1- nearest
2- largest rectangle
3- sum max-min subarray

output array

P1 Given an integer array `a`, find the index of nearest smaller element on left for each `a[i]`. if not exist for `a[i]`, ret -1 (index)

i.e find max `j` . s.t , `a[j]<a[i]` , `j<i`

ex `a = {3, 10, 6, 1, 4}` (indices 0 1 2 3 4)

index {-1, 0, 0, -1, 3} → ans

i ← j

ex `a = {8, 2, 4, 9, 7, 5, 3, 10, 3}` (indices 0 1 2 3 4 5 6 7 8)

index of first smallest { -1 -1 1 2 2 2 1 6 1 }

look left

int a[] ← -1

for i = 0 → n-1

for j = i-1 → 0

if (a[j] < a[i]) { ans[i] = j; break; }

8, 2, 4, 9, 7

idea1

TC: $O(n^2)$

SC: $O(1)$

TC:

SC:

Quiz

a = { 8, 2, 4, 9, 7, 5, 3, 10, 3 } (indices 0 1 2 3 4 5 6 7 8)

18

8, 2, 4, ~~9~~, ~~7~~, 5

idea2

maintain possible answers → how? why?

Keeping most recent smaller number → Smells Stack

① maintain possible answers

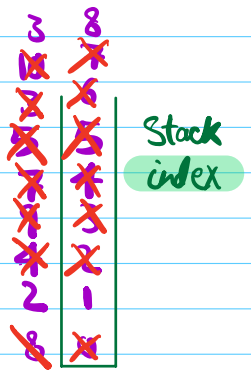
↓ how

② we can use stack. it is ok to pop out larger numbers from stack. because we are 100% sure the won't be a possible answer for right element

a = { 8, 2, 4, 9, 7, 5, 3, 10, 3 } (indices 0 1 2 3 4 5 6 7 8)

{ -1 -1 1 2 2 2 1 6 1 } ans

a[..]



Code

TC: O(n)

SC: O(n)

```
initialize a stack 'st'
for(i=0; i<n; i++){
    //a[i]
    while(!st.isEmpty() && a[st.peak()] >= a[i]){
        st.pop()
    }
    if(st.isEmpty) ans[i]=-1
    else ans[i] = st.peak();
    st.push(i)
}
```

n

push $\leq n$ pop $\leq n$

$\rightarrow \leq 2n$ of $O(1) \sim O(2*n) \sim O(n)$

index

(<=)

varieties

1 - find the index of nearest smaller element on left for each a[i]

2 - find the index of nearest smaller element on right for each a[i]

```
for(i=n-1; i>=0; i--){ ... }
```

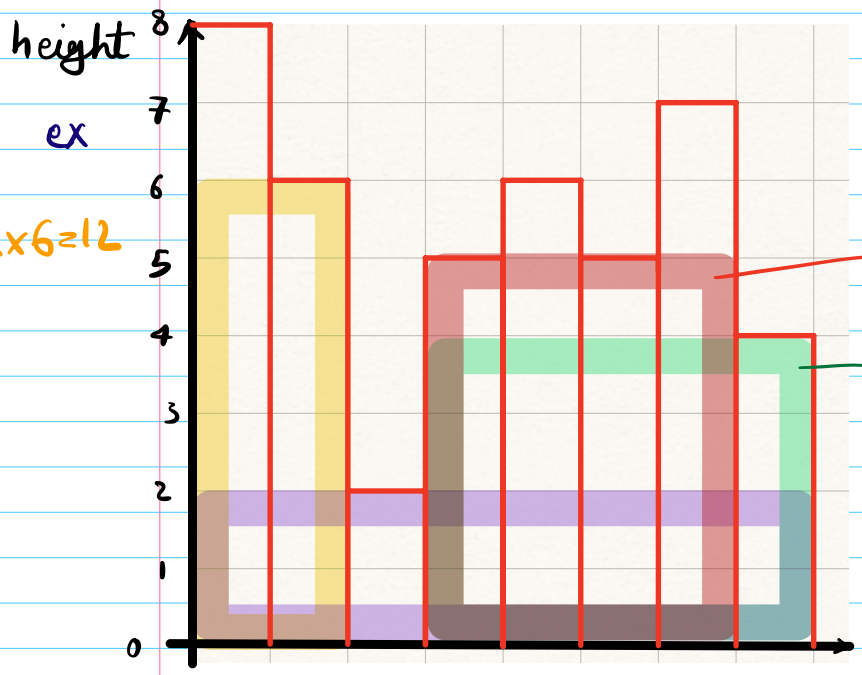
3 - find the index of nearest larger element on left for each a[i]

4 - find the index of nearest larger element on right for each a[i]

```
for(i=n-1; i>=0; i--){ ... }
```

a[i] > 0

P2 Given an integer array a, where a[i] is the height of ith rectangle (i.e. bar with width of 1) find the area of largest rectangle formed by the bars.



