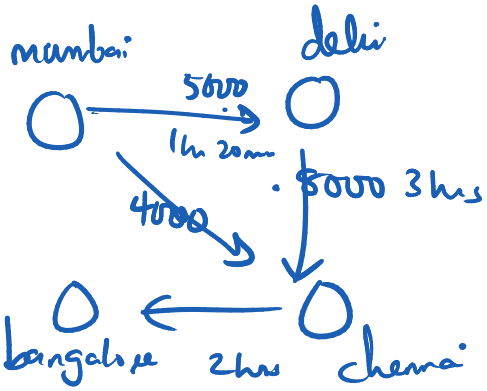
molecules. | abstract
lattice

② directed unweighted

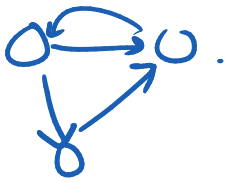


← Users on a social media
→ : follows

③ directed weighted → flights.

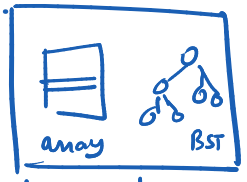


dynamic



How to represent them?

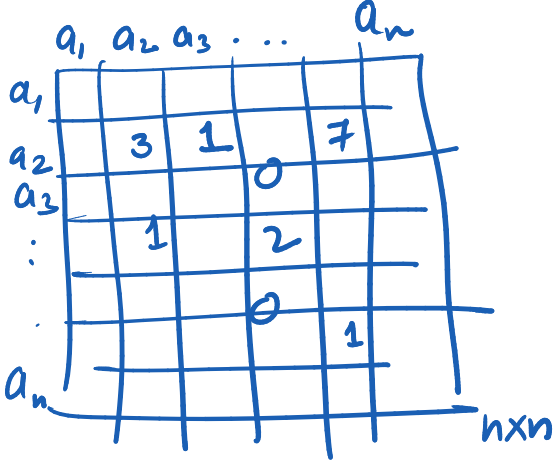
graphs.
- adjacency matrix
- adjacency list. ← more popular.



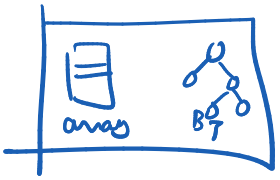
unweighted → edge exists or not

undirected weighted

① Adj matrix.



$a_{ij} = 1$
i^{th} node & j^{th} node are connected.



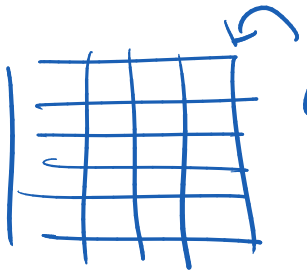
n vertices. edges?

$a_{ij} = a_{ji} = 1$ | if i & j are connected.

directed : a_{ij}, a_{ji} ⇒ separately. $a_{ij} = a_{ji} =$ int

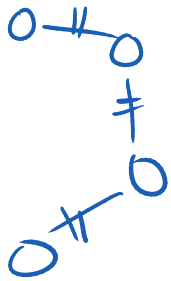
Matrix: $O(V^2)$ | traversal : touch all cells $O(V^2)$

(9) Adjacency List.



6 values

wasting space



undirected

3x2 values

16 values

sparse graph

$O(V^2)$

dense graph.

6 x 2 = 12 values

V vertices.

edges

N V

sparse

N $\frac{V(V-1)}{2}$

dense ‖ matrix is better.

V size

adjacent neighbours.

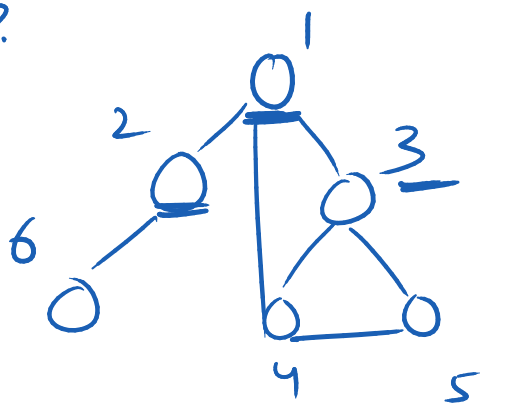
NNs

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

vector <vector <int>>

1 and 6 ?

