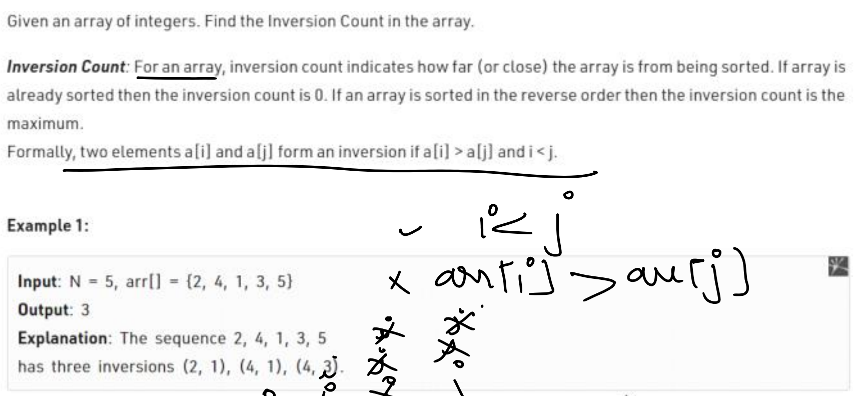
Day 8

16 November 2022 21:07

Count Inversion



Given an array of integers. Find the Inversion Count in the array.

Inversion Count: For an array, inversion count indicates how far (or close) the array is from being sorted. If array is already sorted then the inversion count is 0. If an array is sorted in the reverse order then the inversion count is the maximum.
Formally, two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j.

Example 1:

```
Input: N = 5, arr[] = {2, 4, 1, 3, 5}
Output: 3
Explanation: The sequence 2, 4, 1, 3, 5
has three inversions (2, 1), (4, 1), (4, 3).
```

