Problem 1 Count of Pairs ag

Problem 2 Find sum of all Subarrays sums

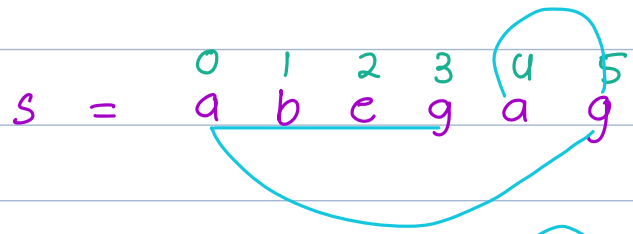
Problem 3 Max subarray sum with length K

Count of pairs ag

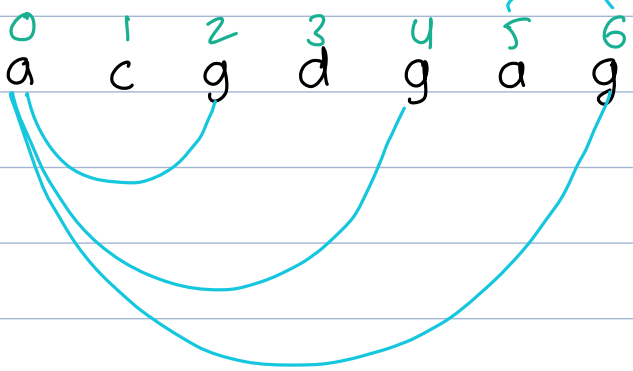
Given a string s of lowercase characters, return the count of pairs (i, j) such that

$i < j$ and $s[i] == 'a'$ and $s[j] == 'g'$

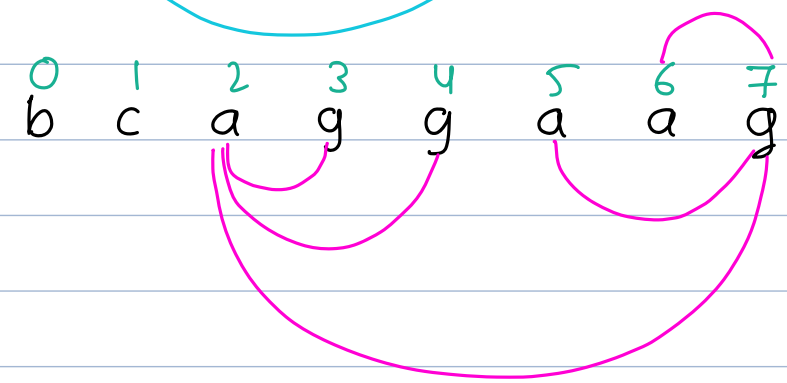
String s = a b e g a g (indices 0 1 2 3 4 5) ans = 3