1-2-6



CRUD.

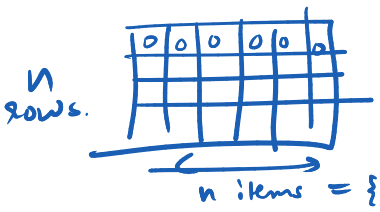
① Creation.

Give them serial number

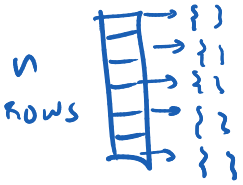
1. You need integers only : 1 to n
   0 to n-1

2. vector <vector <int >> graph.

   ↙ adj. matrix — n × n

   ↘ adj list.

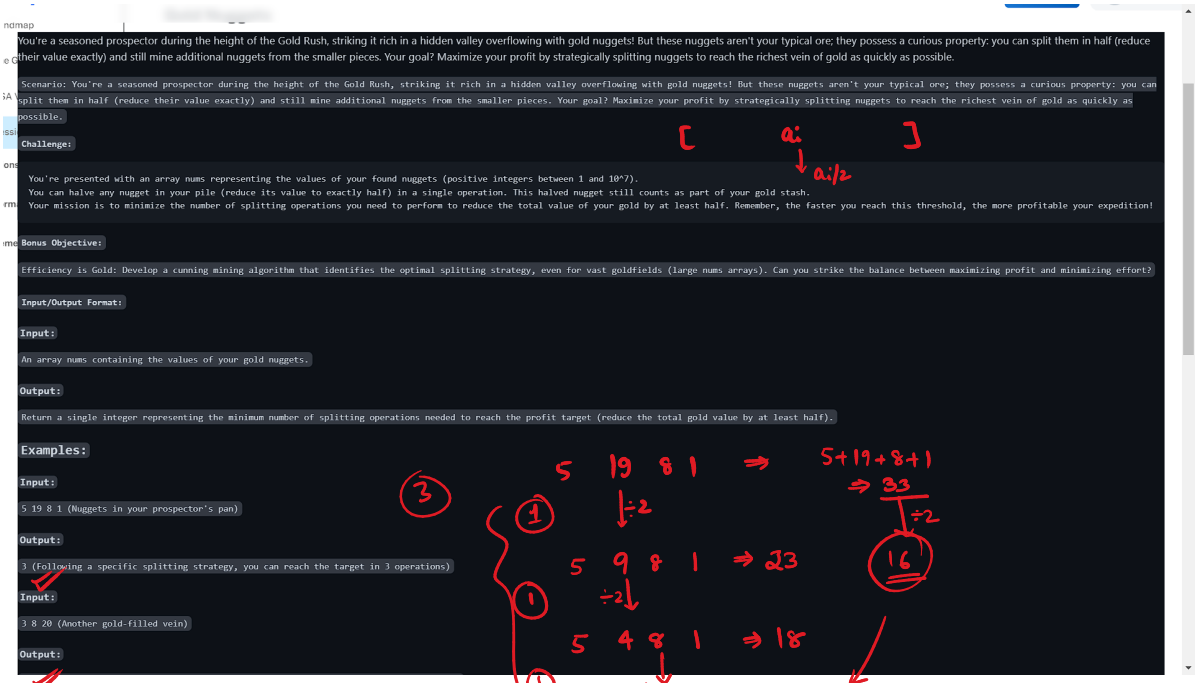
unweighted. ✗

3. add_edge (int i, int j)

undirected → graph[i][j] = 1
graph[j][i] = 1

graph[i].push_back (j)
graph [j] .push_back (i)

directed ✗ ↙ ✗ directed



You're a seasoned prospector during the height of the Gold Rush, striking it rich in a hidden valley overflowing with gold nuggets! But these nuggets aren't your typical ore; they possess a curious property: you can split them in half (reduce their value exactly) and still mine additional nuggets from the smaller pieces. Your goal? Maximize your profit by strategically splitting nuggets to reach the richest vein of gold as quickly as possible.