✓ $i < j$

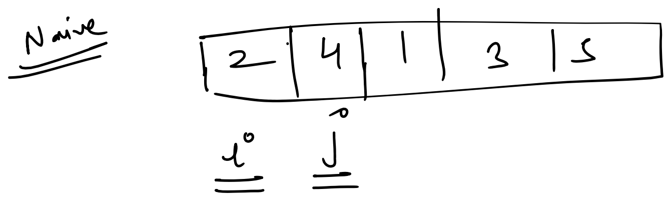
× $arr[i] > arr[j]$

$[2, 4, 1, 3, 5]$ $\quad 2 > 1$

count = ~~1~~ ~~2~~ 3

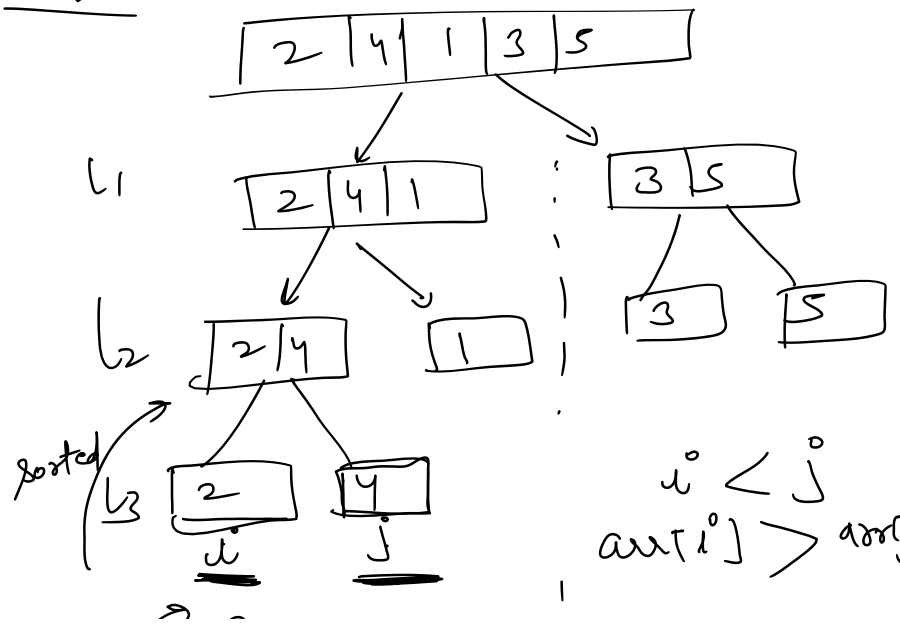
$4 > 1$

Naive