s = a c g d g a g (indices 0 1 2 3 4 5 6) ans = 4

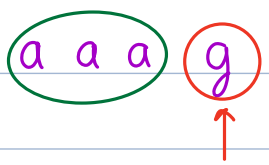


s = b c a g g a a g (indices 0 1 2 3 4 5 6 7) ans = 5

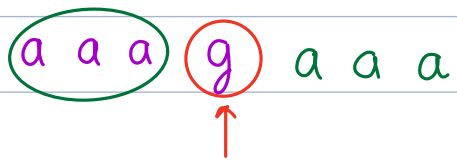


---

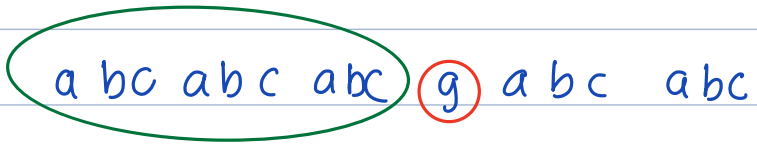
a a a g ans = 3



a a a g a a a ans = 3



a bc ab c abc g a b c abc ans = 3



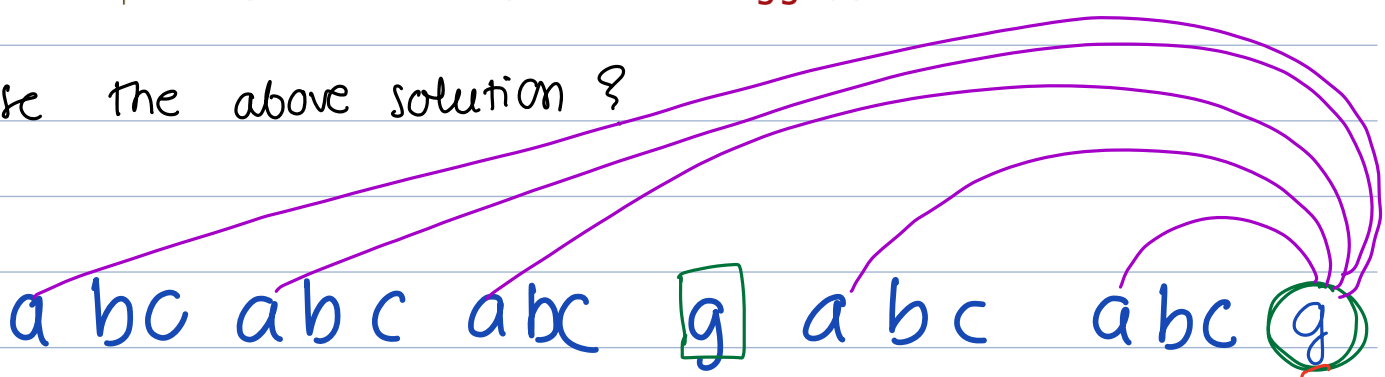
Brute force

```python
def bruteforce(s):
    ans = 0
    N = len(s)

    for i in range(N): # 0...N-1
        if s[i] == 'g':
            count_a = 0
            for j in range(i): # 0...i-1
                if s[j] == 'a':
                    count_a += 1

            ans += count_a

    return ans

print(bruteforce("abcabcabcgg"))
```

TC : $O(N^2)$

SC : $O(1)$

Optimise the above solution ?



Idea ⟶ maintain the count of a and whenever we encounter g, we increment the ans by count of a

Optimised

```python
def count_ag_pairs(s):
    """
    Given a string s of lowercase characters, return
    """
    ans = 0
    count_a = 0

    for ch in s:
        if ch == 'a':
            count_a += 1

        if ch == 'g':
            ans += count_a

    return ans

print(count_ag_pairs("abcabcabcgg"))
```

TC: O(N)

SC: O(1)

Subarray

a continuous part of an array.

A = 4 1 2 3 −1 6 9 8 12

subarray of A

→ 4 1 2 3 ✓

3 −1 ✓

12 ✓

8 9 ✗

A = [2 4 1 6 −3 7 8 4]

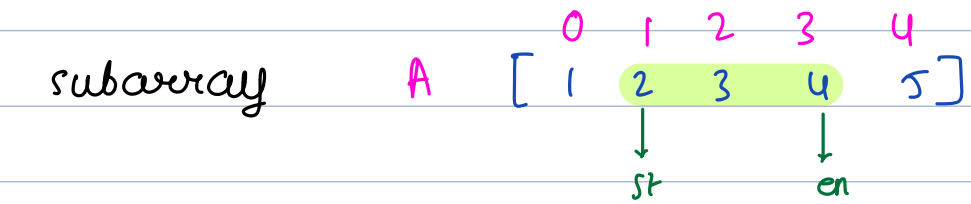
1 6 8 ✗

1 4 ✗

6 1 4 2 ✗

7 8 4 ✓

Representation of subarray



Ways to represent a subarray.

1> start = 2 end = 4

A[1 : 4]

2> start = 1 length = 2

A[st : en+1]

	0	1	2	3	4	5	6
A =	4	2	10	3	12	-2	15

L

R

L, R							
0, 0	4						
0, 1	4	2					
0, 2	4	2	10				
0, 3	4	2	10	3			
0, 4	4	2	10	3	12		
0, 5	4	2	10	3	12	-2	
0, 6	4	2	10	3	12	-2	15

L = 0

R = [0 ..... 6) # 7

	0	1	2	3	4	5	6
	4	2	10	3	12	-2	15

L = 1

R = [1 ..... 6) = # 6

1, 1	2					
1, 2	2	10				
1, 3	2	10	3			
1, 4	2	10	3	12		
1, 5	2	10	3	12	-2	
1, 6	2	10	3	12	-2	15

No. of subarrays of any array.

$$A = [\overset{0}{a_0} \quad \overset{1}{a_1} \quad \overset{2}{a_2} \quad \ldots\ldots \quad \overset{n-1}{a_{n-1}}]$$

subarrays will start at index $0 \longrightarrow n$

————— " ————— $1 \longrightarrow n-1$

⋮ $\quad n-2$

⋮ $\quad$ ⋮

$n-1 \longrightarrow 1$

Sum of all natural no. till $n = \dfrac{n*(n+1)}{2}$

No. of subarrays of an array of size n – $\dfrac{n*(n+1)}{2}$

Count of all subarrays.