Scenario: You're a seasoned prospector during the height of the Gold Rush, striking it rich in a hidden valley overflowing with gold nuggets! But these nuggets aren't your typical ore; they possess a curious property: you can split them in half (reduce their value exactly) and still mine additional nuggets from the smaller pieces. Your goal? Maximize your profit by strategically splitting nuggets to reach the richest vein of gold as quickly as possible.

Challenge:

You're presented with an array nums representing the values of your found nuggets (positive integers between 1 and 10^7).
You can halve any nugget in your pile (reduce its value to exactly half) in a single operation. This halved nugget still counts as part of your gold stash.
Your mission is to minimize the number of splitting operations you need to perform to reduce the total value of your gold by at least half. Remember, the faster you reach this threshold, the more profitable your expedition!

Bonus Objective:

Efficiency is Gold: Develop a cunning mining algorithm that identifies the optimal splitting strategy, even for vast goldfields (large nums arrays). Can you strike the balance between maximizing profit and minimizing effort?

Input/Output Format:

Input:

An array nums containing the values of your gold nuggets.

Output:

Return a single integer representing the minimum number of splitting operations needed to reach the profit target (reduce the total gold value by at least half).

Examples:

Input:

5 19 8 1 (Nuggets in your prospector's pan)

Output:

3 (Following a specific splitting strategy, you can reach the target in 3 operations)

Input:

3 8 20 (Another gold-filled vein)

Output:

[ai]
↓ ai/2

3 → 5 19 8 1 ⇒ 5+19+8+1 ⇒ 33 ÷2 ⇒ 16
1 → 5 9 8 1 ⇒ 23
1 → 5 4 8 1 ⇒ 18
1 → 5 4 4 1 ⇒ 14

3 → 3 8 20 ⇒ 31 ÷2 ⇒ 15
3 → 3 8 10
3 → 3 8 5 ⇒ 16
3 → 3 4 5 ⇒ 12

nums → pq → max-heap.

priority_queue <int, vector<int>> pq

pq.push all numbers from nums.
int sum = sum of all nums.
int target = sum/2
int ops = 0

while (sum > target) {
int num = pq.top(); pq.pop();
sum = sum - num/2;
num = num/2;
} pq.push(num); ops += 1;

cout << ops << endl;

Reading : Traversal.

Recall Tree → Depth first (pre, in, post)
→ Breadth first (level order)

Graphs.

1:

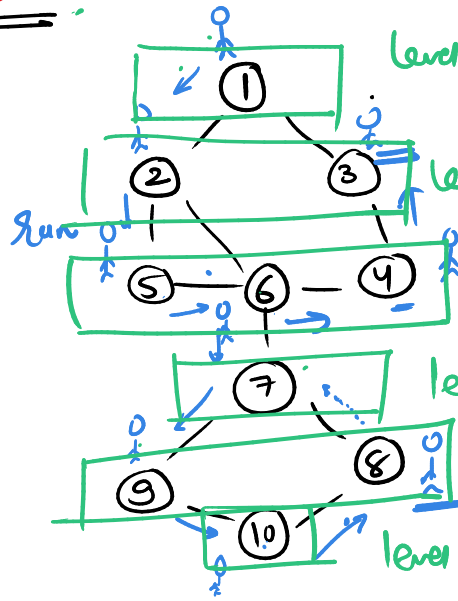
Depth first search (DFS)

Breadth first search (BFS) // level order

X → left root right

1 2 5 6 7

9 10 8 4 3



1 2 3 5 6 4 7

9 8 10

Run

1 → 2 → 6 → 7 → 9

1 2 3 4

logic: keep moving forward. If you hit a dead end, go to alternate path which is available to nearest node in path.

V.V. imp.