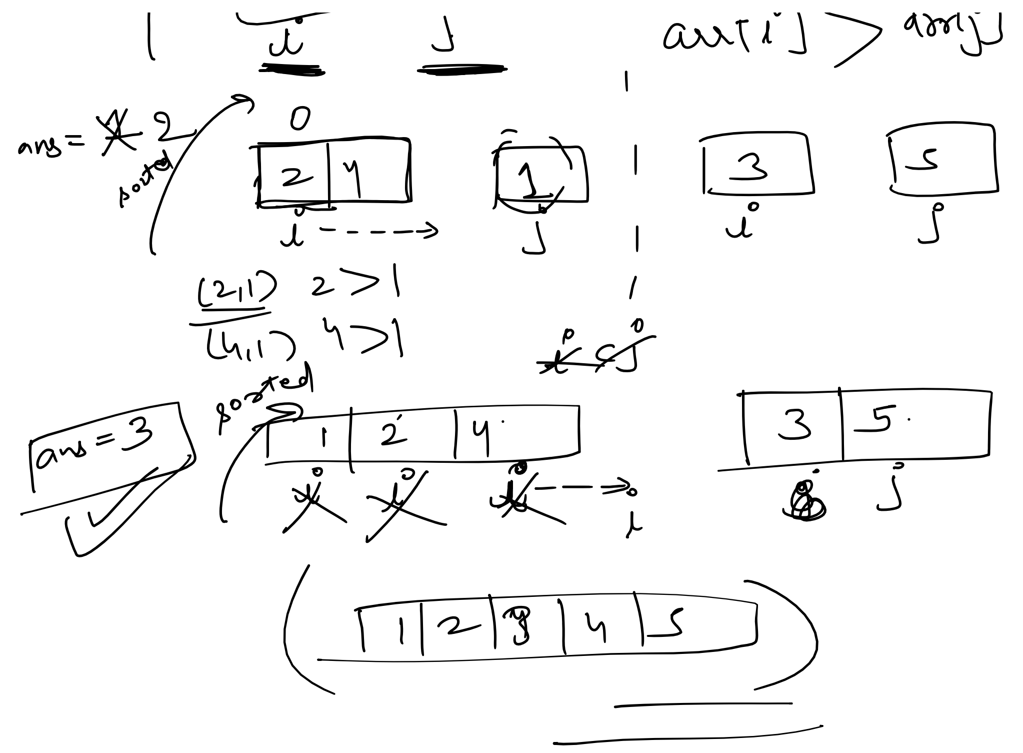
$T.C - O(N^2)$

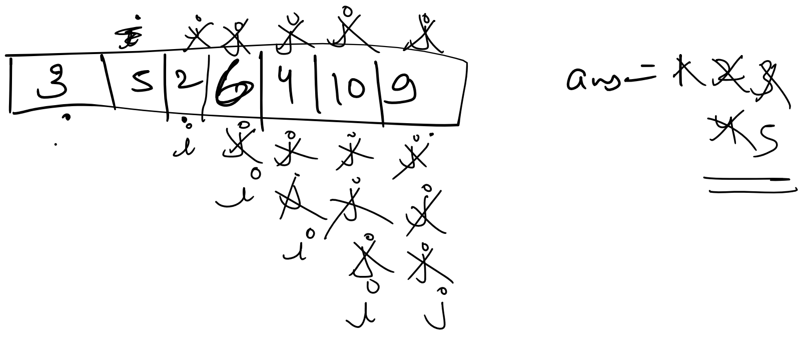
merge sort

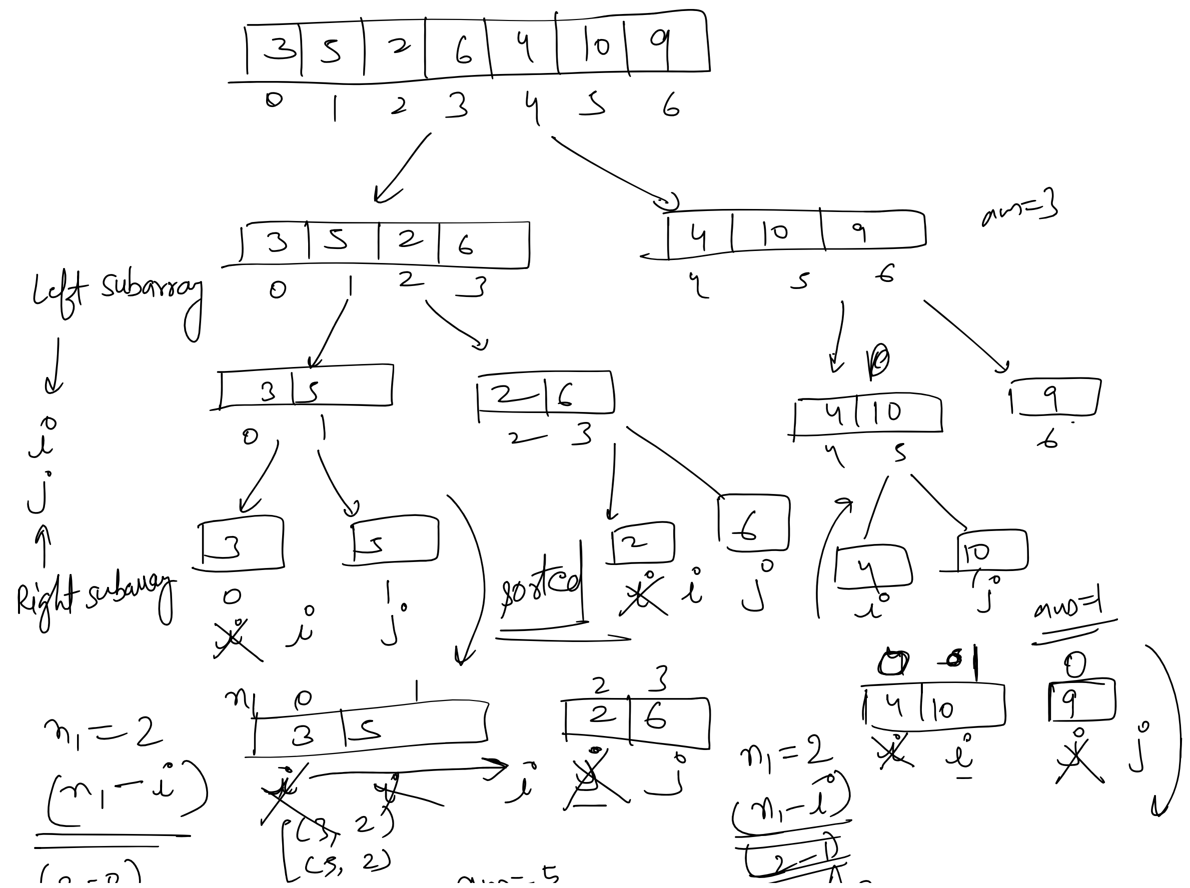


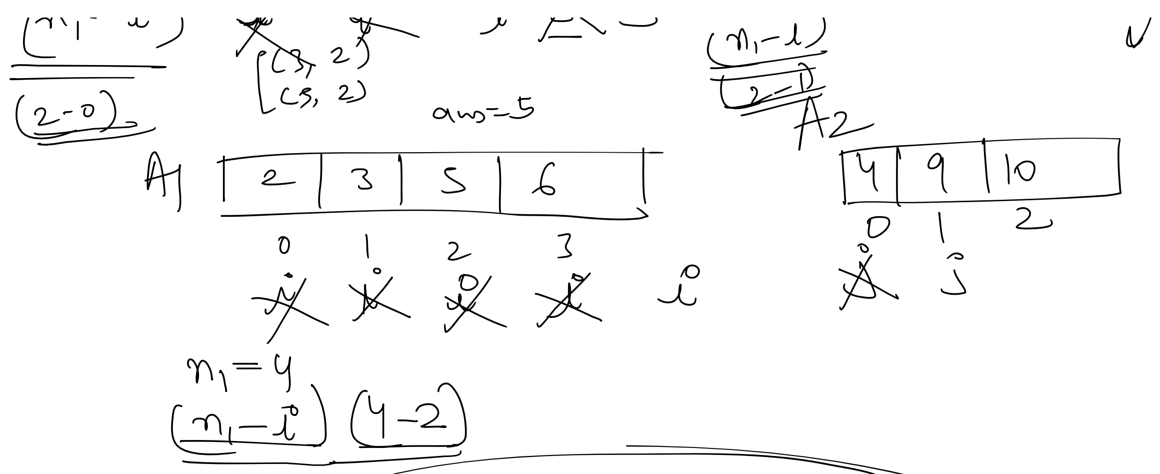
$i < j$