Q> Given an integer array. Find the sum of all possible subarray {continuous part of array} Google

A = [3 2 5]

	sum
[3]	3
[3 2]	5
[3 2 5]	10
[2]	2
[2 5]	7
[5]	5
	32

ans = 32

Bruteforce

```
def bruteforce(A):
  N = len(A)
  overall = 0

  for L in range(N): # 0....N-1
    for R in range(L, N): # L...N-1
      sub = A[L:R+1]
      print(sub, sum(sub))
      total = sum(sub)
      overall += total

  return overall

print(bruteforce([3, 2, 5]))
```

```
[3] 3
[3, 2] 5
[3, 2, 5] 10
[2] 2
[2, 5] 7
[5] 5
32
```

TC : O(N3)

SC : O(N)

Note : Slicing is not O(1) operation.

Prefix Sum

Intuition —— ∵ we are finding sum (L.....R) in A
we can use prefix sum to optimize

```python
def prefix_sum(A):
  N = len(A)

  for i in range(1, N):
    A[i] += A[i-1]

  overall = 0

  for L in range(N): # 0....N-1
    for R in range(L, N): # L...N-1
      if L == 0:
        total = A[R]
      else:
        total = A[R] - A[L-1]
      overall += total

  return overall

print(prefix_sum([3, 2, 5]))
```

```
32
```

TC : O(N²)

SC : O(1)

0	1	2
3	2	5

{carry forward}

L	R	total
0	0	3
0	1	5
0	2	10
1	1	2
1	2	7
2	2	5

ans = 32

Carry Forward { calculate + use }.

A = [1 2 3] (0 1 2)

```
def carry_forward(A):
  N = len(A)
  overall = 0
  for L in range(N): # 0....N-1
    total = 0
    for R in range(L, N): # L...N-1
      total += A[R]
      overall += total

  return overall

print(carry_forward([3, 2, 5]))
```

```
32
```

TC: O(N²)

SC: O(1)

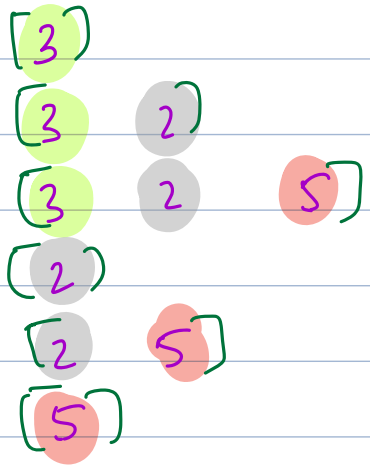
L	R	total
0	0	3
0	1	5
0	2	10
1	1	2
1	2	7
2	2	5

If my query L.... R , L remains fixed and R increments

⇒ you can use carry forward

Further Optimization using Contribution Technique

$A = [3 \quad 2 \quad 5]$



How many subarrays contain 3 ?

3

How many subarrays contain 2 ?

4

How many subarrays contain 5 ?

3

$\Rightarrow \quad 3 * 3 \quad + \quad 2 * 4 \quad + \quad 5 * 3$

$\Rightarrow \quad 9 \quad + 8 + 15$

$= \quad 32$

Contribution Technique

$$\sum_{i=0}^{n-1} (\text{contribution of } A[i])$$

$A[i]$ * # subarrays containing $A[i]$

How to calculate the no. of subarrays containing A[i]?

break: 9:53

subarrays containing A[1]

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

2 (L L) (R R R R R) 5

2 * 5 = 10

sub		
0 1	1 ... 1
0 2	1 ... 2
0 3	1 ... 3
0 4	1 ... 4
0 5	1 ... 5

subarrays containing A[2]

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

3 (L L L) (R R R R) 4

3 * 4 = 12

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

choice of my start index = [0 .... 2] = #3

choice of my end index = [2 .... 5] = #4

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

= 3 * 4 = 12

in range [a .... b] = b - a + 1

→ A_0 A_1 A_2 .... A_i .... A_{n-2} A_{n-1}

[0 ..... i]
i - 0 + 1
i + 1

*

[i ..... n-1]
n-1 - i + 1
n - i

No. of subarrays that will contain A[i] = (i+1) * (n-i)

Contribution of A[i] = A[i] * # subs it part of
= A[i] * (i+1) * (n-i)

A = [3, 2, 5] (0, 1, 2)

i	A[i]	# subs A[i]	# contrib A[i]
0	3	(1) * (3) = 3	3*3 = 9
1	2	2 * 2 = 4	2*4 = 8
2	5	3 * 1 = 3	5*3 = 15