6 → 1 → 2 → 5 → 6

1 → 3 → 4 → 6

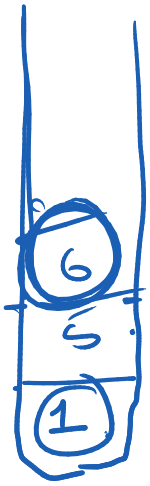
1 → 2 → 6

```
DFS (graph , s):
    visited [n] = {false} , stack<int> st

    // process the start node.

    St. push (s)
    visited [s] = true

    while ( ! st.empty ()) :
        int v = st.top() ;      st.pop () ;
        cout << v << " " ;
        for ( auto x : graph [v]) // goto neighbours
            if ( ! visited [x]) {
                st. push (x)      visited [x] = true
            }
```

graph → adjacency list

visited 1 2 3 4 5

T T T T F.



graph:

1 → 2, 3

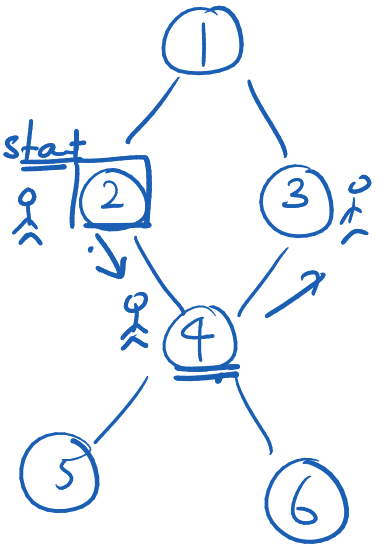
2 → 1, 4

3 → 1, 4

4 → 2, 3, 5, 6

5 → 4

6 → 4.



2 4 3

(2) → ~~1~~ ~~4~~ 3 → ~~1~~, ~~4~~

(4) → ~~2~~, ~~3~~, 5, 6

Graphs. Recap. Graphs = Nodes + Edges. | Connectivity

Represent
- Adj matrix (dense) — g[u][v] = 1 // g[v][u] = 1
- Adj list (sparse)

} → Types of graphs (edges) → direction, weights

CRUD.

① Creating a graph ⇒ vector<vector<int>> ← dynamic 2D Array

add_edge fn. to just create a graph.

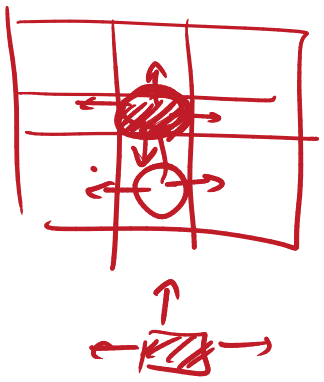
② Traversing / Read the graph ⇒ DFS and BFS. (start node)
↳ x

↳ disconnected

Coming up with a graph is tricky !!

stacks queues.

↳ minimum steps by a knight.
↳ imagining a grid as a graph.



Today's agenda : Write as much code as possible.

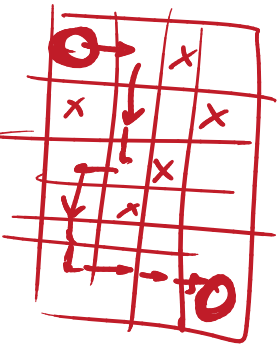
1) checking if there a path from src to dest.

2) If there exist multiple paths from A to B, I want to print them all | No.

3) check if cycles exist.

Certain problems : a) Solve a Maze. ↓ BFS

Longest Island Problem



0 - 1

0 - 3

1 - 3

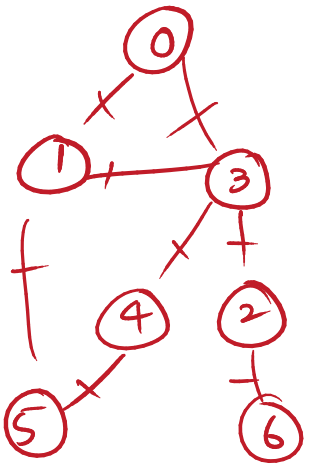
1 - 5

2 - 5

2 - 6

3 - 4

4 - 5



Recap. Graphs → Adj matrix, Adj list.

