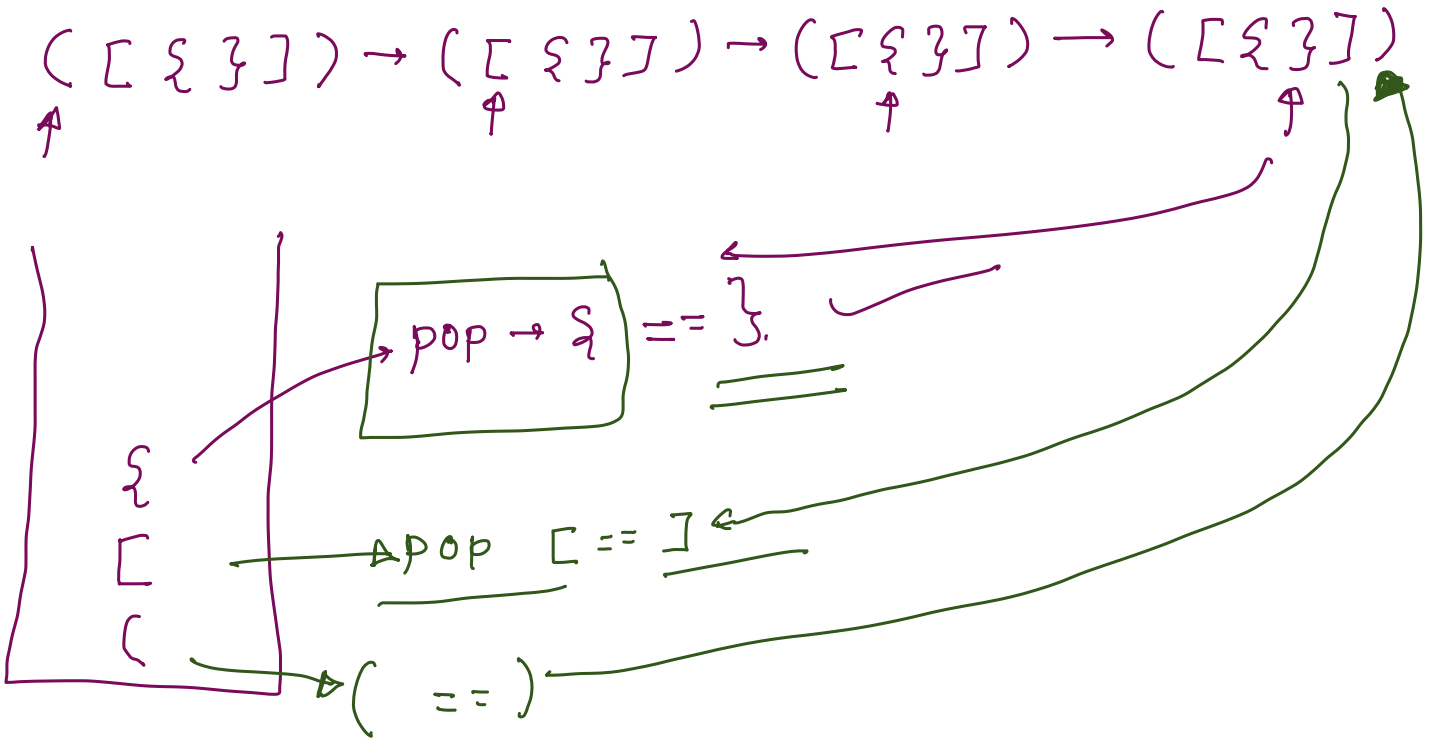
① Valid Parentheses -

we put opening brackets in stack

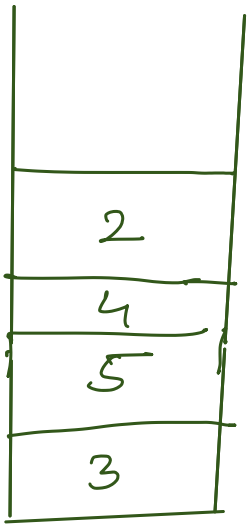
when we see closing brackets we try to pop.

if poped element != current closing bracket

return false.

