Arrays: Carry Forward

- Count AG Pairs
- Carry Forward Technique
- Count leaders
- Subarray basics
- Inbuilt functions
- Closest Min Max

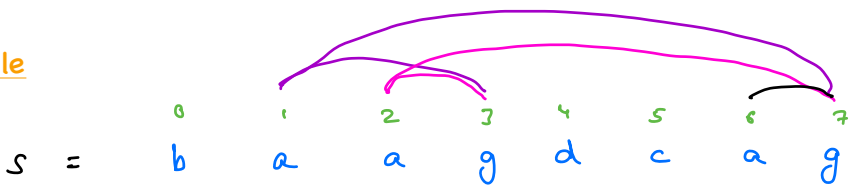
Q1 Count Pairs "ag"

Given a char[], calculate no of pairs (i, j) such that

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

All characters are lower case.

Example

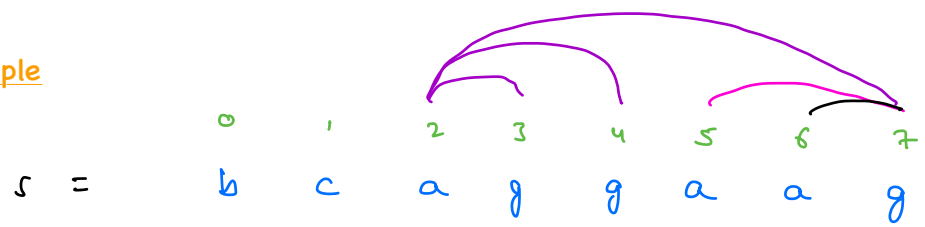


(1,3) , (1,7) , (2,3) (2,7) , (6,7)

Count = 5

Example

Quiz 1

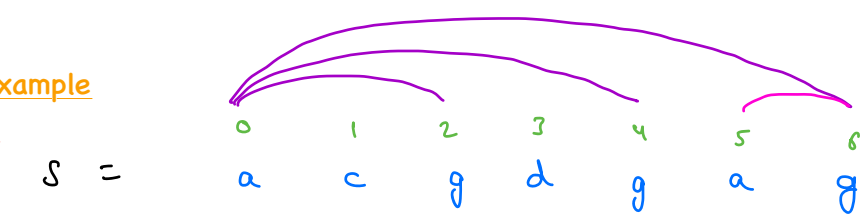


(2,3),(2,4) , (2,7) , (5,7) , (6,7)

Count = 5

Example

Quiz 2



(0,2) , (0,4) , (0,6) , (5,6)

Count = 4

Idea 1

Check for every pair $i < j$

```
countAgPairs(char []s) {
    n = s.length
    c = 0
    for (i=0; i<N; i++) {
        for (j=i+1; j<N; j++) {
            if (s[i] == 'a' and s[j] == 'g')
                c++
        }
    }
    return c;
}
```

Time - $O(N^2)$

Space - $O(1)$

Idea 2

aaggaa

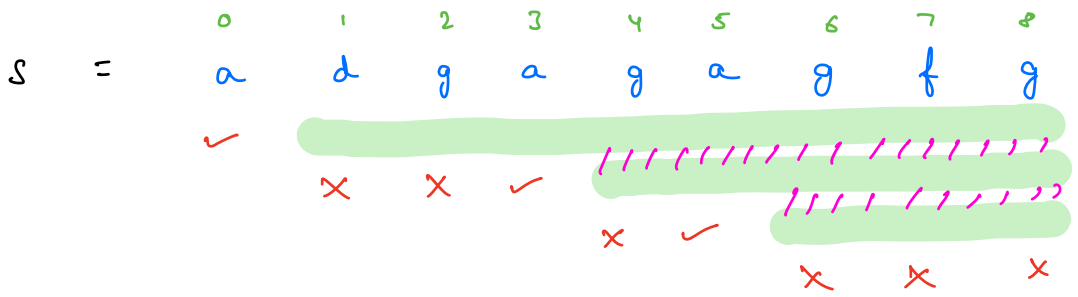
```
countAgPairs(char []s) {
    n = s.length
    c = 0
    for (i=0; i<n; i++) {
        if (s[i] == 'a') {
            for (j=i+1; j<n; j++) {
                if (s[j] == 'g')
                    c++
            }
        }
    }
    return c
}
```

