ASSIGNMENT-3

Tarusi Mittal
1901CS65
Tarusi Mittal

Ques 1:- Merging m sorted lists into a single list.

Ans:- The approach is we will begin with merging arrays into group of two. After the first merge, we will have $m/2$ arrays. Again we will merge arrays in groups, so we will have $m/4$ arrays and so on. we will merge the arrays in bottom-up manner

Algorithm

1. Creating a recursion function which will take m arrays as an input and will return the output array.
2. In the function, if at some point $m = 1$, the array will be returned
3. If value of $m = 2$, then we will merge the two arrays in linear time and return the array.
4. If value of $m > 2$, then we will divide the group of m sorted lists into two equal haves and we will then recursively call the function. (ie. $0 \rightarrow k/2$ array in one function
   $k/2 \rightarrow k$ array in another function)
5. Finally we will print the output array.

Finding Time Complexity.

In the above algorithm we divided the arrays into half at each step so total of it will be $\log m$ and at each level arrays traversed are m.

So, for the above algorithm,

Time Complexity is

$$\boxed{O(n \log m)}$$

Que2:- Running time of Quick Sort when all elements are equal.

Ans:- The running time of Quick Sort when all the elements are equal will be $\boxed{O(n^2)}$

Explanation

In the algorithm we pick the highest pivot but when all the elements are same no matter which pivot is picked, the algorithm have to go through all the values of the array.

The algorithm will cause n recursive calls to be made - each of which needs to make a comparison with the pivot and n- recursion elements ➜ $O(n^2)$ comparisons need to be made.

Also, this time is the worst time of quick sort.

Que3:- In the given condition :- Insertion sort over quick sort.

Ans:- We will take a sample array. ⇒ condition : elements are at most 3 positions away from sorted position.

$[7, 4, 3, 1, 9, 19, 12]$.

For insertion sort

the main code will be a loop which will run from
$i = 1$ to n (size)
and the inner loop will run from $j = 0$ to the given key index arr. ⇒

```
for (int i=0 ; i<n ; i++)
    key index = Arr[i]
    j = i-1
     while( j >= 0 88 Arr[j] > key index)
          → array will be updated
            8 j → j-1
```

Thus the inner loop will run at most n times - To move every element to the final sorted place total moves made will be k.

So the total time complexity will be:

$$\boxed{O(nk)}$$

In Quick Sort

In quick sort we will start from the leftmost element and keep track of index of smaller (or equal to) element as the current element.

So, in any case quick sort will run at least n times in each step and at each step as it will be halved. So total time complexity will be.

$$\boxed{O(n\log n)}$$

Now, according to the given problem K is 3 ie

$$k << n$$

$$\Rightarrow O(nk) < O(n\log n) \quad \left[\begin{matrix}\text{whenever} \\ k << n\end{matrix}\right]$$

$\Rightarrow$ Insertion sort is preferred over quick sort in this scenario.