Interview Problem on Arrays

Mango Trees

↳ You went to the farm wil

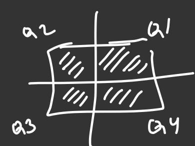
3 friends

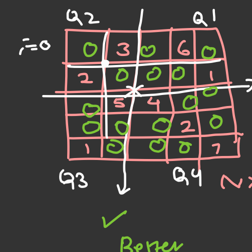
↳ 2 cuts in farm → one horizontal cut
anywhere → one vertical cut

↳ 4 quadrants

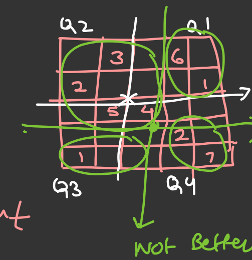
↳ Each friend will get one quadrant ← greedy

↳ sum of mangoes in one quadrant





N×N fruit farm



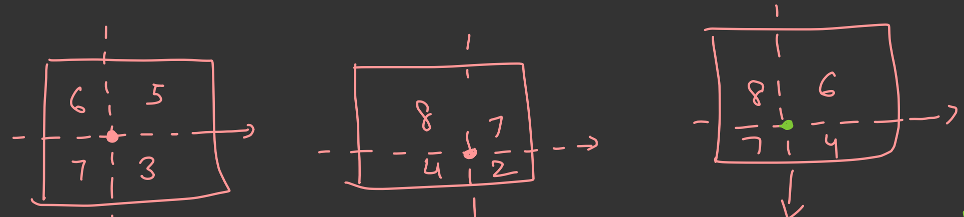
1 →
9
7
14

Example –

6+1 = 7 → F
2+3 = 5 → Lowest one → You will get
1+5 = 6 → F
4+2+7 = 13 → F

You can make cut anywhere

Ⓠ Can you 'maximise your min' mangoes, based upon cut. You will always get the min quadrant, as your friends are greedy.



You → 3

You → 2

You → 4

Best Cut

⇒ Cut the field in such a way that we get max possible mangoes.

Brute Solution → Make a cut at every x, y

cut at every possible intersection

N Rows

N Cols

Total = Σ a[i][j]   max = 0

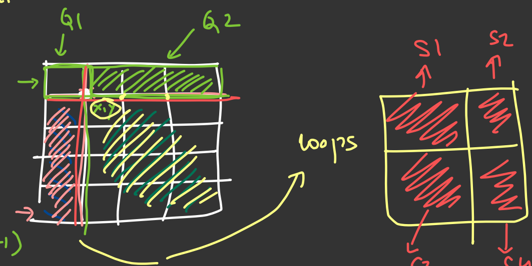
$O(N^4)$

N ← for(x=0, x<N-1 x++){

N ← for(y=0; y<N-1 y++)(

made at cut (x,y)

Sum N^2 ←
- Q1 → sum (0,0 — x,y)
- Q2 → sum (x,y+1 to x,N-1)



loops

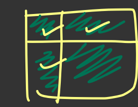
$Q3 \rightarrow \text{Sum}(x+1, y \text{ to } N-1, y)$

$Q4 = \text{Total} - (Q1+Q2+Q3),$

↓

precomputed

// x+1, y+1 --- N-1, N-1



N^2

Max of all mins

$\rightarrow$

mangoes = min(Q1, Q2, Q3, Q4);

if(mangoes > maxMangoes) { maxMangoes = mangoes },

3

3

N^2 cut   for each cut finding what is their in each quad N^2

$= N^2 \cdot N^2$

$= O(N^4)$

Sum(arr, x1, y1, ~~x2, y2~~)

$N^2 \leftarrow$

for(x) → x1 — x2

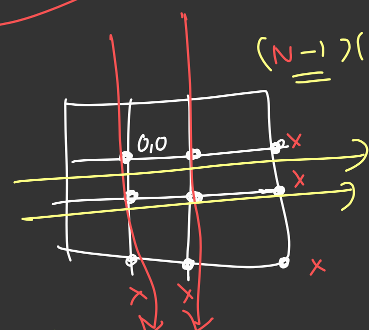
for(y) → y1 — y2

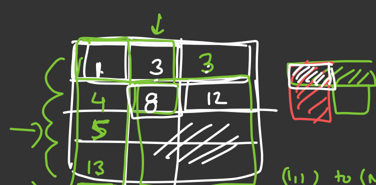
sum = sum + a(x)(y),

3

$(N-1)(N-1)$

$(N-1)^2$ cuts



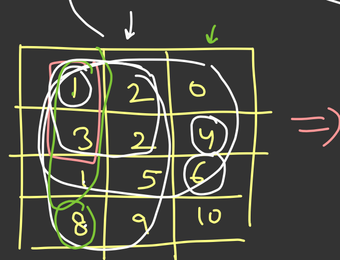


(1,1) to (N-1, N-1)