For each a we search for g in its right

Time - $O(N^2)$

Space - $O(1)$

Example



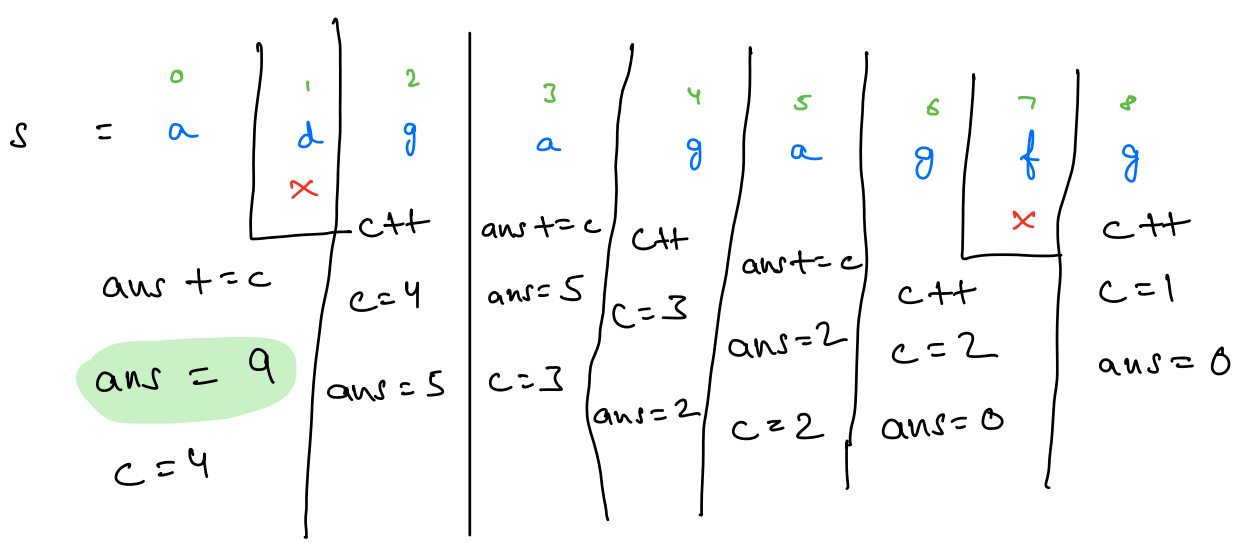
How to remove repetition?

Idea : Calculate no. of g's from right to left

Iterate right to left

Count of g → c = 0

ans = 0



This technique where we carry a certain value to be updated & used over the traversal is called

Carry Forward

Pseudocode

```
countAgPairs(char []s) {
        n = s.length
        c = 0          <-- count of g's
        ans = 0
        for (i = n-1 ; i >= 0 ; i--) {
            if (s[i] == 'g')
                c = c+1
            else if (s[i] == 'a')
                ans += c
        }
        return ans
}
```

Time - O(N)

Space - O(1)

Can we go left to right ? Any other idea ?

O(n) time

and

O(1) space

TODO

77 % - Yes

23 % - No

Java

```java
int solve(String A) {
    int n = A.length();
    int c = 0;
    int ans = 0;

    for (int i = n - 1; i >= 0; i--) {
        if (A.charAt(i) == 'g')
            c++;
        else if (A.charAt(i) == 'a')
            ans = ans + c;
    }

    return ans;
}
```

Python

```python
def countAgPairs(A):
    n = len(A)
    c = 0
    ans = 0

    for i in range(n - 1, -1, -1):
        if A[i] == "g":
            c += 1
        elif A[i] == 'a':
            ans = ans + c

    return ans
```

Q2 Leaders in an Array

Given an arr[N], you have to find all leaders in arr[]. An element is a leader, if is strictly greater than all elements on its right side or strictly greater than max on right.