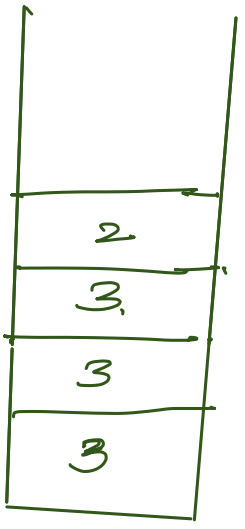


② Minimum Stack -



S1

Regular



S2

Minimum till now

① We have to keep the size of each stack equal.

② Regular stack keeps adding element normally

③ S2 stack.

compares the current incoming element with peek of S2

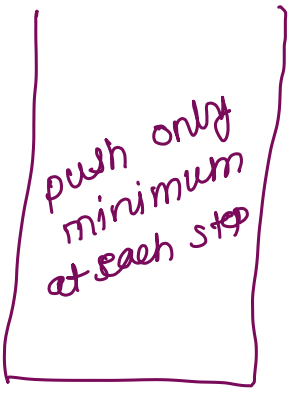
if it is smaller we put it in S2

otherwise we keep putting the previous smaller number

because previous smaller number would still be smaller so we add that number again in S2

This is how we push

S1



S2.

pop → pop both stack.

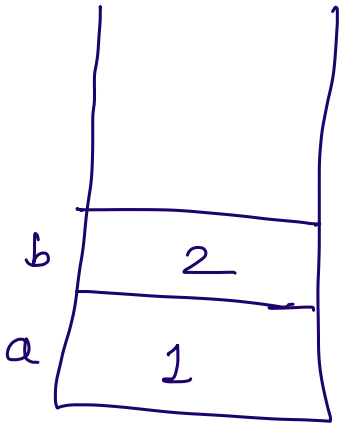
top → top of regular array S1.peek()

getMin → top of S2 (minimum array)

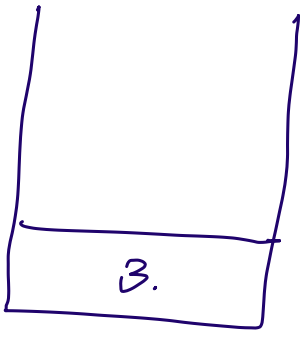
(3) Evaluate reverse polish Notation

Given → [1, 2, +, 3, *, 4, -]

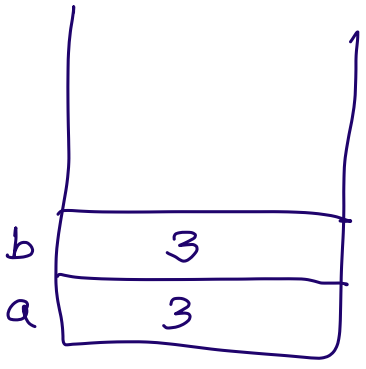
push push.



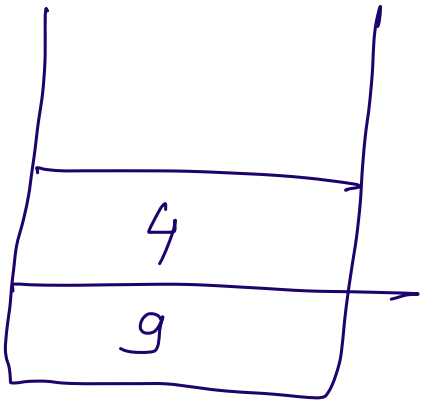
→ 1 + 2 = 3
a b.



1) keep pushing numbers
2) When you see a sign pop two numbers apply that sign
3) push the result
5) keep doing this & pop result at last.