Loops → Expensive

$O(N^2)$

1+2+0
+3+2+4
+1+5+6
= 24

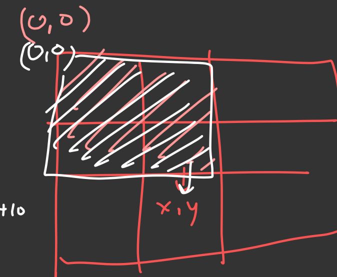


farm

O(1)
14+13−5+9
= 31

24+31−14+10 =

12+14−8+6 ← O(1)
= 24



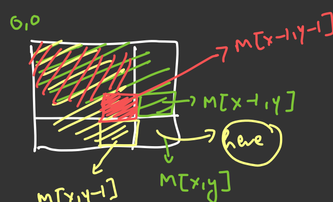
Sum

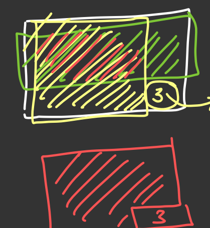
M [x,y] = Green + Yellow − Red + farm[x,y]

= M[x,y−1] + M[x−1,y] − M[x−1,y−1] + farm[x,y]

M → Sum

S [x,y] = sum of all mangoes from 0,0 to x,y





Green + Yellow − Red + 3

2D Prefix Sum

Step-1 Prefix Sum Matrix

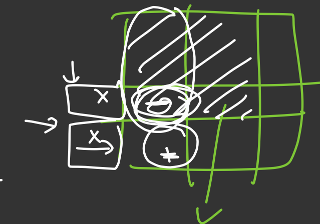
mat[N][N] = {0}

$O(N^2)$

```
for(x=0; x<N-1, x++) {
    for(y=0; y<N-1, y++) {
        M[x,y] = M[x,y-1] + M[x-1,y] - M[x-1,y-1] + arr[x,y]
    }
}
```

M[x,y] → new array

don't exist

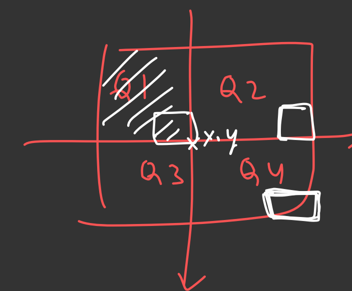


M[x,y]

M[0][0] = arr[x,y]

M[2][0] = M[1][0] + arr[2][0]

Step-2 **Make cuts**



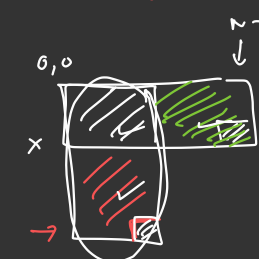

```
for( x ____ )
    for(y=0 ____ ) {
```

$O(N^2)$

O(1) step

- Q1 = M[x,y]
- Q2 = M[x,N-1] - Q1
- Q3 = M[N-1,y] - Q1
- Q4 = M[N-1,N-1] - Q1 - Q2 - Q3

↳ same logic → Track Min & Update Max



3

3

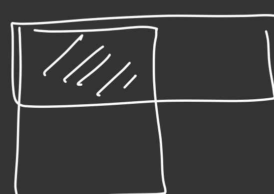
$O(N^2 + N^2) \longrightarrow O(N^2)$

2D Prefix Sum

Optimisation

$O(N^4)$ $O(1)$ space

$O(N^2)$ time $O(N^2)$ space



$\Rightarrow A \cup B = |A| + |B| - |A \cap B|$

$K = 3$

Q Given N Array elements, find max subarray sum of len = K.

