Given arr[n] : 0 1 2 3 4 5 6 7 8 9
-3 6 2 4 5 2 8 -1 3 1

Q: Find sum of all subarrays from l to r

Each query [l, r]
Find sum of all subarrays from l to r

Q: 1 3 → Sum
1 → 3 : 12
2 → 4 : 12
4 → 8 : 9
0 → 9 : 19
3 → 9 : -9

For every query [l, r]
Traverse from l → r
while(l <= r) { Repeating subarrays loop
s = 0
for i, j = l, r, 1
s = s + arr[i]
print(s)
}
TC : O(n * n)
SC : O(1)

Given arr[n] : 0 1 2 3 4 5 6 7 8 9
-3 6 2 4 5 2 8 -1 3 1

Pf : Prefix Sum : -3 3 5 9 14 16 24 15 18 19

Sum → pf[r] - pf[l-1]

pf[0] = 0-0
pf[1] = 0-1 → Sum of all subarrays [0-0]
pf[2] = 0-2 → Sum of all subarrays [0-1]
pf[3] = 0-3 → Sum of all subarrays [0-2]
Sum pf[r] & you pf[l-1]
pf[r] - pf[l-1] = arr[l,r]
[0-2] [0-3] → [2,3]

Approach 1:
for i = 1, n, 1
pf[i] = pf[i-1] + arr[i]
print(pf)
TC : O(n)
SC : O(n)

Approach 2:
for i = 0, n-1, 1
pf[i] = arr[i]
for i = 1, n, 1
pf[i] = pf[i-1] + arr[i]
print(pf)
TC : O(n)
SC : O(n)

Note: pf[0] = arr[0]
pf[i] = pf[i-1] + arr[i]

Approach Pf[i-1]

Approach 1:
L - R : pf[R] - pf[L-1]
[0 - 4] : pf[4] - pf[-1] ×
→ pf[4]
Conclusion:
Given L & R
if (L == 0)
ans = pf[R]
else
ans = pf[R] - pf[L-1]

Approach 2:
L - R : pf[R] - pf[L-1]
[0 - 4] : pf[4] - pf[-1] ×
→ pf[4]
Conclusion:
Given L & R
if (L == 0)
ans = pf[R]
else
ans = pf[R] - pf[L-1]

Q: 2
[1 - 3] : pf[3] - pf[0] = sum of all [3-1] + arr[1]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9] = [0, 1, 2]
[1 - 5] : pf[5] - pf[0] = arr[1] = sum of all [5-1] + arr[1]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9] [0, 1, 2, 3, 4]

Approach Conq
Given L & R
ans = pf[R] - pf[L-1] + arr[L-1] →
Time Conq : [0 - 4]
pf[4] - pf[-1] + arr[-1]
[0, 1, 2, 3, 4] = [0-4] + [0-0]

Q: 3
Given arr[n] : 0 1 2 3 4 5 6 7 8 9
0 0 0 0 0 0 0 0 0 0
-4 2 2 2 2 2 2 2 1 -2
-4 2 2 2 2 2 2 2 -1 -3
1 1 1 0 0 0

Q → Queries
→ l, R, x
for in l, R, x
Traverse from l → R and add x
while(Q -> {
l, R, x = Q.pop()
for i = l, R, 1
arr[i] = arr[i] + x
}
print(arr)
}
TC : Q * O(n) → O(Q * n)

Plain Brute Approach
Use for every query
Traverse from l → R and add x
while(Q -> {
l, R, x = Q.pop()
for i = l, R, 1
arr[i] = arr[i] + x
}
print(arr)
}
TC : Q * O(n) → O(Q * n)

0 1 2 3 4 5 6 7 8 9
0 -4 2 2 0 0 -2 0 2 -1
-4 2 2 0 0 0 -2 -1 -3
-4 2 2 0 0 0 -2 -1 -3
1 1 1 0 0 0

arr[] : 0 -4 2 2 0 0 -2 0 2 -1
pf[] : 0 -4 -2 0 0 0 -2 -2 0 -3

Q
l R x
1 3 2 → arr[1] += 2 arr[2] += 2
0 4 -1 → arr[0] += -1 arr[4] += -1
2 5 1 → arr[2] += 1 arr[5] += 1
1 2 -3 → arr[1] += -3 arr[2] += -3

Apply pf[i] in modified array.

Brute Force Code :
while(Q -> {
l, R, x = Q.pop()
for i = l, R, 1
arr[i] += x
if (R < N-1)
arr[R+1] -= x
}
Construct pf[]
TC : Q * O(n) → O(Q * n)

0 1 2 3 4 5
s = 0
arr[] : 0 -4 2 2 0 0
Sum 0 -4 -2 0 0 0
Conq 0 -4 -2 0 0 0

s = s + arr[i]
pf[i] = s

In simple words taking prefix sum in arr[] itself.
SC : O(1)

TC : O(N+Q)

Q2 : Sliding Window

Why for non-zero or non-one

0 1 2 3 4 5
2 1 2 1 2
-1 -1

Q : 1 3 2

2 7 9 11 10 12

Ideally : 2 5 5 5 2 2

For non zero elements
V. Imp

1) Step 1 → Copy all elements in new array
2) In original array minimize all arr[i]
3) Update all que
4) Taking prefix sum
5) Add the original array with final prefix sum values

