Name - Akanksha Dubey
Section - C
ROLL NO - 2016600(70)

Tutorial-3

1. Write linear search Pseudocode to search an element in a sorted array with minimum comparision.

```
for (i = 0 to n)
 { if (arr[i] == value)
      comp++;
 }
```

2. Write Pseudo Code for iterative & recursive insertion sort. Insertion sort is called Online sorting. Why? What about other sorting algorithms that has been discussed

Iterative

```
void insertion_sort (int curr[], int n)
{ for (int i = 1; i<n ; i++)
   { j = i-1;
     x = curr[i];
   while (j>=1 && curr[j]>x)
    { curr[j+1] = curr[j];
        j--;
    }
   curr[j+1] = x;
  }
}
```

Recursion

```
void insertion_sort(int arr[], int n)
 { if (n<=1) return;
   insertion_sort(arr, n-1);
   int last = arr[n-1];
    int j = n-2;
 while(j>=0 && arr[j]>last) { arr[j+1] = arr[j]; j--; }
   arr[j+1] = last;
}
```

Insertion sort is called 'online sort' because it does not need to know anything about what value it will sort and information is requested while algorithm is running.

Other sorting Algorithm

1. Bubble sort
2. Quick sort
3. Merge sort
4. selection sort
5. Heap sort

3. Complexity of all sorting algorithm

Sorting Algorithm	Best	Worst	Average
Bubble sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$
Heap sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick sort	$O(n \log n)$	$O(n^2)$	$O(n \log n)$
Merge sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

4. Divide all sorting algorithms into inplace/stable/online sorting

Inplace sorting	stable sorting	online sorting
Bubble sort	Merge sort	Insertion sort
Selection sort	Bubble sort	
Insertion sort	Insertion sort	
Quick sort	Count sort	
Heap sort		

5. Iterative Pseudocode for binary search.

```
int bsearch(int arr[], int l, int h, int key)
{ while(l<=h)
  { int m = (l+h)/2;
    if(arr[m] == key)
      return m;
    else if(key > arr[m])
      l = m+1;
    else
      h = m-1;
  }
  return -1
}
```

Recursive

```
int bsearch(int arr[], int l, int h, int key)
{ while(l<=h)
  { int m = (l+h)/2;

    if(arr[m] == key)
      return m;
    else if(arr[m] < key)
      return bsearch(arr, m+1, h, key);
    else
      return bsearch(arr, l, m-1, key);
  } return -1;
}
```

Time complexity = O(n) → Linear search

B. Tc of binary search → O(logn)

6. Write Recurrence relation for Binary recursive search

$T(n) = T(n/2) + 1$ — (1)

$T(n/2) = T(n/4) + 1$ — (2)

$T(n/4) = T(n/8) + 1$ — (3)

$T(n) = T(n/2) + 1$

$= T(n/4) + 1 + 1$

$= T(n/8) + 1 + 1 + 1$

$T(n/2^n) + 1$ (k times)

let $2^k = n$

$k = \log n$

$T(n) = T(n/n) + \log n$

$T(n) = T(1) + \log n$

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

7. find two index such that $A[i] + A[j] = K$ in min T.C

```
for (int i = 0; i<n; i++)
  { for (int j = 0; j<n; j++)
    { if (a[i] + a[j] == k)
        cout << i << j;
    }
  }
```

8. Which sorting is best for practical uses? Explain

Quick sort is fastest general-purpose sort. In most practical situations quicksort is the method of choice. If stability is important and space is available, mergesort might be best.

SEE OTHER

9. What do you mean by ~~inversions~~ inversion in an array? Count the no. of ~~inversions~~ inversion in Array arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5} using merge sort.

A Pair (A[i], A[j]) is said to be ~~inv~~ inversion if
- A[i] > A[j]
- i < j

- Total no. of inversions in given array are 31 using merge sort.

10. In which cases Quick sort will give best & worst case time complexity.

* Worst case :- $O(n^2)$ - The worst case occcurs when the pivot element is an extreme (smallest / largest) element. This happens when input array is sorted or reverse sorted and either first or last element is selected as pivot.

* Best case $O(n \log n)$:- The best case occurs when we will select pivot element as a mean element.

11. Recurrence Relation for Merge Sort / Quick sort in best & worst case. What are similarities & difference b/w complexities of two algorithm & why?

* Merge Sort →

Best case :- $T(n) = 2T(n/2) + O(n)$ } $O(n \log n)$
Worst case - $T(n) = 2T(n/2) + O(n)$ }

* Quick Sort

Best case — $T(n) = 2T(n/2) + O(n) \rightarrow O(n \log n)$
Worst case - $T(n) = T(n-1) + O(n) \rightarrow O(n^2)$