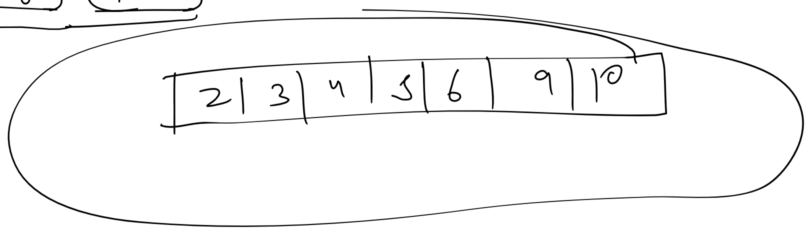
$arr[i] > arr[j]$

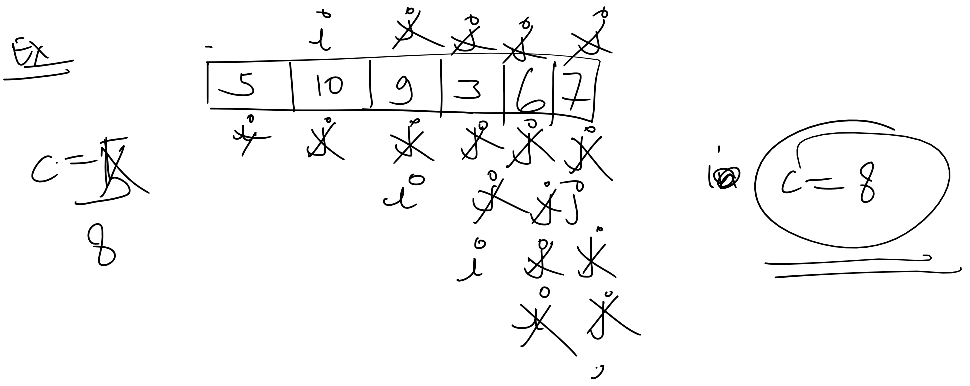

i j arr[i] > arr[j]
ans = ~~1~~ 2
sorted
(2,1) 2 > 1
(4,1) 4 > 1
ans = 3
sorted
1 2 3 4 5