Eg : arr[] : 0 1 2 3 4
4 2 1 2 6
5 5 3 4 5

Q
1 2 2
2 4 -1
0 2 1

0 1 2 3 4
4 2 1 2 6
1 -1 -1
1 3 2 1 -1
4 2 1 3 6
5 5 3 4 5

Q3 :
0 1 2 3 4 5 6 7 8
arr[] : 2 5 4 3 8 -1 3 2 0

Contains lft[i] & right[i] arrays
Zero Array.

lft[0] = [0, 0] : 2
lft[1] = [0, 1] : 6
lft[2] = [0, 2] : 6
lft[3] = [0, 3] : 8
lft[4] = [0, 4] : 8
lft[5] = [0, 5]

lft[0] = [0, 0] : 2
lft[1] = [0, 1] : max(lft[0], arr[1])
max(2, 5)
lft[2] = [0, 2] : max(lft[1], arr[2])
max(6, 4)
lft[3] = [0, 3] : max(lft[2], arr[3])
max(6, 3)
lft[4] = [0, 4] : max(lft[3], arr[4])
max(8, 8)
lft[5] = [0, 5] : max(lft[4], arr[5])
max(8, -1)
lft[6] = [0, 6] : max(lft[5], arr[6])
max(8, 3)
lft[7] = [0, 7] : max(lft[6], arr[7])
max(8, 2)
lft[8] = [0, 8] : max(lft[7], arr[8])
max(8, 0)

Approach
max(a, b) : return max(a, b)
min(a, b) : return min(a, b)
sort( ) , abs( )
TC : O(N*log N)
INT_MAX , INT_MIN

0 1 2 3 4 5 6 7 8
arr[] : 2 5 4 3 8 -1 3 2 0

Approach 1:
for i = 1, n, 1
lft[i] = max(lft[i-1], arr[i])
for i = n-1, 0, -1
rgt[i] = max(rgt[i+1], arr[i])
for i = 0, n, 1
ans = ans + min(lft[i], rgt[i]) - arr[i]
print(ans)
TC : O(n) + O(n) + O(n) → O(3n)
SC : O(n) + O(n)

Approach 2:
for i = 1, n, 1
lft[i] = max(lft[i-1], arr[i])
rgt[n-1] = arr[n-1]
for i = n-2, 0, -1
rgt[i] = max(arr[i], rgt[i+1])
for i = 0, n, 1
ans = ans + min(lft[i], rgt[i]) - arr[i]
print(ans)
TC : O(n) + O(n) + O(n) → O(3n)
SC : O(n) + O(n)

Q4 :
0 1 2 3 4 5 6 8
arr[] : 2 5 4 3 8 -1 3 2 0

Approach 1:
rgt[n-1] = 0 - 0
rgt[n-2] = 8 - 8
rgt[n-3] = 3 - 8 : 8
rgt[n-4] = 8 - 8 : 8
rgt[n-5] = 6 - 8 : 3
rgt[n-6] = 0 - 8 : 8
rgt[n-7] = max(3, 8)
for i = n-2, 0, -1
rgt[i] = max(arr[i], rgt[i+1])
rgt[n-1] = arr[n-1]
for i = n-2, 0, -1
rgt[i] = max(arr[i], rgt[i+1])
print(rgt)

Approach 2:
rgt[n-1] = 0
rgt[n-2] = 8
rgt[n-3] = 3
rgt[n-4] = 8
rgt[n-5] = 6
rgt[n-6] = 8
rgt[n-7] = 3
rgt[n-8] = 2
rgt[n-1] = arr[n-1]
for i = n-2, 0, -1
rgt[i] = max(arr[i], rgt[i+1])
print(rgt)

Q5 : Given a string S which contains only Non Alphabetic Characters
Find Minimum swaps required to make all Non Alphabetic characters together.

0 1 2 3 4 5
Ex : a b c g a g
Ans : 0 1 2 3 4 5
0 1 2 3 4 5
Ans : 0 1 2 3 4 5
0 1 2 3 4 5
Ans : 0 1 2 3 4 5

Ans = 5

Solution : Check all pairs
for i = 0, i < N, i++ {
if (S[i] == 'a') {
for j = i+1, j < N, j++ {
if (S[j] == 'g') {
swap(i, j)
}
}
}
}

if S[i] == 'a'
for j = i+1, j < N, j++
if S[j] == 'g'
swap(i, j)
break

range = 0
for i = 0, i < N, i++ {
if (S[i] == 'a') {
for j = i+1, j < N, j++ {
if (S[j] == 'g') {
swap(i, j)
range = range + (j - i - 1)
break
}
}
}
}
print(range)

Q6 : Given an array find the length of longest subsequence such that contains both the 2 max elements of array

Ex : 0 1 2 3 4 5 6 7 8
{ 1, 2, 3, 1, 3, 4, 5, 6 }

Ex : 0 1 2 3 4 5 6 7 8
{ 2, 3, 6, 4, 5, 1, 5, 7 }

Ex : 0 1 2 3 4 5 6
{ 1, 3, 3, 1, 3, 4, 3 }