Pseudocode

```
def sum_of_all_subarray_sums(A): # Contribution Technique
  """
  Given an array of integers, find the total sum of all po
  """
  N = len(A)
  overall = 0

  for i in range(N):
    overall += A[i] * (i+1) * (N-i)

  return overall

print(sum_of_all_subarray_sums([3,2,5]))
```

```
32
```

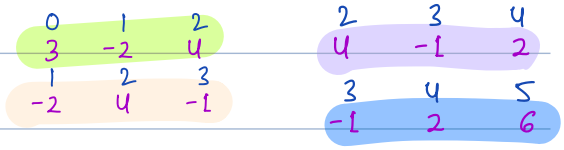
TC: O(N)
SC: O(1)

whenever you see sum of all sums ⇒ Think contribution Technique.

Q) Total # subarrays of length K (<= N) ?

	0	1	2	3	4	5
A = [	3	-2	4	-1	2	6]

K = 3



Output 4

	0	1	2	3	4	5
A = [	3	-2	4	-1	2	6]

K = 4

K	# subarrays.
1	6
2	5
3	4
4	3
5	2
6	1

$N - K + 1$

Total no. of subarrays of length == K $N-K+1$

A_0 A_1 A_2 A_3 .... A_{N-K} A_{N-K+1} .... A_{N-1}

L	R
0	K-1
	⋮
	N-1

first subarray of size K.

$\#[K-1 \ldots N-1] = N-1-(K-1)+1$

$= N-1-K+1+1 = N-K+1$

Given A[], print start and end indices of subarrays of length K.

N = 8 K = 3.

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

L	R
0	2
1	3
2	4
3	5
4	6
5	7

Think how to write this code?

10:25

Pseudocode

Print all L, R for all subarrays of size K

[19]

```
def print_all_k(A, k):
  N = len(A)
  L = 0

  for R in range(k-1, N): # R [k-1...N-1]
    print(L, R)
    L += 1

print_all_k([1, 2, 3, 4], 2)
```

```
0 1
1 2
2 3
```

Q) Given an integer array. Find max subarray sum of subarray of length = K

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

K = 4

L	R		total
0	3	[3 -2 4 -1]	4
1	4	[-2 4 -1 2]	3
2	5	[4 -1 2 6]	11

ans = 11

Bruteforce

For each subarray of length K
Find total and keep track of max total

```
def brute_force(A, k):
  N = len(A)
  L = 0
  max_total = float("-inf")

  for R in range(k-1, N): # R [k-1...N-1]
    sub = A[L:R+1]
    total = sum(sub)
    print(sub, total)
    max_total = max(max_total, total)
    L += 1

  return max_total

print(brute_force([3, -2, 4, -1, 2, 6], 4))
```

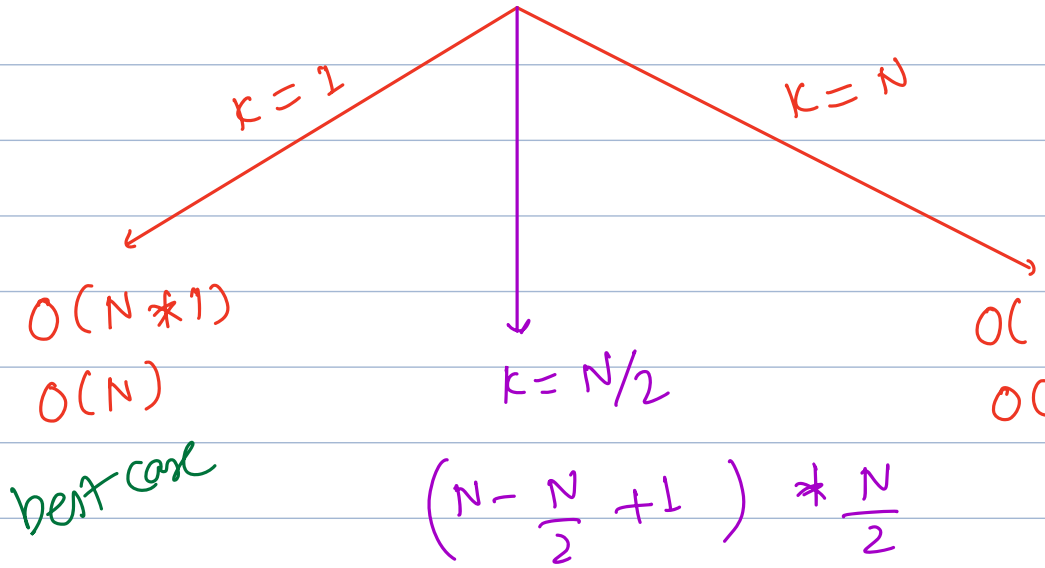
```
[3, -2, 4, -1] 4
[-2, 4, -1, 2] 3
[4, -1, 2, 6] 11
11
```