Note: arr[N-1] is always considered a leader.
Last element

Example 1

arr[8] =	0	1	2	3	4	5	6	7
15	-1	7	2	5	4	2	3

x = 5
x > 2 → Yes
x > 5 → No
x >= 5 → Yes

Count = 5

Example 2

Quiz 3

arr[6] =	0	1	2	3	4	5
10	7	9	3	2	4

Count = 3

Example 3

arr[7] =

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

Count = 5

Example 4

arr[3] =

0	1	2
10	8	8

Count = 2

max ?

-5 -2 -9 -1 -6

Logic & Pseudocode

Simple Idea

→ Iterate & check for every index if it is a leader.

→ for each i, arr[i] > max on its right side, max[i+1 ... n-1]

```
countLeaders(int []arr){
    n = arr.length
    c = 1
    for (i=0; i<n-1; i++) {
        maxVal = arr[i+1]
        for (j=i+1; j<n; j++) {
            if (arr[j] > maxVal)
                maxVal = arr[j]
        }
        if (arr[i] > maxVal)
            c++
    }
    return c
}
```

i == n-1 → Error

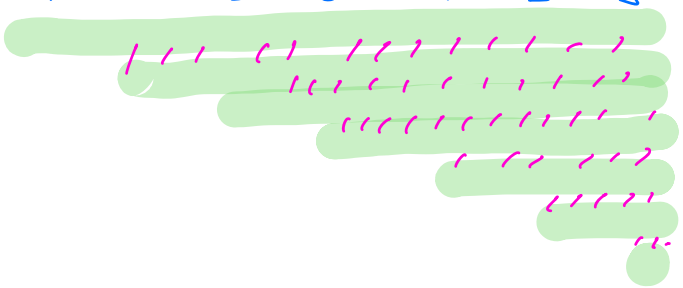
Time - O(N^2)

Space - O(1)

For each element, we are computing max element on its right.

Observations & Optimisations

	0	1	2	3	4	5	6	7
arr[8] =	15	-1	7	2	5	4	2	3



Iterate right to left

And carry forward the maxValue.

---

10 → 10 > 9 → c++ → c = 3 → maxVal = 10

8 → 8 > 9 → ✗

9 → c++ → c = 2 → maxVal = 9

2 → c = 1 → maxVal = 2

Right to left

9 > maxValue

9 > 2

```
countLeaders(int []arr){
        n = arr.length
        c = 1
        maxValue = arr[n-1]
        for( i = N-2; i >= 0; i--) {
            if ( arr[i] > maxValue ) {
                maxValue = arr[i]
                c++
            }
        }
        return c;
}
```

Time - O(N)

Space - O(1)

Java

```java
int solve(int[] A) {
    int n = A.length;
    int c = 1;
    int maxVal = A[n-1];

    for (int i = n - 2; i >= 0; i--) {
        if (A[i] > maxVal) {
            c++;
            maxVal = A[i];
        }
    }

    return c;
}
```

Python

```python
def countLeaders(A):
    n = len(A)
    c = 1
    maxVal = A[n - 1]

    for i in range(n - 2, -1, -1):
        if A[i] > maxVal:
            c += 1
            maxVal = A[i]

    return c
```

Can we carry left to right and solve this problem ?

Not possible

54% - Yes

46% - No

Meaningful reverse logic

cannot be constructed

Break till 10:31 PM

range(n-2, 0, -1) → [n-2, n-1... 1] >0

range(n-2, -1, -1) → [n-2, n-1... 1, 0] >-1

Subarray Basics

1. Continuous part of an array is called Subarray.
2. A single element is a Subarray.
3. Entire array is a Subarray.
4. Empty cannot be a Subarray.