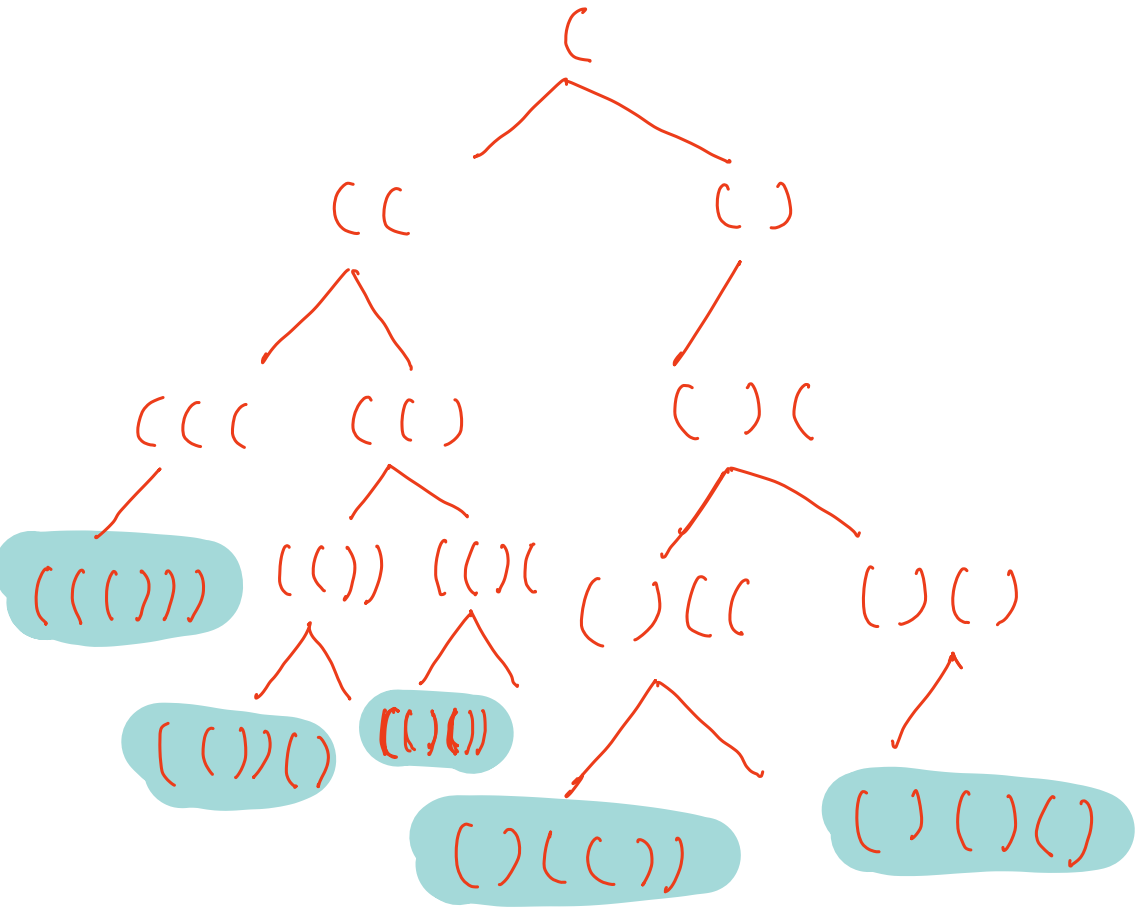
3 * 3.
a b.



9 - 4. = 5
a - b

⑨ Generate parenthesis

$n = 3$.



we can keep adding openparenthesis

if openCount < n.

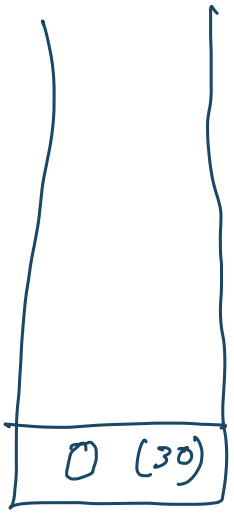
result, n, open+1, close, str + '('

if close < open.

result, n, open, close+1, str + ')'

if (str.length == 2n) result.add(str).