0	1	2	3	4	5	6	7	8	9

arr = { -3, 4, -2, 5, 3, -2, 8, 2, -1, 4 }

n-3 n-2 n-1

0

Basic Approach

1. Find all subarrays of len k
   - ↳ of $\boxed{\text{len} = K}$
     - ↳ sum.

[0 2]
[1 3]
[2 4]
:
[i i+k-1]
:

$\propto N$

$O(NK)$

N times

K times

```
for(i=0; i<=n-k; i++){
  x -> j = i+k-1
  sum = 0
  for(k = i ---- j){
     sum = sum + a[k];
  }
  if (sum > largest) -> update largest
```

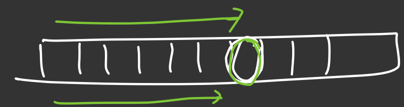
3

Better

Prefix Sum

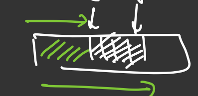
Step-1

O(N)



$ps[i] = ps[i-1] + a[i]$

Step-2 find out subarray.



O(N)

```
for (i=0, i<=N-K; i++){
    j = i+K-1
    → sum = ps[j] - ps[i-1],     // handle for i-1 is -ve
    ⇒ update largest   if (sum > largest) { largest = sum },
}
```

= O(N) time

= O(N) space → prefix array

Example

arr | 1 | 3 | 4 | 6 | -2 | -3 | 7 |

ps | 1 | 4 | 8 | 14 | 12 | 9 | 16 |

$ps(i) = ps(i-1) + a(i)$

$O(1) \rightarrow ps[j] - ps[i-1]$

$12 - 4$

$= 8$

optimised

wop

$4 + 6 - 2$

$= 8$ ✓

Sliding Window

$k = 3$

i=0 1 2 3 4

| 1 | 3 | 4 | 6 | -2 | -3 | 7 |

0 1 2 ✓

i-3 i-2 i-1 i

| - | | + |

prev sum + a[i] - a[i-k]

Better

O(N) time

O(1) space

Sum[0,2] $\boxed{8}$ -

sum[1,3] $\boxed{8}$ + 6 - 1 = 13

Sum [2,4] $\boxed{13}$ + a[4] - a[4-3]

13 + a[4] - a[1] = 13 - 2 - 3

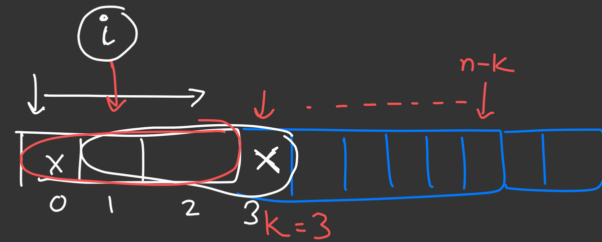
= 8

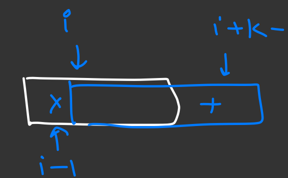
⋮

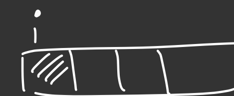
window that is moving towards Right

+ Add

− Subtract







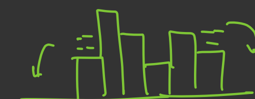
[0, k-1]
[1, k]
[2, k+1]
⋮

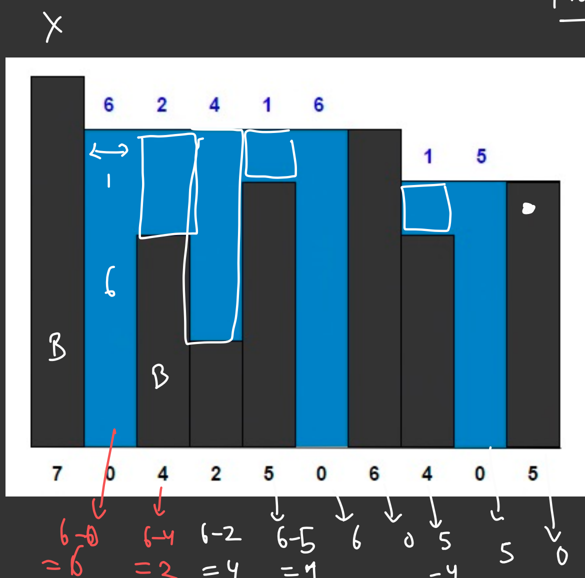
```
sum = 0
for( i=0; i<k; i++){
    sum = sum + a[i];
}
ans = sum;
for( i=1; i<=n-k; i++){
    sum = sum + arr[i+k-1] - arr[i-1];
    ans = max(ans, sum);
}
```

O(N) time
O(1) space

10:40

Rainwater Trapping





Solution

buildings = [7, 0, 2, 5, 0, 6, 4, 0, 5]

total water trapped.