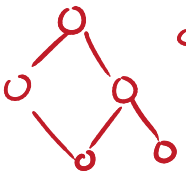
CRUD. → add_edge()

BFS, DFS. → matrix, list.

templates. → path.

Pending: 2) cycles.

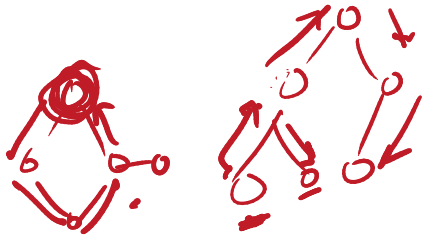
Classic problem: * Maze solving
* largest island.



cycle ⇒ 4 nodes.

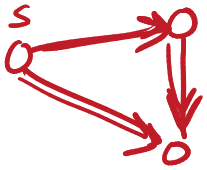
1) src → dest.

Start ① Cycles | Undirected easy. DFS



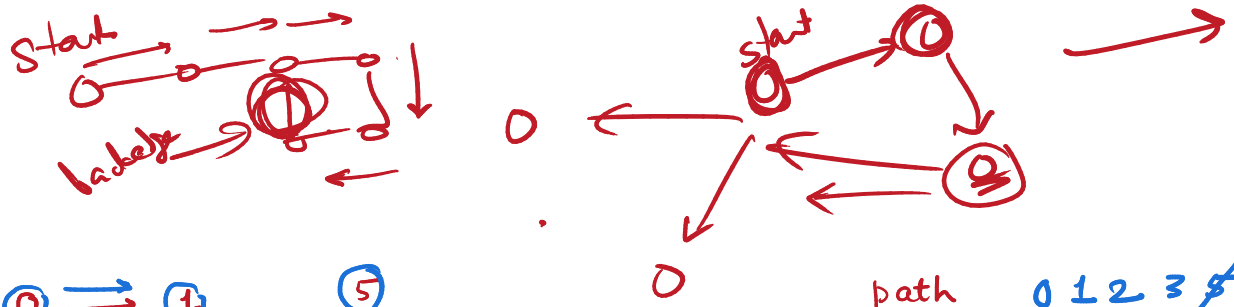
Directed graph.

It needs to have a back edge.

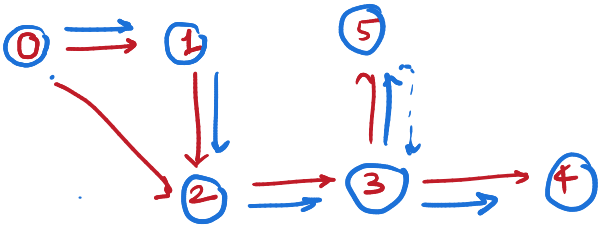


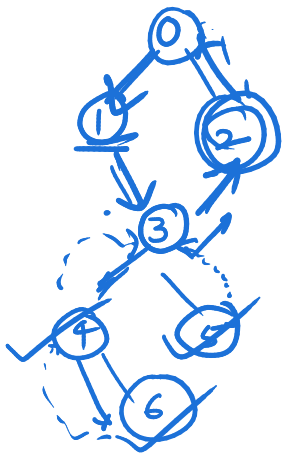
not a cycle set

nodes { , 



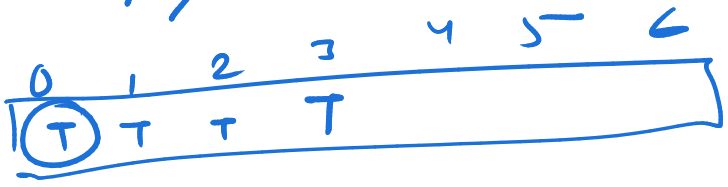
path 0 1 2 3 ~~4~~ 4





path: 1 3 ~~5~~ 4 ~~6~~ 2 0

recursion stack.



