Tutorial - 3

Name - Harshita Pratap

Section - G

Rollno - 36

Subject - Design and Analysis of Algorithms

Q1. Write a linear search pseudocode to search an element in a sorted array with minimum comparisons

Ans.
```
for (i = 0 to n)
{
    if (arr[i] == value)
        // element from d
}
```

Q2. Write Pseudocode for iterative and recursive insertion sort. Insertion Sort is called online sorting. Why? What about other sorting algorithms that has been discussed.

Ans. Iterative
```
void insertion - Sort (int arr [], int n)
{
    for (i = 1; i < n ; i++)
    {
        j = i - 1;
        x = arr[i];
        while (j > -1 && arr[j] > x)
        {
            arr [j+1] = arr[j];
            j--;
        }
        arr[j+1] = x;
    }
}
```

Recursive

```
void insertion_soot (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 values it will sort and information is requested while algorithm is running.

Other sorting Algorithm =>

1) ~~[illegible]~~ Bubble sort
2) Quick sort
3) Merge sort
4) selection sort
5) Heap sort

Q3. complexity of all sorting algorithms that has been discussed in lectures.

Sorting Algorithms	Best	Worst	Average
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$
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	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Q4 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		

Q5. write recursive/iterative pseudocode for binary search. what is time and space complexity of linear & Binary search.

Iterative

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

Recursive

```
int b-search (int arr[] , int l, int r, int key)
{
        while (l <= r) {
        int m = ((l+r)/2);

        if (key == arr[m])
            return m;

        else if (key < arr[m])
            return b-search (arr, l, mid-1, key);

        else
            return b-search (arr, mid+1, r, key);

      }
     return -1;
}
```

Time complexity

1. Linear Search O(n)

2. Binary search O(logn)

Q. 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

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

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

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

$\vdots$

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

let $2^k = n$

$k = \log n$

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

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

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

Q7. Find two index such that A[i] + A[j] = k in minimum time complexity.

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

Q8. Which sorting is best for practical uses. In most Quicksort is fastest general purpose sort. In most practical situations quicksort is the method of choice as stability is important and space is available. merge sort might be best.

Q9. What do you mean by inversions in an array. What is the number of inversions in Array arr[] = count the number of inversions
{7, 21, 31, 8, 10, 1, 20, 6, 4, 5}
A Pair (A[i], A[j]) is said to be inversion of

- A[i] > A[j]
- i < j
- Total no. of Inversions in given array are 31 using merge sort

Q10. In which case Quick sort will give best and worst case time complexity.

Worst case $O(n^2)$

The worst case occurs 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.

Q11. Write Recurrence Relation of Merge / Quick sort in best of worst case. What are the similarities of differences between complexities of two algorithm and why?

Ans. 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 $T(n) = T(n-1) + O(n)$

In quick sort, array of element is divided into 2 parts repeatedly until it is not possible to divide it further.

Q 12. Selection sort is not stable by default but can you write a version of stable selection sort.

```
for(int i=0 ; i<n-1 ; i++)
{
    int min = i;
   for(int j= i+1 ; j<n ; j++)
   {
      if(a[min] > a[j])
         min = j ;
   }
   int key = a[min];
   while (min>i)
   {
      a[min] = a[min -1];
       min--;
   }
  a[i] = key;
}
```

Q13. Bubble sort scans array even when array is sorted can you modify the bubble sort so that it does not scan the whole array once it is sorted.

A better version of bubble sort, known as m bubble sort includes a flag that is set of exchange is made after an entire pass over. If no exchange is made then it should be called the array is already order because no two elements need to be switched

```
void   bubble (int arr[], int n)
{
      for(int i=0;    i<n; i++)
       {
             int swaps=0;
           for(int j=0 ; j<n-i-j ; j++)
           {    if (arr[j] > arr[j+1])
                {
                     int t = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = t;
                     swap++;
                }
            }
         if(swap == 0)
              break;
       }
}
```