Name :→ Harshita Mishra
University Roll no. :→ 2017498
Section :→ CST-SPL-1
Roll no. :→ 08

TUTORIAL- 3

Q1 → Write Linear Search pseudocode to search an element in a sorted array with minimum comparisons.

Ans →
```
void search( int arr[], int n , int x)
{
    if (arr (n-1) == x)
        Cout <<" Found" << endl;
    int t= a[n-1] ;
    arr [n-1]= x ;
    for (int i=0; i<=n; i++)
    {
        if (arr [i] = = x)
        {
            arr [n-1] = = t;
            if ( i<n-1)
            Cout << " Found" << endl;
        else
            cout << " Not Found" << endl;
        }
    }
}
```

Q2 → Write Pseudocode for iterative and recursive insertion sort. Insertion sort is called online sorting. why? What about other sorting algorithm that has been discussed in Lecture ?

Ans → For iterative →
```
void insertionSort ( int arr[], int n) {
    for (int i=1; i<n; i++)
        int t = a[i];
        int j = i;
        while ( j>0 && a[j-1]>t) {
```

```
a[j] = a[j-1];
   j--;
a[j]=t ; }
}
```

For Recursive :-

```
    void insertionSort (int a[], int n) {
        if (n<= 1)
          return;
      insertionSort ( arr , n-1);
     while( j>=0 && a[j]>temp) {
          a[j+1]= a[j];
          j--;
      }
       a[ j+1] = temp;
    }
```

Insertion Sort is also called online Sorting because it do not need to know anything about what values it will sort and th information is requested with the algorithm is running, Simply it can grab new values at every iteration.

Q3- Complexity of all sorting algorithm that has been discussed in Lectures:

Ans-

Sorting Type	Worst Case Time / Space	Average Case	Best Case
Bubble Sort	$O(n)^2/O(1)$	$O(n^2)$	$O(n)$
Insertion Sort	$O(n^2)/O(1)$	$O(n^2)$	$O(n)$
Selection Sort	$O(n^2)/O(1)$	$O(n^2)$	$O(n^2)$
Quick Sort	$O(n^2)/O(\log n)$	$O(n\log n)$	$O(n\log n)$
Merge Sort	$O(n\log n)/O(n)$	$O(n\log n)$	$O(n\log n)$
Count Sort	$O(k)/O(k)$	$O(n+k)$	$O(n)$
Randomised Quick	$O(n^2)/O(\log n)$	$O(n\log n)$	$O(n\log n)$
Heap Sort	$O(n\log n)/O(n)$	$O(n\log n)$	$O(n\log n)$

Q4→ Divide all the Sorting algorithms into inplace/ stable / online.

Ans→

Sorting Type	Inplace	Stable	Online
Bubble Sort	✓	✓	✗
Insertion Sort	✓	✗	✓
Selection Sort	✓	✓	✗
Quick Sort	✓	✗	✗
Merge Sort	✗	✓	✗
Count Sort	✗	✓	✗
Randomised Quick	✓	✗	✗
Heap Sort	✓	✗	✗

Q5→ Write Iterative / Recursive Pseudocode for binary Search. What is the time and space Complexity of Linear and Binary Search?

Ans→ Recursive:→

```
int binarySearch( int arr[], intl, intr, int x)
{
    if( r>=l)
        m= (l+r)/2 ;
    if (arr [mid] = x)
        return mid ;
    else if ( arr[mid] >x)
        return Binary Search (arr,l, mid-1 ,r,x) ;
    else
        return BinarySearch (arr, mid+1, r,x);
    return -1 ;
}
```

Iterative :→

```
int binarySearch (int arr[], int l, int r, int x) {
    while ( l <= r) {
        int m= (l+r)/2 ;
        if (arr [m] = x);
            return m;
        else if (arr[m] < x)
            l= m+1;
        else
            r =m-1;
    }
    return-1 ; }
```

Time Complexity		Space Complexity		Search Algo.
recursive	iterative	recursive	iterative	
O(n)	O(n)	O(n)	O(1)	Linear Search
O(logn)	O(logn)	O(logn)	O(1)	Binary Search

Q6→ Write recurrence relation ~~between~~ for binary search recursive?

Ans:→ Recurrence Relation ⇒ $T(n) = T(n/2) + 1$

where T(n) is the required time for binary search in an array of size n.