false

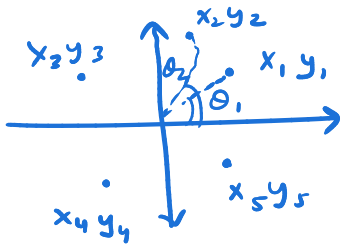
Problem Solving Domain

Chapter 1 : Introduction to PS

1. Understanding the templates | nxn grid → m is even/odd, n is even/odd → Student swap, Chocolate break, Chessboard cutting
2. Time complexity — 10^8 ops ~ 1 sec — given 'n', figure out mathematical class.
3. Recursion overview ⇒ factorial (linear), fibonacci (tree), GCD
4. Sorting algos. → $O(n^2)$ | bubble sort
   $O(n \log n)$ | Mergesort quicksort.
5. Imp. f^n of Qs → partition method $O(n)$ | Rearrangement: alternate +ve -ve, odd even, push zeros to the end.
6. Custom sort → comparator f^n. | numbers as strings → form largest no. Sort strings based on length or reverse of them for rhymes.

HW: (x_i, y_i) ⇒ n points | sort them based on the polar angle.

o/p: $[(x_1,y_1), (x_2,y_2) \dots (x_5,y_5)]$

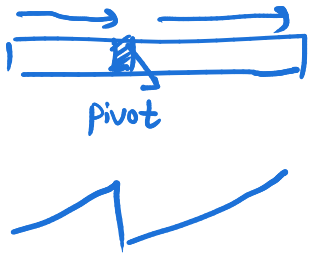


7. Searching
   - Linear search (unsorted) — $O(n)$ to search, $O(1)$ to insert.
   - binary search — 1 time effort $O(n \log n)$ to sort, $O(\log n)$ to search
   - BS as a method to identify best value x in range 1 to n for some b^n. f(x)

H.W Aggressive Cows SPOJ

8. Hashing
   - table → double the table | $O(1)$ amortized insertion time
   - collision methods ⇒ probing
   - Use sets & maps which are in-built.
     - sets: uniqueness
     - maps: frequency

9. Rotations → left/right | Reversal algo O(n)



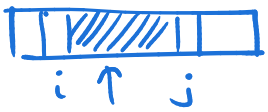
⇐ values graph in rotated array.

10. Two pointer method i, j ⇒ index

Case 1 : i → ← j Typical : Two sum

Case 2 : i → j → Typical : → Dejavu: partition function
→ Maximum sum subarray / Kadane's problem.

11. Sliding window technique i, j → window



This window is imp.

Window : set

Typical : longest non-repeating char substring in string

12. prefix sum arrays & difference arrays | $\int$ and $\frac{d}{dx}$ of arrays

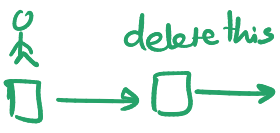
↓ cumulative freq. ↓ array of deltas

Typical : Bulb toggle | pattern: You will get alot of range queries to update your array.

H.W. Rainwater Trapping Problem Keep track of them in diff arr, Use prefix array to fix it eventually.

Chapter 2 Linked list (unpopular in interviews)

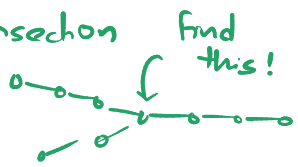
1. CRUD on data structure
   - Insert length of list, search, sum of ele
   - Traverse →
   - Update / search for a key & replace it
   - Delete



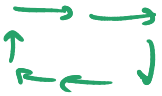
2. Specifics on LL: Reversing

Sorting → Mergesort

Intersection — find this!



3. loops : → → → → | slow & fast pointer.



4. Circular LL, Doubly LL. ← not discussed (ss)

Chapter 3: Stacks & Queues

1. Used STL | we did implement queues once.

2. Design questions in stacks & queues (ss) Imp!!

3. Basic classic f^n : prefix infix postfix of expression

Balanced pair (checking)

4. V.V.Imp. NGR NGL NSR NSL → Max area under histo.
   → Stock span problem

Chapter 4: Dynamic Programming

min, max, longest, ... ↑ #ways ↑

1. How to identify DP problem? ⇒ Optimization OR Combinatorial

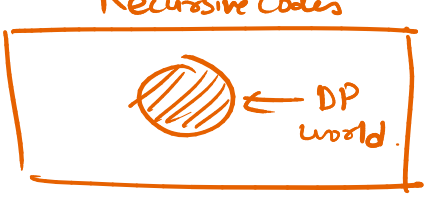
   a) Greedy → Counter case: Optimal strategy every step isn't optimal overall.