TC: # subs of length K * TC to cal sum

TC: O((N-K+1) * K)

TC: O(N^2)

. $O((N-k+1) * k)$



$$\left(N - \frac{N}{2} + 1\right) * \frac{N}{2}$$

$$\Rightarrow \left(\frac{N+2}{2}\right) * N/2$$

$$\Rightarrow O(N^2)$$

Prefix Sum

```python
def prefix_sum(A, k):
  N = len(A)
  L = 0
  max_total = float("-inf")

  for i in range(1, N):
    A[i] += A[i-1]

  for R in range(k-1, N): # R [k-1...N-1]
    if L == 0:
      total = A[R]
    else:
      total = A[R] - A[L-1]
    max_total = max(max_total, total)
    L += 1

  return max_total


print(prefix_sum([3, -2, 4, -1, 2, 6], 4))
```

```
11
```

TC: O(N)

SC: O(1)

Observation {sliding window}

$k = 4$

	0	1	2	3	4	5
A = [	3	-2	4	-1	2	6]

total = 4

	0	1	2	3	4	5
A = [	3	-2	4	-1	2	6]

total = 3

	0	1	2	3	4	5
A = [	3	-2	4	-1	2	6]

total = 3 + 6 − (−2)

= 9 + 2 = 11

Idea ⟶ Find total of first window

next total = total + new element − element out of window

[0 ..... k − 1] first window

Pseudocode

```
def max_subarray_sum_with_length_k(A, k):
  """
  Given an array of N elements. Print maximum subarray sum for s
  """
  N = len(A)
  L = 0

  total = 0
  for i in range(k): # 0....k-1
    total += A[i]

  max_total = total

  for R in range(k, N):
    total += A[R]
    total -= A[L] # L keeps track of elements out of window
    L += 1
    max_total = max(max_total, total)

  return max_total

print(max_subarray_sum_with_length_k([3, -2, 4, -1, 2, 6], 4))
```

O(k)

N - k

```
11
```

TC: O(N - k + k) = O(N)

SC: O(1)

Main Takeaway: This document outlines solutions to several common array-based problems, including counting specific character pairs, calculating subarray sums, and finding maximum subarray sums of a given length. It emphasizes optimizing solutions from brute force (often O(N^2) or O(N^3)) to more efficient approaches like prefix sums, contribution technique, and sliding windows (typically O(N)).

Count of Pairs 'ag'

Problem Definition

Given a string `s` of lowercase characters, count pairs `(i, j)` such that `i < j`, `s[i] == 'a'`, and `s[j] == 'g'`.

Examples

- `s = "bggsag"` -> 3 pairs
- `s = "aaga"` -> 3 pairs
- `s = "abcabcabgabc"` -> 3 pairs

Solutions

1. Brute Force

- **Time Complexity (TC):** O(N^2)
- **Space Complexity (SC):** O(1)

2. Optimized Solution

- **Idea:** Maintain a `count_of_a`. Whenever 'g' is encountered, increment the total pair count by the current `count_of_a`.
- **Pseudocode Logic:**

```
count_a = 0
total_pairs = 0
for char in s:
    if char == 'a':
        count_a += 1
    elif char == 'g':
        total_pairs += count_a
return total_pairs
```

- **Time Complexity (TC):** O(N)
- **Space Complexity (SC):** O(1)

Subarray Concepts

Definition

A subarray is a continuous part of an array.

Examples

- `A = [4, 1, 2, 3, −1, 6, 9, 8, 12]`
  - Subarrays: `[1, 2, 3]`, `[−1]`, `[8, 9]`, `[1, 6, 8]`, etc.

Representation of Subarrays

1. **Start and End Indices:** `A[start ... end]` (e.g., `start = 2, end = 4` for `A[2...4]`)
2. **Start Index and Length:** `A[start : start + length − 1]` (e.g., `start = 1, length = 2`)

Number of Subarrays in an Array of Size N

- A subarray can start at index `0` (N possible subarrays).
- A subarray can start at index 1 (N−1 possible subarrays).
- ...
- A subarray can start at index N−1 (1 possible subarray).
- **Formula:** The total number of subarrays for an array of size N is the sum of natural numbers up to N: `N * (N + 1) / 2`.

Sum of All Possible Subarray Sums

Problem Definition