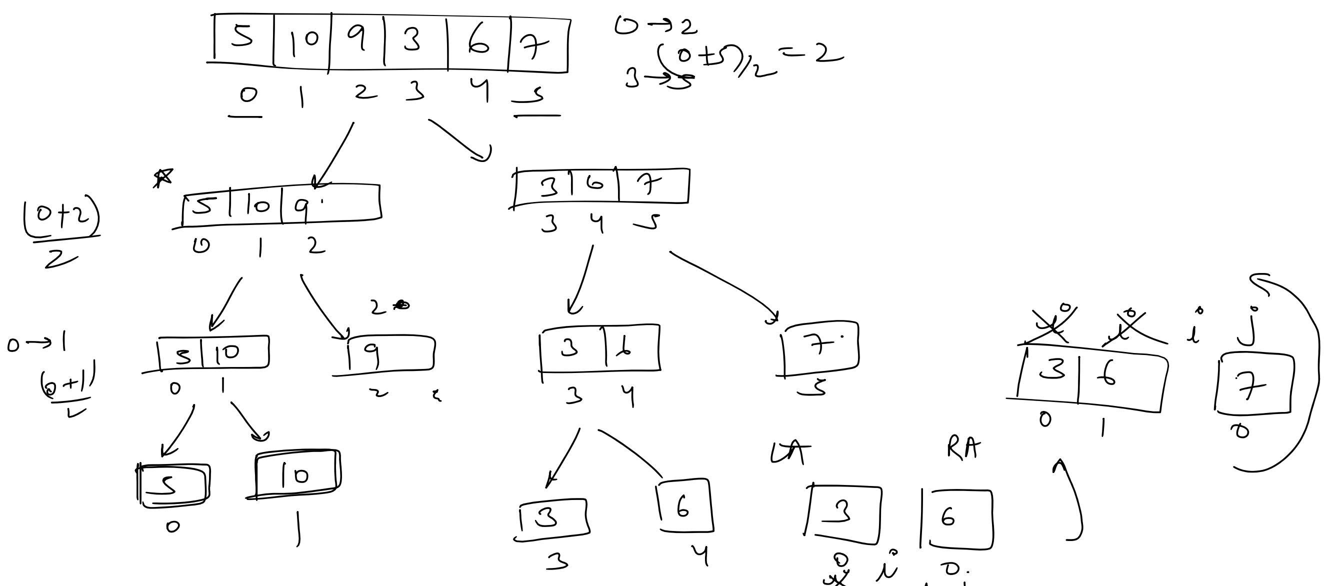

3 5 2 6 4 10 9
ans = ~~1~~ ~~2~~ ~~3~~
~~4~~ 5


3 5 2 6 4 10 9
0 1 2 3 4 5 6
ans=3
Left subarray
i
j
Right subarray
sorted
ans=1
$n_1 = 2$
$(n_1 - i)$
(3, 2)
(5, 2)
ans=5
$n_1 = 2$
$(n_1 - i)$
(2 − 1)





