Recap:

① Basic idea of problem solving : Template + Variation → Problem solved

② Time Complexity → Demo : Bulb toggle problem using code

$O(nq) \rightarrow O(n+q)$

③ 1st DS : Arrays → Recursion : linear recursion (1 call)

Eg. factorial

Every recursion :

a) Base cond^n

b) Problem decomposition

c) Problem Recomposition.

: Tree-based recursion (2 calls)

Eg. fibonacci

a) → Trivial

b) → Assume it gives us the ans.

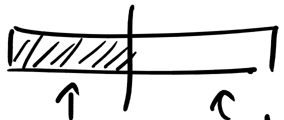
c) Just figure how you will combine the partial ans.

④ Sorting → $O(n^2)$: Bubble sort → no good

→ $O(n\log n)$: Merge sort

Quicksort → partition f^n

if cond^n :
 swap(,)



Rearrangement problems.

a) Alternate +ve -ve

b) Alternate even odd

c) Push zeros to the end

d) DNF problem arr[0,1,2...] sort it.

Per element.

⑤ Searching : → linear search $O(n)$ search $O(1)$ insertion

: Binary search $O(\log n)$ search $O(n)$ insert.

Use binary search to find optimal value in a range.

1. output you want → mid
2. You find the range for this mid.
3. When you will move left & when right.

unique-ness ← set & map. → freq

Hashing O(1) search O(1) Insert.

← Complicated → readymade available sol^n

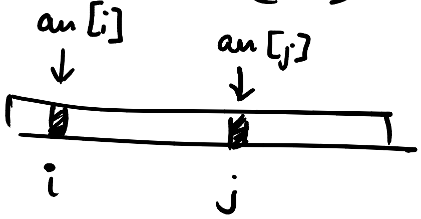
Bucketsort.

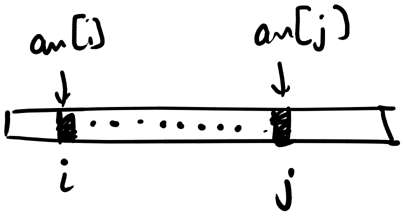
In future (trees) O(logn) search O(logn) Insert

Binary Search Tree (BST)

with no extra space.

(6) 2 pointers. :





i and j will change based on some rules

→ 2 pointer method.

i & j ⇒ imp.

all elements bet^n i & j

sliding window technique →

Simplest standard problems.

(1) 2 sum problem. Q. arr = $[a_1, a_2, a_3, a_4 \dots]$ Sum = 12

Is there a pair of elements that can give me this sum ?

Bruteforce : # no. of pairs possible with n elements $= {}^nC_2 = \frac{n(n-1)}{2}$

[1, 2, 3, 4]

(1,2) (2,3) (3,4)
(1,3) (2,4)
(1,4)

T.C ~ $O(n^2)$

```
for (int i = 0 ; i < n-1 ; i++)
    for (int j = i+1 ; j<n ; j++)
        print (arr[i], arr[j] )
```

Optimized : 1. Sort the array → sort( )

sum = 12

arr = [1 3 4 6 7 9 10 11 13]

i ~~i~~ ~~i~~ ij ~~j~~ ~~j~~ ~~j~~ ~~j~~ j

start   end   end

→   ←

pair_sum <= sum   pair_sum > sum

1 + 13 = 14 > 12

(1,11) 1+11 = 12

3+11 = 14 > 12

3+10 = 13 > 12

(3,9) 3+9 = 12.

4+9 = 13 > 12

4+7 = 11 < 12

6+7 = 13 ~~>~~ 12

```
        i=0
while (i != j)  j = len-1
    if  arr[i] + arr[j] == sum :
        → save this pair
          i++
    else if  arr[i] + arr[j] < sum :
          i++
    else :
          j--
```

Use the higher scope array.

O(n)

T.C. O(nlogn) + O(n)

~ O(nlogn).

Strings. (problems) → also valid array problems
→ string-specific problems (pattern matching) → x

str = "malayalam"

str[2] = 'l'

str = "abcbcbca"

find "bcbc" 2 times.