⑤ Daily temperature -



for $i = 0$
push (i)

for $i = 1$

index → 0

result[0] = 1 - 0 = 1.

when we pop any element
we will get to know about that element.

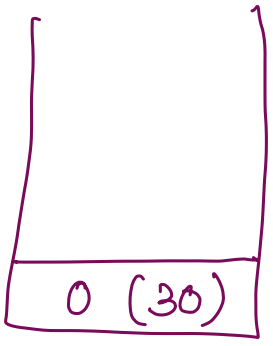
$$result[\text{poped element's index}] = \underset{\text{current element's index}}{i} - \text{poped element's index}$$

if we are left with any indices in stack
we say
they have no more temperatures left

① put elements by their index
② if you find any element above the temperature.
mark the difference between their indices
& mark it as the days for poped element
to get next higher temp.
③ add current element in stack.

[30, 38, 30, 36, 35, 40, 48]

result = [1. | | | | | |]

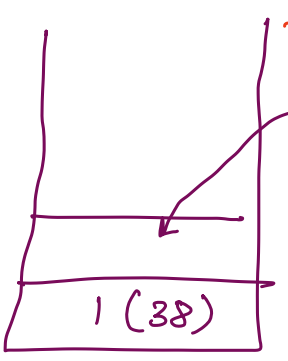


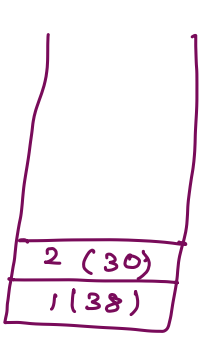
1
(38)

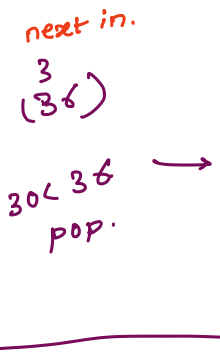
30 < 38
so pop

0th index → 1 − 0 = 1

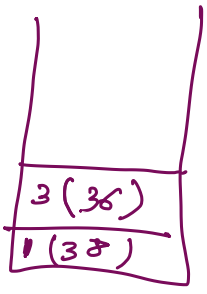
result = [1 | 1 | | | | |]



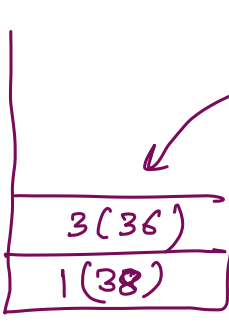


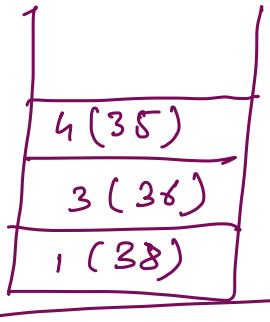


2nd index = 3 − 2 = 1.



result = [1 | 4 | 1 | 2 | 1 | | |]

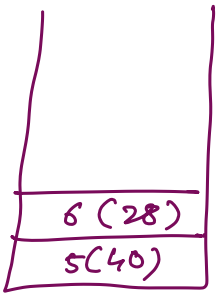




5
(40)
40 > 35
pop.

4th index → 5 − 4 = 1
3rd index → 5 − 3 = 2.
1st index → 5 − 1 = 4.

result = [1 | 4 | 1 | 2 | 1 | 0 | 0.]