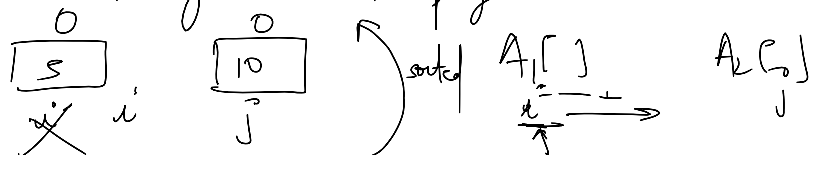


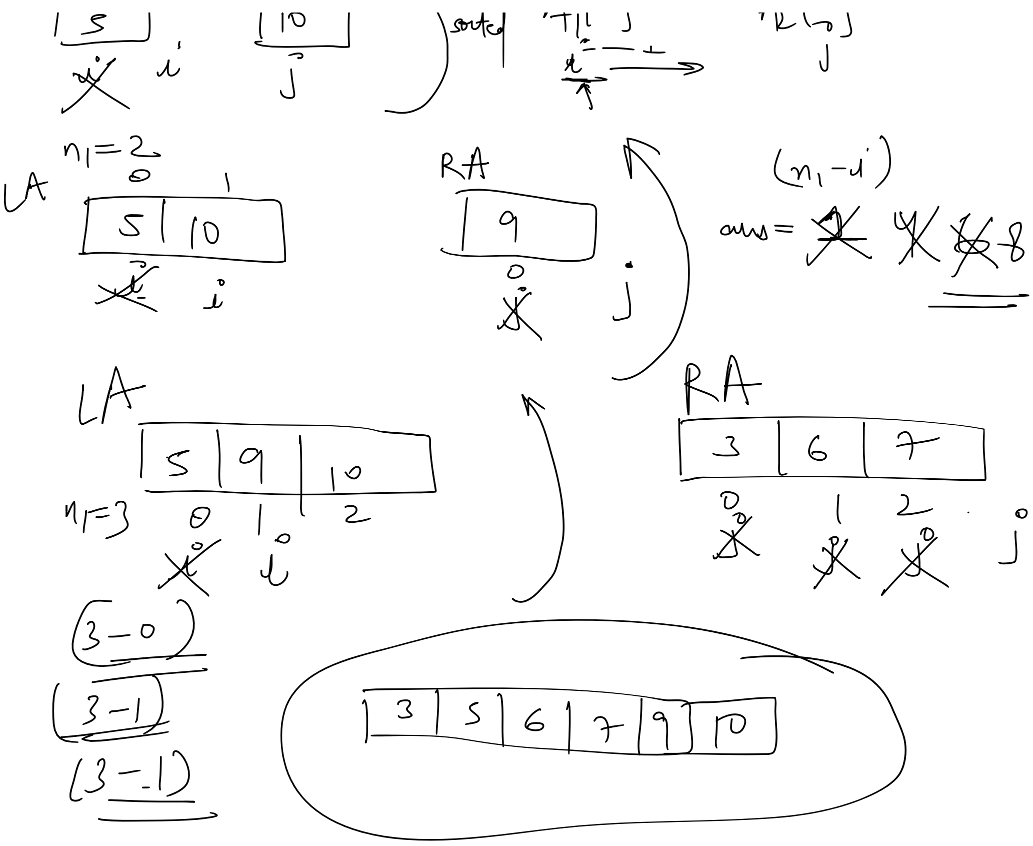


Merge two sorted Array   Division- Original

Left Sub Array ⟶ temporary

Right Sub Array ⟶ tempory





Merge Sort Implement

```
long long int inversionCount(long long arr[], long long N)
{
    long long int c = 0;
    merger_sort(arr,0,N-1,c);
    return c;
}
```

merge-sort(arr, 0, ending, c)

Divide wala Part

merge sort

```
void merger_sort(long long arr[],long long int start_index,long long int ending_index,long long int &count){
    if(start_index < ending_index){
        int middle_index =  start_index + (ending_index - start_index)/2;
        merger_sort(arr,start_index,middle_index,count);
        merger_sort(arr,middle_index+1,ending_index,count);
        merge(arr,start_index,middle_index,ending_index,count);
    }
}
```

// outbound

$\frac{(start\_index + end\_index)}{2}$

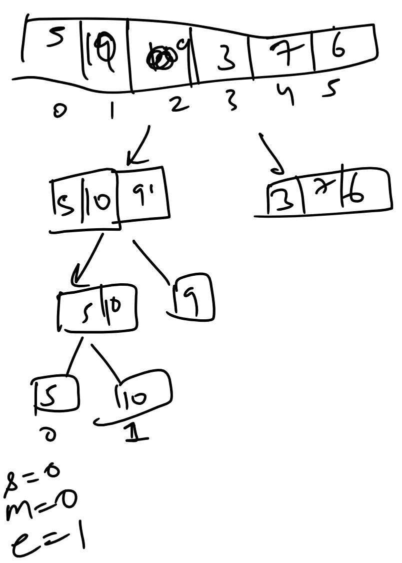
starting == ending

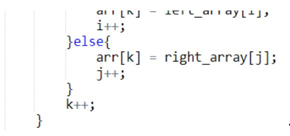
$[startindex + \frac{(e\_index - s\_index)}{2}$

Comparing and merging yala part

merge sort

```
void merge(long long arr[],long long int start_index,long long int middle_index,long long int ending_index,long long int &count){
    long long int n1 = middle_index - start_index + 1;
    long long int n2 = ending_index - middle_index;
    long long int left_array[n1];
    long long int right_array[n2];
    for(int i = 0;i < n1;i++){
        left_array[i] = arr[i+start_index];
    }
    for(int i = 0;i < n2;i++){
        right_array[i] = arr[middle_index +1+i];
    }

    int i = 0,j =0,k = start_index;
    while(i< n1 && j < n2){
            if(left_array[i] > right_array[j]){
                count += (n1 - i);
                ~~j++;~~
            }else{
                i++;
            }
    }

    // sorted merge
    i = 0,j = 0;
    while(i < n1 or j < n2){
        if(i < n1 and j < n2){
            if(left_array[i] < right_array[j]){
                arr[k] = left_array[i];
                i++;
            }else{
                arr[k] = right_array[j];
                j++;
            }
        }else if(i< n1){
            arr[k] = left_array[i];
            i++;
        }else{
            arr[k] = right_array[j];
            j++;
        }
        k++;
```





