Name = Gurkirat Singh (IT)
Roll No: 11912048

Date / /
Page

100 on 100 STUDY NOTEBOOK

[Assignment (Sorting & B.S.T)]

Ques: Analysis on time Complexity of insertion sort in best case →

→ Insertion sort Algo. logic :-

```
→ for (x=1; x<n ; x++)
  { printf ("%d
    temp = a[x] ;
    for ( y = x-1 ; y >=0 && a[y] > temp; y--)
    {
      a[y+1] = a[y] ;
    }
    a[y+1] = temp;
  }
```

→ Now, in best case, All the data is in sorted order.

& let example of array

index →	0	1	2	3	4	5	6	7
	10	20	30	40	50	60	70	80

	x	1	2	3	4	5	6	7
	el	20	30	40	50	60	70	80
	y	0	1	2	3	4	5	6
	temp	20	30	40	50	60	70	80
{a[y] > temp}	Comparison	1	1	1	1	1	1	1
	Moves	1	1	1	1	1	1	1
		2(1)	2(1)	2(1)	2(1)	2(1)	2(1)	2(1)

Moves = 1 Only for outer for loop there is no movement in inner for loop for sorted array.

Now

Complexity

$2(1) + 2(1) + 2(1) + \dots$ (n-1) times

$= 2[1 + 1 + 1 + 1 + 1 \dots$ (n-1) times.$]$

$\rightarrow 2(n-1)$

So Complexity $= O(2(n-1))$

$\simeq O(n)$

→ Modification in Insertion sort to reduce its time complexity :-

→ * Not efficient Binary Insertion sort :- By using Binary Search we can reduce the no. of comparisons in insertion sort. By using Binary sort we can reduce time complexity of Comparison in Insertion sort to $O(n^2)$ to $O(\log n)$ in worst case.

So by using ~~insertion~~ binary search.

(movement)

Time complexity in shifting = $O(n^2)$

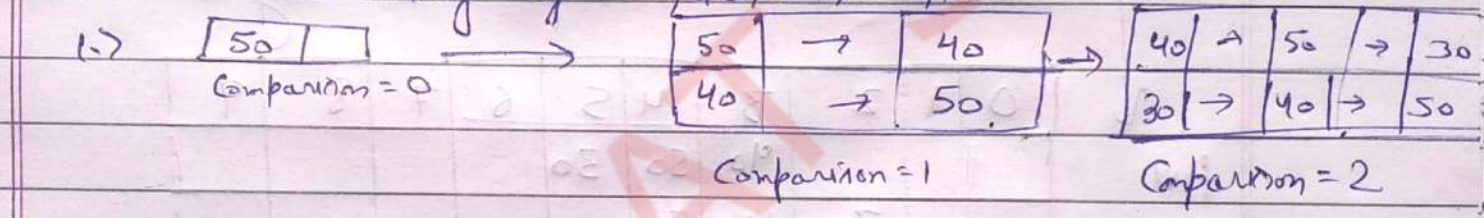
Time complexity in comparision = $O(n \log n)$

Total time complexity = $O(n^2)$

So still problem is not solved.

Now if we use linked list ~~then~~ instead of array then Time Complexity in shifting = $O(1)$ (As there New element is inserted at right without New shifting)

ex. while sorting of 50, 40, 30, 20, 10



So.

T.C. in Movement = $O(1)$

" " Comparison = $O(n^2)$

Total time Complexity = $O(n^2)$

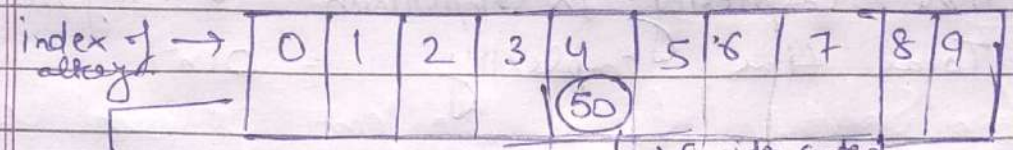
So it is not a efficient Method to reduce its time complexity.

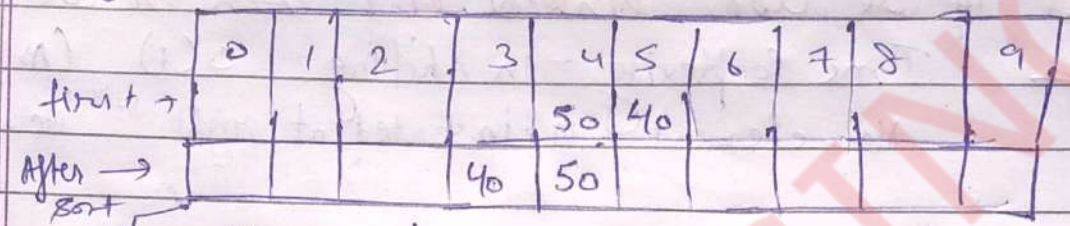
by increasing space Complexity :- As space complexity is less important than time complexity. We can use $2n$ space in array, where n is total no. of element. The insertion of element will start from $(n-1)^{th}$ position of the array. The same insertion sort is followed in this technique finding the suitable position of the element will be done by binary search.

So in worst Case

~~Total Movement = n-1~~

eg Sorting of 50, 40, 30, 20, 10





Moves = 1 Comparision = log 1

Moves = 1 Comparision = log 2

↓ And so on.

So

(Movement)

Total Shifting = n-1
≈ O(n)

Total Comparison = log 1 + log 2 + log 3 + ...
= log (1 * 2 * 3 * 4 ...)

= log (n-1)!, factorial.

≈ (n-1) log (n-1)
≈ O n log n

So total

O(n) + O(n log n) ≈ O(n log n)

→ (Modified Program in my github)