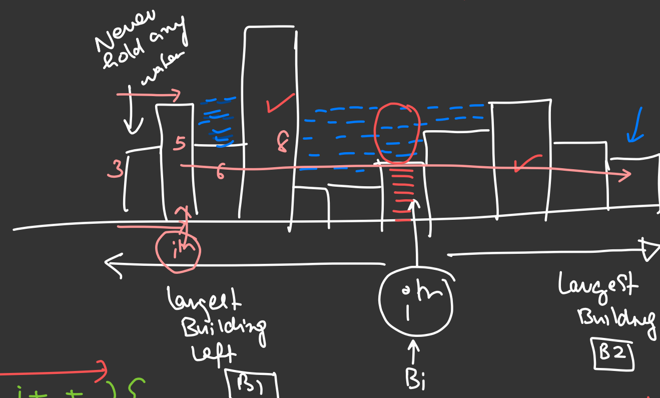
= 1

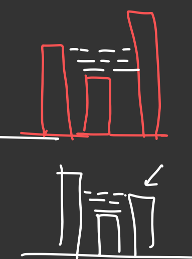
$6+2+4+1+6+1+5$

$= \underline{25 \text{ unit}}$

Algorithm → we need to water for $\boxed{i^{th} \text{ building}}$

↳ Add for every building





B1 => 5

B2 => 8

min(B1,B2) − bi

= 5 − 5

= 0

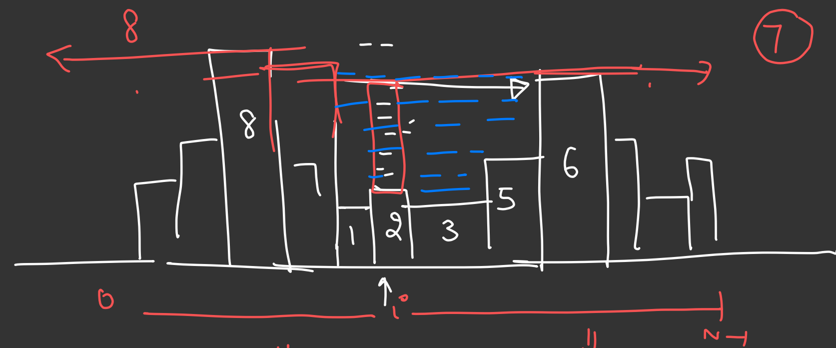
N water = 0

```
for(i=1; i<=n-2; i++){
    N { B1 -> Find largest element (0, i) => loop
        B2 ->  "      "       "    (i, N-1) => loop
    water = water + min(B1,B2) - bi
}
print(water)
```

$\underbrace{\text{min(B1,B2)}}_{\text{water level}} - \underbrace{\text{bi}}_{\text{Building ht}}$

$\boxed{O(N^2)}$

$= \underbrace{\overbrace{\min(B1,B2)}^{\text{water height}} - \overset{\text{Building Ht}}{B_i}}_{\text{water stored}}$



7 min(8, 7)