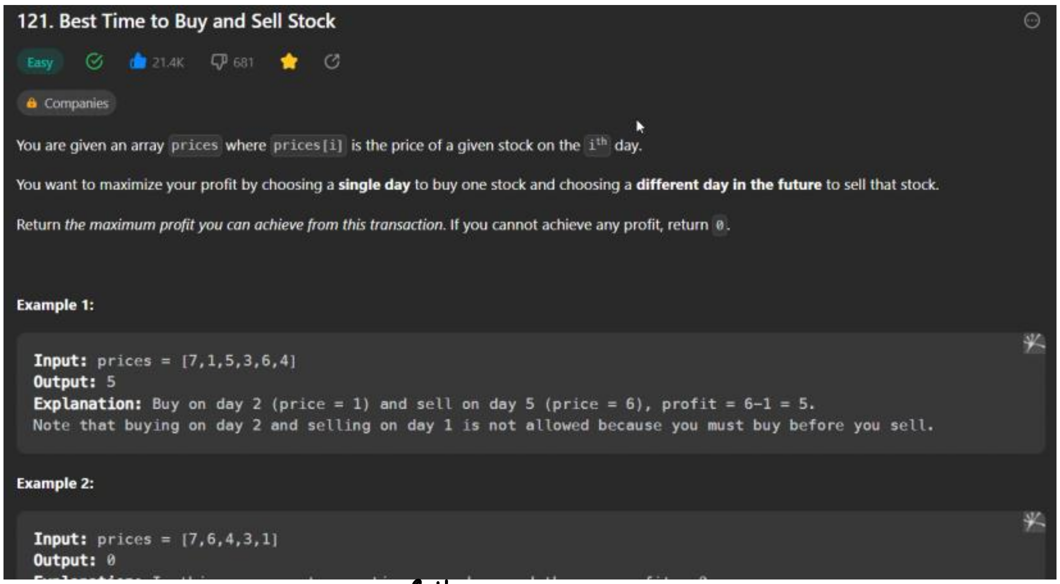
```
            arr[k] = left_array[i];
            i++;
        }else{
            arr[k] = right_array[j];
            j++;
        }
        k++;
    }
```

e=1

merge sort

```
//{ Driver Code Starts
#include <bits/stdc++.h>
using namespace std;


// } Driver Code Ends
class Solution{
  public:
    // arr[]: Input Array
    // N : Size of the Array arr[]
    // Function to count inversions in the array.
    void merge(long long arr[],long long int start_index,long long int middle_index,long long int 
ending_index,long long int &count){
        long long int n1 = middle_index - start_index + 1;
        long long int n2 = ending_index - middle_index;
        long long int left_array[n1];
        long long int right_array[n2];
        for(int i = 0;i < n1;i++){
            left_array[i] = arr[i+start_index];
        }
        for(int i = 0;i < n2;i++){
            right_array[i] = arr[middle_index +1+i];
        }

        int i = 0,j =0,k = start_index;
        while(i< n1 && j < n2){
                if(left_array[i] > right_array[j]){
                    count += (n1 - i);
                    j++;
                }else{
                    i++;
                }
        }



        // sorted merge
        i = 0,j = 0;
        while(i < n1 or j < n2){
            if(i < n1 and j < n2){
                if(left_array[i] < right_array[j]){
                    arr[k] = left_array[i];
                    i++;
                }else{
                    arr[k] = right_array[j];
                    j++;
                }
            }else if(i< n1){
                arr[k] = left_array[i];
                i++;
            }else{
                arr[k] = right_array[j];
                j++;
            }
            k++;
        }

    }
    void merger_sort(long long arr[],long long int start_index,long long int ending_index,long long int 
&count){
        if(start_index < ending_index){
            int middle_index =  start_index + (ending_index - start_index)/2;
            merger_sort(arr,start_index,middle_index,count);
            merger_sort(arr,middle_index+1,ending_index,count);
            merge(arr,start_index,middle_index,ending_index,count);
        }
    }
    long long int inversionCount(long long arr[], long long N)
    {
        long long int c = 0;
        merger_sort(arr,0,N-1,c);
        return c;
    }

};

//{ Driver Code Starts.

int main() {

    long long T;
    cin >> T;

    while(T--){
        long long N;
        cin >> N;

        long long A[N];
        for(long long i = 0;i<N;i++){
            cin >> A[i];
        }
        Solution obj;
        cout << obj.inversionCount(A,N) << endl;
    }

    return 0;
}

// } Driver Code Ends
```

Best Time to Buy and sell Stock.