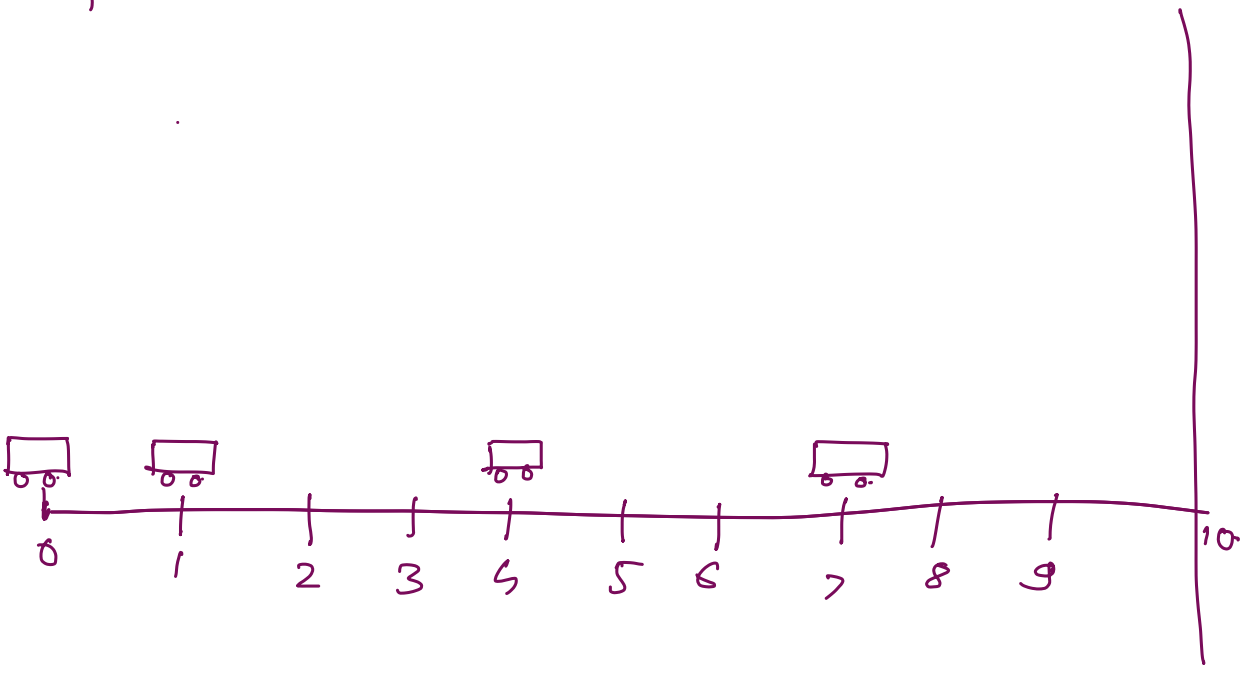
pop 6th index → 0
pop 5th index → 0

⑥ Car fleet

target = 10

position = [4, 1, 0, 7]

speed = [2, 2, 1, 1]



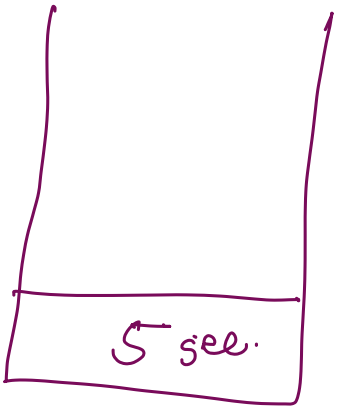
Basically

we sort the array by distance.

7, 4, 1, 0.

because '7' car has greatest chance to reach destination.

Now, we calculate time for each car to reach destination



other car

(4)

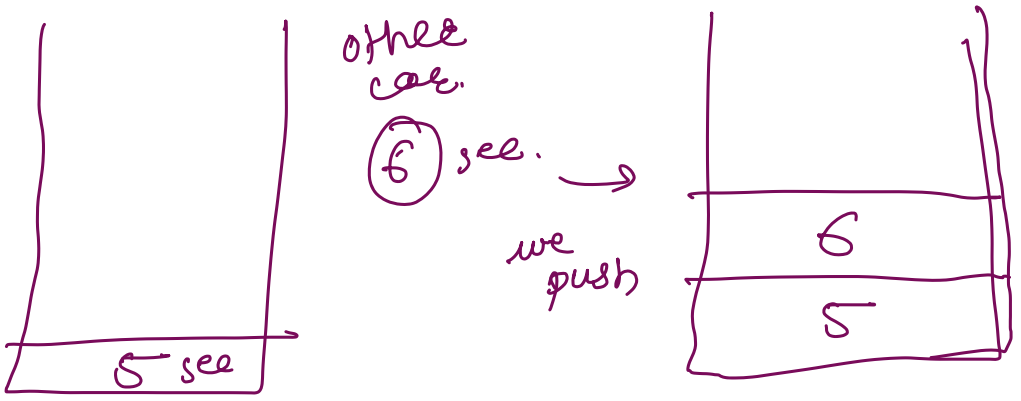
5 > 4.