	0	1	2	3	4	5	6	7	8
ar[9] =	-3	4	6	2	8	7	14	9	21

Ex 1: indices [2, 3, 4, 5] ✓

Ex 2: indices [3, 4, 6, 7, 8] ✗

Ex 3: indices [1, 2, 3] ✓

Ex 4: indices [5] ✓

Ex 5:

Subarray [3 7]: [3, 4, 5, 6, 7] ← indices

(3 = start, 7 = end)

Subarray [s e] ⇒ length = e - s + 1

Using predefined functions

1. `min(a, b)`

return min of two numbers

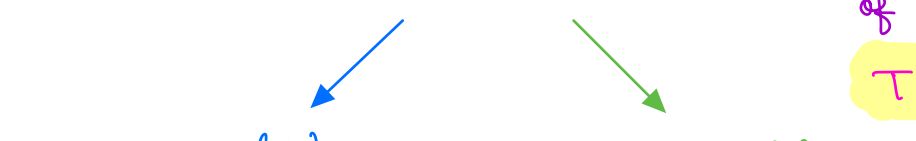
TC: O(1)

2. `max(a, b)`

returns max of two numbers

TC: O(1)

3. `sort(A)`



To sort an array of N items.

TC: $O(N \log_2 N)$

(Dual pivot Quicksort)

(Timsort)

Note: Before using any predefined function, read its TC

Q. Given N array elements, find the minimum of them

A = 5, 2, 9, 3, 1

```
A.sort()
x = A[0]
```

} TC: $O(N \log_2 N)$

```
minVal = A[0]
for (i=1; i<N; i++)
    minVal = min(minVal, A[i])
```

] $O(N)$ time

minVal = min(A) — O(n) time

---

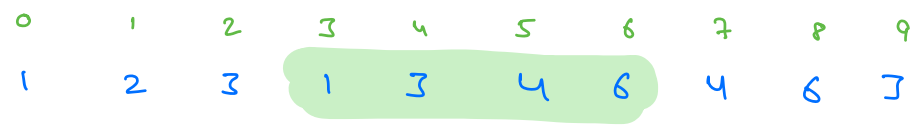
a **b = $O(\log_2 b)$

a^b

Q3 Closest Min Max

Given an array, find the length of smallest subarray which contains both min & max of the array

Ex 1



min = 1

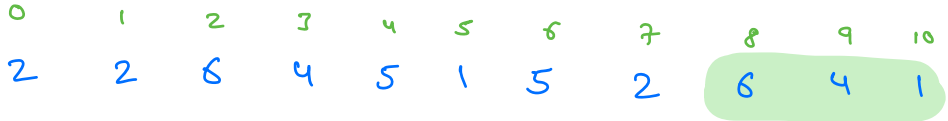
max = 6

[3 6]

length = 4

Quiz 4

Ex 2



min = 1

max = 6

subarray [8 10] ⇒ 3

Ex 3

0	1	2	3	4
8	8	8	8	8

min = 8

max = 8

Quiz 5

subarray [0 0]

subarray [1 1]

subarray [2 2]

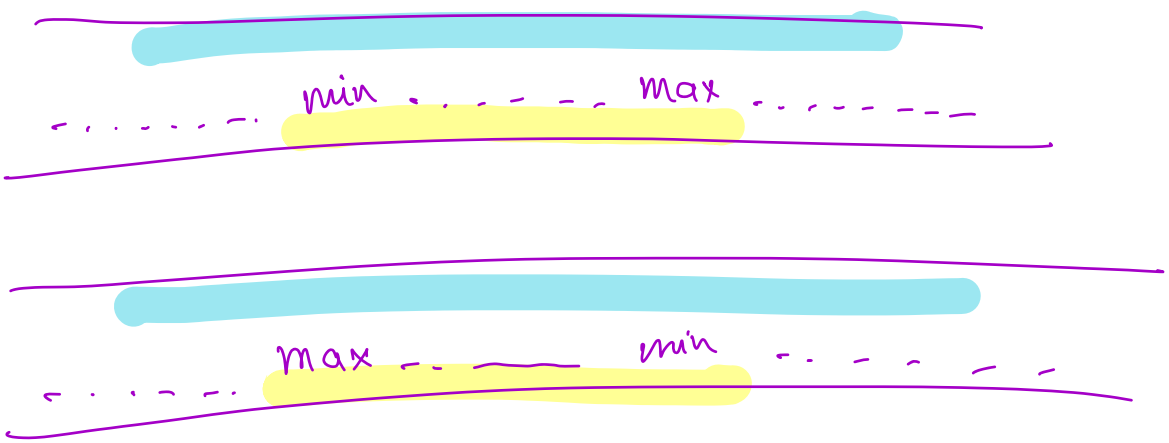
} length = 1

if (maxVal == minVal)
return 1

Observations

1) In final ans subarray.

minValue & maxValue should be at either corners.



