ESO207 - Assignment 1

Mandar Bapat

September 20 , 2020

1 Pseudo Code

```
function InversionCount(Array A , left , right){
 if(left < right) then {
    mid = (left + right)/ 2
    x = InversionCount(Array A , left , mid)
    y = InversionCount(Array A , mid+1 , right)
    z = MergeStep(Array A , length , left , mid , right)
    return (x + y + z)
 }
 else{
    return 0
 }
}

function MergeStep(Array A , length , left , mid , right){
    p = left
    q = mid + 1
    r = left
    Array temp[length]
    count = 0
    while(r<=right){
        if(A[p] <= A[q]){
            temp[r] = A[p]
            r++
            p++
            continue
        }
        if(A[p] > A[q]){
            temp[r] = A[q]
            r++
            q++
            count += mid - p + 1
            continue
```

```
            }
            if(p>mid and q<=right){
                temp[r] = A[q]
                r++
                q++
                continue
            }
            if(p<=mid and q>right){
                temp[r]=A[p]
                r++
                p++
                continue
            }
        }
        for(p=left , p<=right ,p++){
            A[p]=temp[p]
        }
        return count
    }
```

2 Correctness of Algorithm

In the above pseudo code , we have 2 functions , a InversionCount function and a MergeStep function.The InversionCount function divides the array into 2 parts and calls the functions recursively on them. Finally the inversions from the MergeStep function are added.

We first prove the correctness of the MergeStep function by the **Loop invariant method** and then prove the correctness of the InversionCount function by **Strong Induction**.Loop invariant method is essentially a form of Induction.

2.1 Proving Correctness of MergeStep function using Loop Invariant method

2.1.1 Loop Invariants

```
1. temp[left,r-1] is sorted

2. temp[left,r-1] is a permutation of A[left,p-1] U  A[mid+1,q-1]

3. temp[left,r-1] <= A[q] , A[p]  , if p <= mid and q <= right
                     <= A[p]         , if p <= mid and q > right
                     <= A[q]         , if p > mid and q <= right

4. left <= p <= mid+1 and
   mid+1 <= q <= right+1 and
```

```
   r = p + q - mid -1

5. count = Number of inversions in the array A[left,q-1]
```

Invariants 1 to 4 have been proven in Lecture-4.Here we show the proof of property 5.

2.1.2 Invariance of property 5

```
Initializtion : Showing that the property 5 holds before the loop
                starts.
Proof : Before the loop starts ,
        p=left
        q=mid+1
        r=left
        count = 0
        and the temp array is empty
        Now , consider the array A[left,q-1] , i.e , A[left,mid].
        This array is sorted and hence number of inversions in
        A[left,mid] = 0.Also count is 0.
        Hence property 5 holds True before the start of the loop.

Maintainance : If property 5 holds after the t^th iteration ,
               then it holds after (t+1)^th iteration
Proof : Assume that after 't' iterations of the loop ,
        p=p_o
        q=q_o
        r=r_o
        count=c_o , where c_o denotes the number of
                        inversions in the array A[left,q_o - 1]
        The loop has 4 'if' blocks.Hence 4 cases arise:

    Case 1: If A[p_o] <= A[q_o] holds true , then
            after the execution of the loop ,
            temp[r_o] = A[p_o] and
            p = p_o + 1

            Since the array A[left,q-1] depends on q and
            q has not been changed , no new inversion pairs
            are added.
            The variable count is unchanged.
            Hence property 5 holds True.

    Case 2: If A[p_o] > A[q_o] holds true , then
            after the execution of the loop ,
            temp[r_o] = A[q_o] and
```

q = q_o + 1
count = c_o + $(mid - p_o + 1)$

Now , the array A[left,$q_o - 1$] *has changed to*
A[left,q_o] , *i.e* , q_o *is added at the end of*
the array A[left,$q_o - 1$]
Note that A[left,mid] is sorted.
Hence if A[p_o] $> A[q_o]$, *this implies*
A[$p_o + 1$] , $A[p_o + 2]$ .....$A[mid] > A[q_o]$
Hence, the new number of inversion pairs formed is
mid - p_o + 1.
This is exactly the amount by which
the count variable is updated.
Hence property 5 holds True.

Case 3: Consider the situation before the execution of
if(p > mid and q<= right)

By property 4 , left <= p <= mid + 1 .So if p > mid ,
only possibility is p = mid + 1
Before the execution ,
by property 2 we have , temp[left,$r_o - 1$] *is a*
permutation of A[left,$p_o - 1$] U $A[mid + 1, q_o - 1]$
and by property 3 , temp[left,$r_o - 1$] $<=$ $A[q_o]$.
Hence by above 3 statements ,
A[left , mid] <= A[q_o] *before the execution.*
After the execution , q = q_o + 1.*Since*
A[mid+1,right] is sorted , A[q_o] $<$ $A[q_o + 1]$
Hence A[left , mid] < A[$q_o + 1$] *after the execution.*
Hence by adding q_o *to the array* $A[left, q_o - 1]$
no new inversions were introduced.
Count was not updated.
Property 5 holds True.

Case 4: Consider the situation before the execution of
if(p <= mid and q > right)

By property 4 , mid+1 <= q <= right .So if q > right ,
only possibility is q = right + 1
Hence all numbers A[mid+1] to A[right] have been
compared.
There are no numbers left for comparison.
Hence no new inversions are found.
Count is not updated.
Property 5 holds True.

```
Termination : To show that property 5 holds after the loop ends
Proof : The loop ends when , r > right.By property 1 ,
        p = mid + 1 , q = right + 1
        Now count = Number of inversions in the array A[left,q-1]
        , i.e , A[left,right]
        This is the desired output.
        Hence the property 5 holds true.

Hence MergeStep function outputs the correct number of inversions.
```

2.1.3 Correctness of InversionCount function

We prove this by the Strong Induction rule. Define len(array A) = number of elements in array A

```
Base case : If len(A) = 1 , then there is only 1 element.No
            inversion pair possible.Hence number of inversions
            is 0 , which is what the function outputs
            If len(A) < 1 , the array is essentially empty.Hence
            here too , no inversion pairs are possible and the
            output should be 0 , which is what the function
            outputs.
Now suppose we have len(A) = n
We now assume that the function outputs correct number of inversions
for len(A) = 2 , 3 , 4 , .......(n-1)
Now the InversionCount function divides A into 2 parts :
A[left,mid] and A[mid+1,right] each of which have sizes < n.
Hence by our assumption , the recursive call outputs the correct
number of inversions for A[left,mid] and A[mid+1,right].
We already proved the correctness of MergeStep above.
Hence combining all 3 , we find that the function outputs the
correct number of inversion pairs for len(A) = n.

Hence by Strong Induction rule , the function outputs correct number
of inversions for all arrays A with len(A) = m ,
where m = non-negative integer
```

3 Time Complexity of the Algorithm

As can be seen , the pseudo code for the algorithm is very similar to the Merge sort algorithm except for few lines of code which take constant time.Statements like

```
count += mid - p + 1
```

take constant time and dont depend on input size directly.

This does not affect the time complexity analysis as for large size of inputs the constants are nevertheless dropped.

Hence time complexity = O(nlog(n)) for this algorithm too.