idea 1

for i = 0 → n-1
  for j = i → n-1
    h = min(a[i] ... a[j]) ← possible top height (O(n))
    ans = Max(ans, (j-i+1) * h)
          (j-i+1): len, h: height

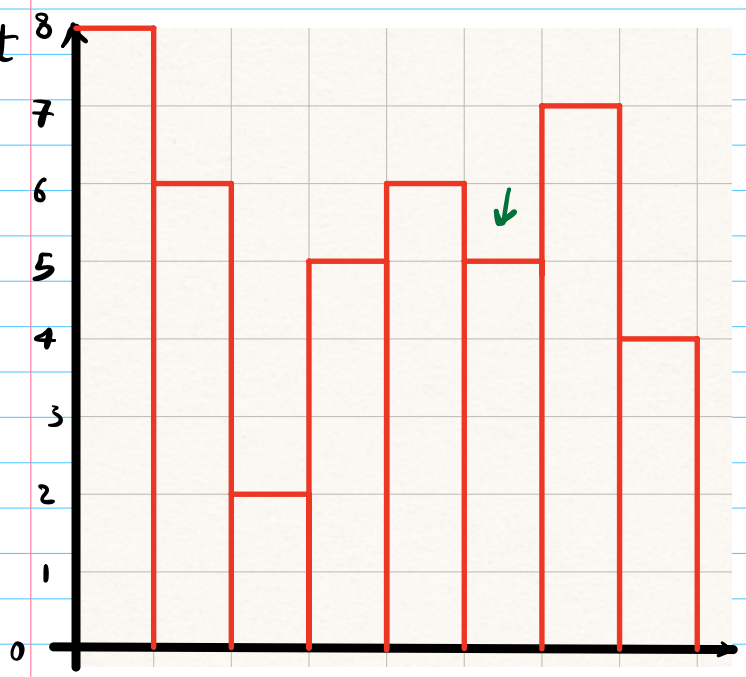
TC: $O(n^3)$
SC: $O(1)$

break?

iden 2

height

ex



a { 8 6 2 5 6 5 7 4 }

① 0 1 2 3 4 5 6 7

leftSmaller[] find all the first smaller item on left | O(n) P1 ver1

+ | +

② | O(n) P1 ver2

rightSmaller[] find all the first smaller item on right

+

③ for i=0 → n-1

- h = a[i]
- l = leftSmaller[i] + 1 → O(1) special case? -1?
- r = rightSmaller[i] - 1 → O(1)
- w = r-l+1
- ans = Max(ans, h*w)

TC: O(① n + ② n + ③ n) = O(n)

SC: O(n)

P3 Given an integer array a, with distict integers, for all sub arrays find (max - min) & return its sum as answer

int

ex $a: \{2, 5, 3\}$ (indices 0, 1, 2)

not subsets

$\{2,3\}$ is not sub arr

sub arrs	(+) max	(−) min	max − min
{2}	2	2	0
{5}	5	5	0
{3}	3	3	0
{2,5}	5	2	3
{5,3}	5	3	2
{2,5,3}	5	2	3
			8 = ans

② m_n → number of times n happens as min

① M_n → max

TC: $O(n^3)$

```
for i = 0 → n-1
    for j = i → n-1
        ans += max(a[i]...a[j]) - min(a[i]...a[j])
```

$O(n)$

max − min

$2 \times (M_2 - m_3) + 5(M_5 - m_5) + 3(M_3 - m_3)$ (indices 1, 3 / 4, 1 / 1, 2)

$-4 + 15 - 3 = 8$ ✓

how to find count of subarrays where a[i] is max element

ex $a = \{1, 8, 3, 5, 4, 2, 11, 7, 6\}$ (indices 0 … 8)

(start, end)

ss $\{3, 5\}$ 2

es $\{5, 4, 2\}$ 3

$2 \times 3 = 6$ M_5

M_5

① {5}
② {3,5}
③ {5,4}
④ {5,4,2}
⑤ {3,5,4}
⑥ {3,5,4,2}

Quiz

l r

r − l + 1

C1 this part of code

for each element a[i] with index i

#s = i − (index of) first larger element on left (+1) → +1

#e = first larger element on right (−1) → − i + 1 (l, +1)

sum += (s × e) × a[i] r → for max contribution

r − l + 1

how to find count of subarrays where a[i] is min element

ex $a = \{1, 8, 3, 5, 4, 2, 11, 7, 2\}$ — {5,4}, {4}

C2 this part of code

for each element a[i] with index i

#S = i − (index of first smaller element on left) + 1 — (i − l) + 1, r − l + 1

#e = (index of first smaller element on right) − 1 — − i + 1, l + 1

sum −= (S x e) x a[i] r — for min Contribution

---

sum = 0

1. (index of) first larger element on left — O(n)
2. (index of) first larger element on right — O(n)
3. (index of) first smaller element on left — O(n)
4. (index of) first smaller element on right — O(n)
5. C1 — O(n) — #M_n
6. C2 — O(n) — #m_n

TC = O(6n) ~ O(n)

SC = O(n)