121. Best Time to Buy and Sell Stock

Easy 21.4K 681

Companies

You are given an array `prices` where `prices[i]` is the price of a given stock on the `i`th day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

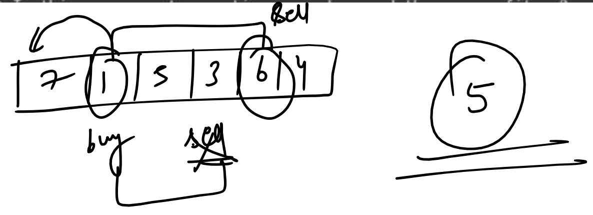
Return *the maximum profit you can achieve from this transaction*. If you cannot achieve any profit, return `0`.

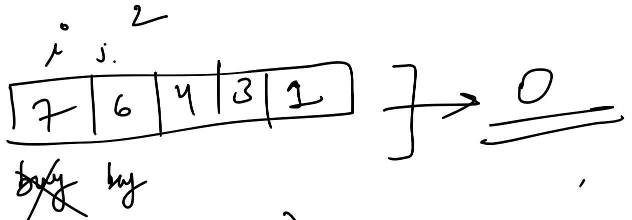
Example 1:

```
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.
```

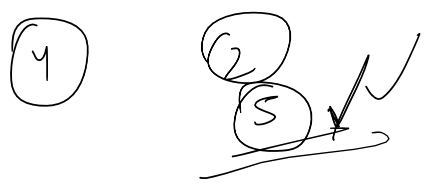
Example 2:

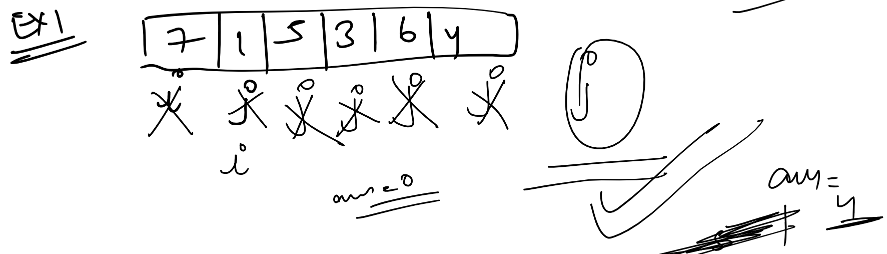
```
Input: prices = [7,6,4,3,1]
Output: 0
```

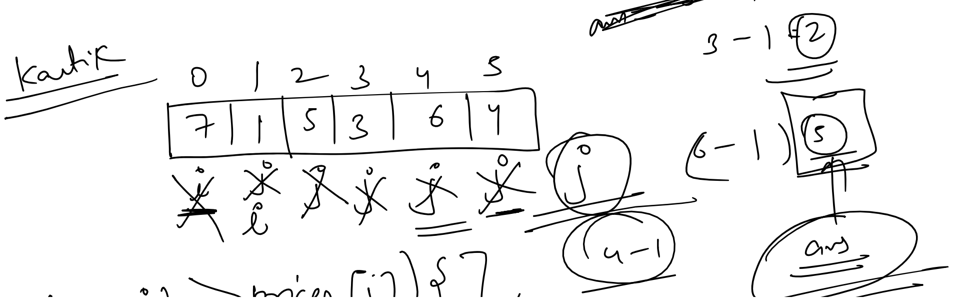




T. C − O(N²)







```
if (prices[i] > prices[j]) {
    i = j;
}
```

X $\frac{4-1}{3}$ ans

Generic

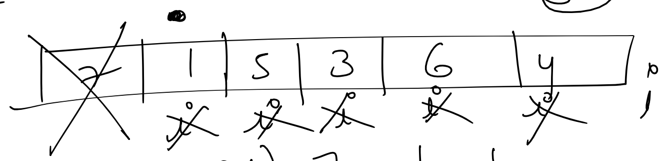
T.C — O(N)

P

buying = 7

2

3



buying = min(buying, arr[i]); 7 → b = 1

ans = max(ans, arr[i] - buying); ans = 0

1-1

~~5-1~~ ans = ~~4~~ 5

T.C - O(N)

S.C - O(1)

Two loops → Generic

Two pointer

Implement } 2 → Count, Best

2