   Eg: change=6 coins={1, 3, 4}

   b) Try for recursion. |

   c) If not recursion, its a adhoc OR some other domain problem.

2. For DP to show its power, you need duplicate recursive calls. Cache them.



Recursion + Memoization → DP.

We need this first

To master this

3. Revisited Recursion. → Backtracking → Sudoku
→ Knights Tour
→ N Queens.

HW. Try to generate all possible subsets of an array.

i/p: [1,2,3] o/p. [[], [1], [2], ..[2,3], [1,]]

i/p: n
o/p: 2^n | Power set.

4. Templates of DP: many variations!!
a) Fibonacci
b) Knapsack
c) LCS
d) MCM

a) Min cost path → (2 dirs allowed)

5. Classic Standalone DP problems.
b) Word Break problem
c) Edit distance problem.
d) Gantt chart based scheduling.

HW: CSES problem set on DP.

Chapter 5: Trees

1. Types/Terminology: full, perfect, complete, skewed
2. Recursive codes: many variations
3. Traversals: Depth → Pre in post
Breadth → level order → variations!!
4. Construction → a) Given 2 traversals, can you make a tree?
b) Catalan number (variation of C_n)
5. Hierarchy → LCA, cousins, siblings, etc.
6. path ⟹ node to x, node to all leaves, diameter of tree.

Chapter 6: BST and Heaps.

1. BST → CRUD | predecessor & successor logic.
→ quickselect algo to find median in O(n) time.

2. Heap → heapify O(n) (array represented) | extract min again & again from min heap → Heapsort.

Imp: Heaps are priority queues. Classic: Join the ropes

Chapter 7: Graphs.

1. Graph terminology

2. C → add_edge | 2 representations: matrix list

   R → BFS → shortest path (advantage)

   DFS → disconnected graphs can be handled easily.

   U → search for a key

   D → If you delete a node, remove all edges dependant on that node.

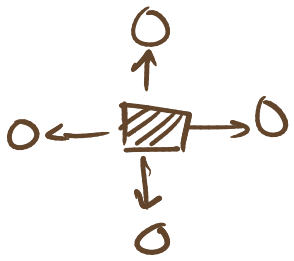
3. Many variations of BFS & DFS.

   a) Identifying graph is the key.

   b) print the path from src to dest. | all paths.

   c) Cycles

   d) V.V. Imp Grids as graphs !!



all 4 directions mean 4 connected cells are my neighbours.

HW.. SPOJ: Prime path (you have mastered BFS)

SPOJ: MM Knight moves (mastered BFS)

No. of components (DFS is done!!)

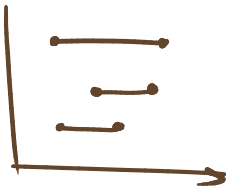
Max area island (DFS you killed it!)

→

NOT DONE. (not so popular in interviews)

(elaborate it tom)

Max jobs ⇒



① A huge section on graphs.

② Divide and Conquer :
- a) Binary exponentiation
- b) Fast Fourier Transforms

③ Greedy :
- a) Scheduling Problems.

④ DP :
- a) Digit DP
- b) Bit DP

⑤ String matching:
- a) KMP, Z, etc. | We have done Rabin-Karp

⑥ Range Queries
- a) Sqrt Decomposition $O(\sqrt{n})$
- b) Segment Trees
- c) Fenwick Trees / B trees (DBMS use B+ trees)

⑦ Number Theory
- a) GCD LCM
- b) Prime number
- c) Modular arithmetic
- d) Algebra & Geometry ⇒ Convex Hull problem.

⑧ Game Theory
- a) A performs a move
  B performs a move
  A ..
  B ...

  ———
  who wins ?

A majority of GT problems can be solved with pure logic.