Q7→ Find two indexes such that $A[i] + A[j] = k$ in minimum time Complexity?

Ans:→
```
int find (A[], n, k) {
    Sort (A, n);
    for (i = 0 to n-1) {
        x = binary Search (A, 0, n-1, K-A[i]);
        if (n)
            return 1;
    }
    return -1 }
}
```
Time Complexity = $\boxed{O(n \log n)}$

Q8→ Which Sorting is best for practical use? explain.

Ans:→ Quick Sort is the fastest general purpose sort. In most of practical situations, Quick Sort is a method of choice. If Stability is important and space is available, merge Sort might be best.

Q9:→ What do you mean by no. of inversion in an array. Count no. of Inversions in Array arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5} using merge Sort?

Ans:→ Inversion in an array indicates how ~~the~~ far the array is ~~array~~ being sorted. If the array is already sorted then its inversion count is 0, but when it is sorted in reverse order the inversion count is maximum.

arr = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5} → has 31 inversions.

Q10 → In which Cases Quick Sort will give the best and worst case time Complexity ?

Ans :→ The worst case time Complexity of Quick Sort is $O(n^2)$. The worst case occur when the picked pivot is always extreme (Smallest or Largest) element. This happens when i/p array is sorted or reverse sorted and either first or Last element is picked as pivot.
The best case Time Complexity of Quick Sort is $O(n \log n)$. It occurs when we select pivot as a mean element.

Q11 → With recurrance relation of merge Sort & Quick Sort in best & worst case? What are the Similarities and difference b/w complexities of two algorithms and why ?

Ans :→ Recurrence relation of Merge Sort → $T(n) = 2T(n/2)+n$
Quick Sort → $T(n) = 2T(n/2)+n$

Merge Sort is more efficient and works faster than Quick Sort in case of larger array size or datasets.
Worst Case Complexity for Quick Sort is $O(n^2)$ whereas for merge Sort is $O(n \log n)$.

Q12 → Selection Sort is not Stable by default but you can write a version of Selection Sort.

Ans :→
```
void Selection Sort (int arr[], int n) {
        for (int i=0; i< n-1; i++) {
            int num= 10;
            for (int j = i+1; j<n; j++) {
                if (arr[num] > arr[j])
                    min = j; }
            int key = arr[min];
            while (min>1) {
                    arr [min] = arr[min-1];
                    min --;
            }
            arr [i] = key;
    } }
```

Q13 → Bubble Sort can Scans whole array even when array is Sorted. Can you modify the bubble Sort so that it doesn't Scan the whole array once it is Sorted.

Ans :→
```
void Bubble(int arr[], intn) {
        four( int i=0; i<n; i++) {
            int swap=0;
          for(int j=0; j<n-1; j++) {
            if (a[j] > a[j+1]) {
               int t = a[j];
               a[j]=a[j+1];
               a[j+1] =t;
               swap ++;
        } }
        if (swap==0)
            break; }
        }
```

Q14 :→ Your Computer has a RAM (Physical Memory) of 2GB and you are given an array of 4 GB for Sorting. which algorithm you are going to use for this purpose & why? Also explain the concept of External & Intornal Sorting.

Ans :→ we use external Sorting for this purpose.
If 1 GB = 1024 B

Divide source file into 2 small temp files of size 2GB each (RAM)

$$\begin{tikzcd}
 & \text{4GB Random} \arrow[dl] \arrow[dr] & \\
\text{2GB Random} \arrow[d] & & \text{2GB Random} \arrow[d] \\
\text{2GB} \arrow[d] & \text{RAM} \arrow[dr] & \\
\text{2GB Sorted} & & \text{2GB sorted}
\end{tikzcd}$$

→ pointers are initialised in each file.
→ A new file of each size 4GB is created.
→ Ist element is Compared from each file with pointer.
→ Smallest element is copied into new 1GB file and pointer gets incremented in file which pointed to smallest element.

→ Same process is followed for all pointers have transversed their respective file.

→ when all pointer have transversed we have a new file having

1GB of sorted Integer.

This is how the larger file can be sorted when there is a limitations on the size of RAM.

Internal Sorting :-

If the input data is such that it can be adjusted in the main memory at once it is called Internal Sorting.

External Sorting :- If the Input data is such that it can not be adjusted in main memory entirely at once it needs to be sorted in a hard disk/ floppy disk / or any Storage devices.