don't push

because:

they will form fleet

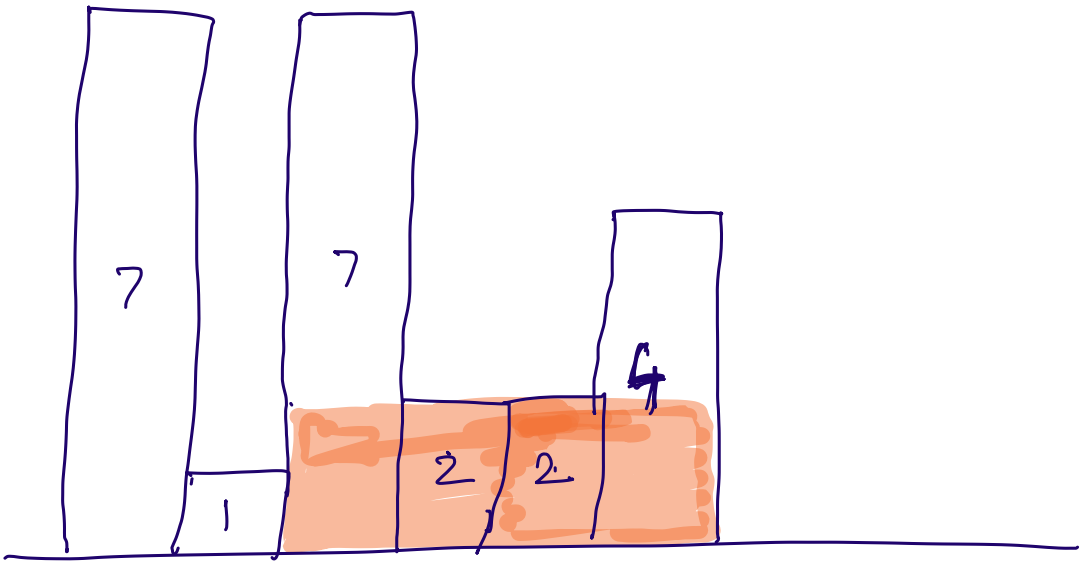
because they both will reach within 5 sec



& at the end

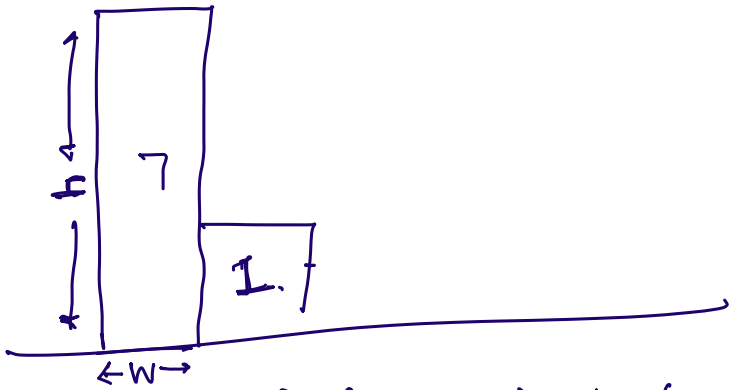
the size at stack would be the no. of fleets.

⑦ Largest rectangle in histogram -



1) keep adding the heights in stack

2) if you get smaller height than stack's peek
pop the stack
calculate the area before that current element



height (7) × width (1)

tricky part is to figure out width -

if stak is empty → i

Not empty → i - stackpeek() -