2)

If you have a min value, search for the first max on right.

If you have a max value, search for the first min on right.

3)

Ex min=1 , max=6

0	1	2	3	4	5	6	7	8	9	10	11	12
2	2	6	4	5	1	5	6	2	4	3	4	1
x	x		x	x		x		x	x	x	x	

[2 5] length = 4

[5 7] length = 3

[7 12] length = 6

Only iterate if you encounter either a maxVal or minVal.

Brute Force - Pseudocode

```
closestMinMax(int []arr) {
    // Iterate & get the minV and maxV   <- TODO O(N)
    if (maxV == minV)
        return 1

    ans = N
    for(i=0; i<N; i++) {
        if(arr[i] == minV) {              <- Search for first max
            for(j=i+1; j<n; j++) {
                if (arr[j] == maxV) {
                    ans = min(ans, j-i+1)
                    break
                }
            }
        }

        else if (arr[i] == maxV) {        <- Search for first min
            for (j=i+1; j<n; j++) {
                if (arr[j] == minV) {
                    ans = min(ans, j-i)
                    break
                }
            }
        }
    }
    return ans
}
```

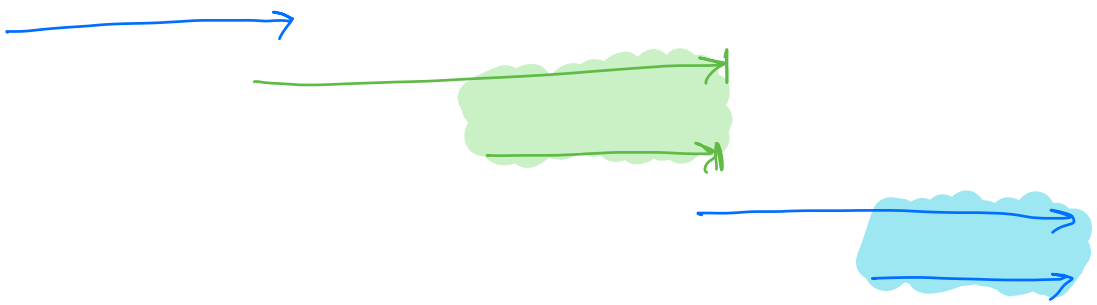
Time - $O(N^2)$

Space - $O(1)$

Optimisation

Ex

..... min .... max ..... max .... min .... min ... max



for each min — First index of max on right

for each max — first index of min on right

Iterate R to L & store minIndex & maxIndex ?