= 7 − 7

= 0

min(8 , 6) − Bi

= 6 − 2

= $\boxed{4}$

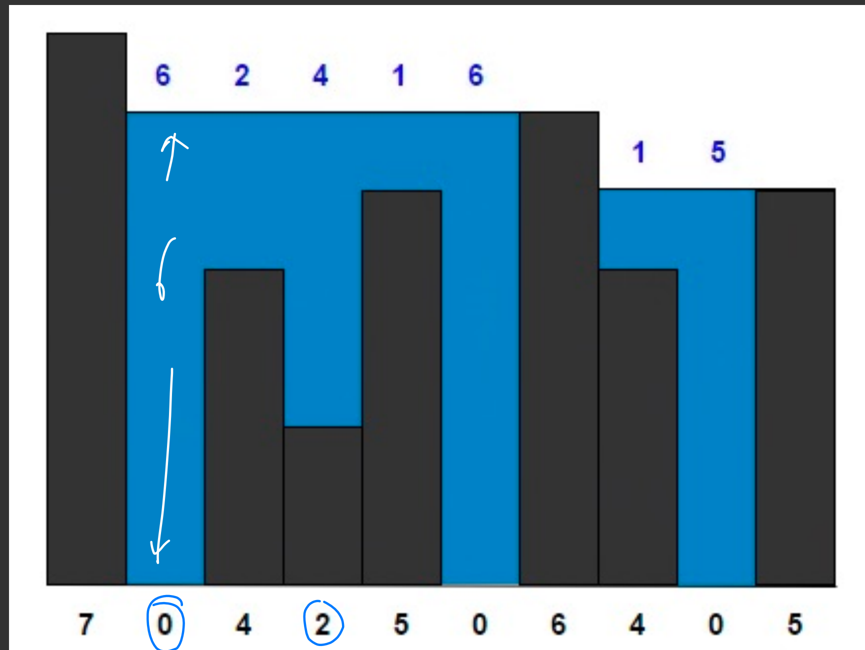
Better

Carry forward idea

easy →

max (0, i) for every i

max (i, N−1) for every i



arr →

Max	left B1 (0, i)	7	$\boxed{7}$	7	7	7	7	7	7	7	7
Max	Right B2 (i, N-1)	7	$\boxed{6}$	6	6	6	6	6	5	5	5

left(i) = max(a(i), left(i-1))

min (B1, B2) → 7 6 6 6 6 6 6 5 5 5

water − 7, 0, 4, 2, 5, 0, 1, 4, 0, 5

Bi

0 + 6 + 2 + 4 + 1 + 6 + 0 + 1 + 5 + 0 ← water at i^{th} Building

= 25 units

Left Max → O(N)

Right max → O(N)

water level → for (i = 0 —— N−1) {

water = water + min(leftMax(i), RightMax(i)) − b[i];

}

O(N) time ←

O(N) space

Carry forward Technique

☆ Given a binary array of size N, we can replace a single '0' with 1, find the max consecutive ones that we get in the array

7ones

0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0

4 ones

6 ones.

3 ones

max → 4, 6, 5, 3

⇒ 6

Brute force

iterate over entire array

↳ if i get 0

↓ counted how many 1's can be formed.

Track

3 6 Replace this 4

0 1 1 1 1 $\boxed{0}$ 1 1 1 0 0 1 1 0

Left + 1 + Right

← i-1 i (i+1) →

ans = ~~0~~ ~~3~~ $\boxed{6}$

Acceptable

TC = O(N)

Space = O(1)

ans = 0

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

if (a[i] == 0){

L → count 1s in left of i

L=0 , j=i-1

while (j>=0 && a[j]==1){ j--, L++ };

R → count 1s in Right of i

R=0, j=i+1

while (j<n && a[j]==1){

R++;

j++;

}

current = L+R+1

if (current > ans){

ans = current;

}

}

}

~~O(N^2)~~

$\boxed{O(N)}$

0 1 1 1 1 0 1 1 1 1 0

j=i-1

1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 0

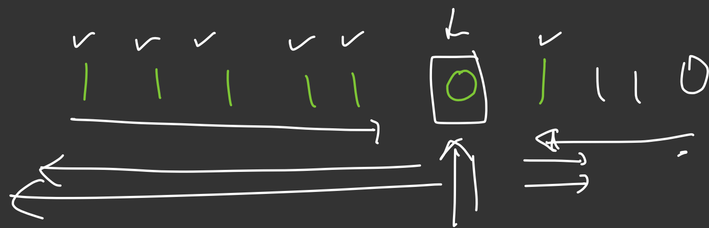
At max how many time each element is visited $\rightarrow \boxed{2} + 1$

$= 3$ times

$= 3N$

$= O(N)$

5 steps



Two times

$2N$

$= O(N)$

Good Night ☆ ☆ ☆

Long Weekend :)