Sorting. strings ✓

1 particular string → sort characters.

strings → length / reverse of string

hashing : bucketsort. map the characters → freq.

unique names → sets

2 pointer : 2 sum

Q. How will you identify if a string is palindrome?

str= m a l a y a l a m

i ~~i~~ ~~i~~ ~~i~~ ij ~~ij~~ j

```
          i<j
while ( ~~i!=j~~ ) :
    if  str[i] == str[j] :
        i++ , j--
    else
        return false
return true
```

DP. a b b a

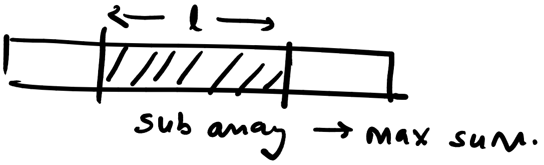
~~i~~ ~~ij~~ j

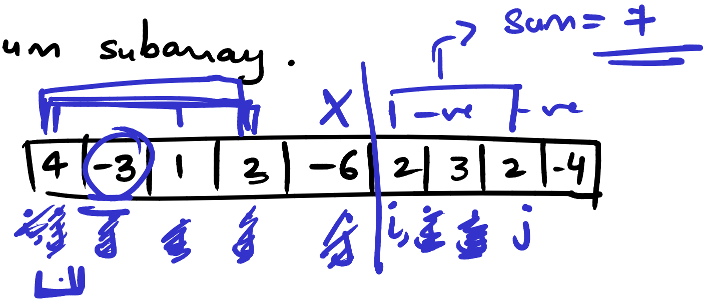
Moral of story : Whatever you learn for arrays → transfer to strings.

2 pointer method : i→ ←j , next: i→ j→

classic question.

Kadane's algo : Max sum subarray.





current_sum=0 → sum of all elements between i & j

max_sum=0 → largest sum seen so far.

ans=5

0,3

cum-sum = 0   cum_sum += arr[j]   max_sum = 0

4   j++   = 4

+ve

= 1   = 4

cum_sum = 2   = 4

= 2   = 5   = 5

= 5   = 7

-ve

= -1, = 0, j++   = $\boxed{\cancel{3}}$ max_sum = 7

i=j

```
ans = max_sum   (7)
    cum_sum = max_sum = i = j = 0, sub_i, sub_j
    while j<n :
        if curr_sum > 0 :
            cum_sum += arr[j]
            j++
            max_sum = max(max_sum, cum_sum)  →  if max_sum < curr_sum :
                                                    max_sum = curr_sum
                                                    subi = i
                                                    subj = j
        else:
            cum_sum = 0       |  Reset phase
            j++               |
            i=j               |
    return max_sum
```

Kadane's algo

O(n)

Sliding window problem.

i & j → valuable → but everything in betn. window.

Container ← window.

window → set, map, array

set → uniqueness

map ↑ freq

array ↑ in order

Q. String s = " pwwkew "

longest substring with no duplicates.

max length of substring with no duplicates → 3.

99%

set

map

FCFS — queue

LCFS — stack.

logic :

```
set <char> seen;   → Container
max_length = 0, i = j = 0

while (j < n):
    if !seen.find (s[j])  :        // not present
        j++
        seen.insert (s[j])
        max_length = max (max_length , j-i+1)

    else :                         // duplicate found
        { seen.remove (s[i])
          i++ }                    I am not updating the 'j'. ✗

return max_length
```

~~a~~ ~~b~~ ~~c~~ d c

~~0~~ ~~i~~ ~~i~~ i j

Hard Leetcode

Q. s = " adobecodebanc "

t = "abc"

~~b~~ ~~a~~ n ~~c~~

~~a~~ ~~b~~ ~~c~~

O/p : that substring

Q. Find the shortest substring in s such that all chars of t are available in that substring.

" " if no substring is found

t → duplicates : a = "b a b a"   t = "a a"   O/p : "a b a"