Ex

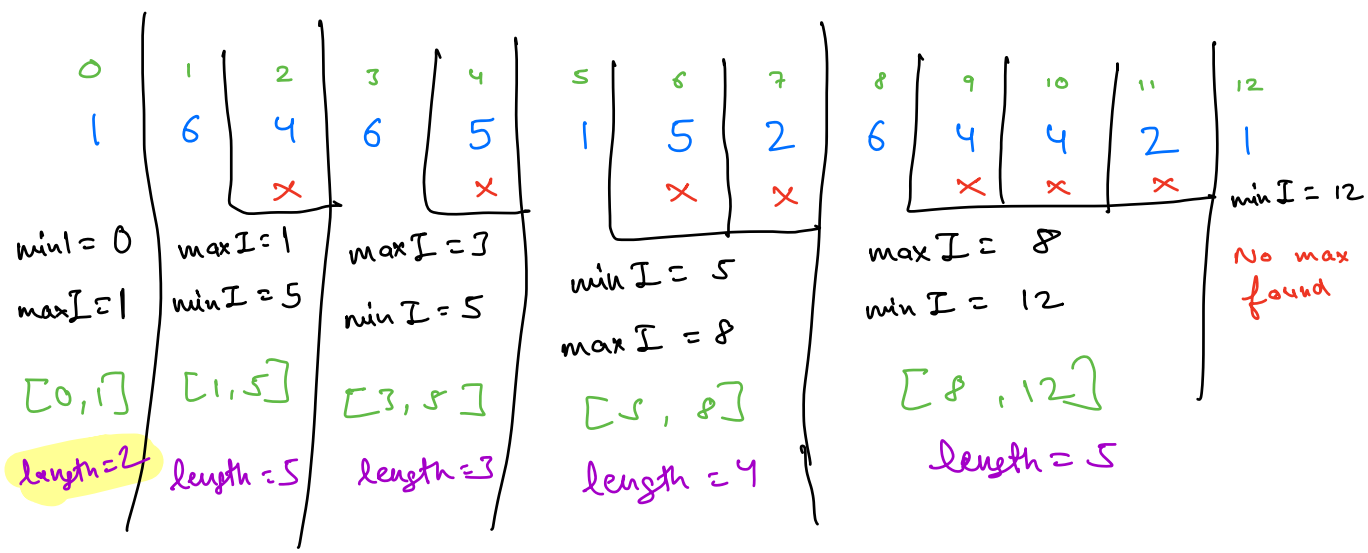
maxVal = 6

minVal = 1

ans = N

minIndex = -1

maxIndex = -1



Smallest length = 2

Pseudocode

1) Iterate & get maxV and minV

2) if maxV == minV

return 1

3) Initialise

minIndex = -1

maxIndex = -1

ans = N

4) Iterate R to L

```
for (i = N-1; i >= 0; i--) {
    if (a[i] == minV) {
        minI = i
        if (maxI != -1) {                    [minI, maxI]
            len = maxIndex - minIndex + 1
            ans = min(ans, len)
        }
    }
    else if (a[i] == maxV) {
        maxI = i
        if (minI != -1) {                    [maxI, minI]
            len = minI - maxI + 1
            ans = min(ans, len)
        }
    }
}

return ans
```

Time - O(N)

Space - O(1)

Java

```java
int closestMinMax(int[] A) {
    int n = A.length;
    int minVal = A[0], maxVal = A[0];
    for (int i = 1; i < n; i++) {
        minVal = Math.min(minVal, A[i]);
        maxVal = Math.max(maxVal, A[i]);
    }

    int minIndex = -1, maxIndex = -1;
    int ans = n;

    for (int i = n - 1; i >= 0; i--) {
        if (A[i] == minVal) {
            minIndex = i;
            if (maxIndex != -1) {
                int length = maxIndex - minIndex + 1;
                ans = Math.min(ans, length);
            }
        }
        if (A[i] == maxVal) {
            maxIndex = i;
            if (minIndex != -1) {
                int length = minIndex - maxIndex + 1;
                ans = Math.min(ans, length);
            }
        }
    }

    return ans;
}
```

Python

```python
def closestMinMax(A):
    n = len(A)
    minVal = A[0]
    maxVal = A[0]
    for i in range(1, n):
        minVal = min(minVal, A[i])
        maxVal = max(maxVal, A[i])

    minIndex = -1
    maxIndex = -1
    ans = n
    for i in range(n - 1, -1, -1):
        if A[i] == minVal:
            minIndex = i
            if maxIndex != -1:
                length = maxIndex - minIndex + 1
                ans = min(ans, length)

        if A[i] == maxVal:
            maxIndex = i
            if minIndex != -1:
                length = minIndex - maxIndex + 1
                ans = min(ans, length)
    return ans
```

Doubts

Thank You

Font : Jetbrains Mono

[a b] → b - a + 1

[3 5] → [3 4 5]

↓ 5 - 3 + 1 ↓ length = 3

[1 6] → [1, 2, 3 ... 6]

↓ 6 - 1 + 1 ↳ 6

Sliding Window → Extension of Carry Forward

Good Night

Thank You

Friday