Tutorial - 3

Name - Harsh Agarwal
Sec - F
Roll no. - 59

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

(A)
```
for( i = 0 to n)
    {
        if (arr[i] == value)
            // element found
    }
```

Q2. Write Pseudo code for iterative & recursive insertion sort. Insertion sort is called Online sorting. Why? What about other sorting algorithms that has been discussed?

(A) Iterative

```
void insertion_sort (int arr[], int n)
{
    for (int 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;
    }
}
```

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 values it will sort & information is requested while algorithm is running.

Other Sorting Algorithms-

- Bubble Sort
- Quick Sort
- Merge Sort
- Selection Sort
- Heap Sort

complexity of all sorting algorithm that has been discussed in lectures.

(A)

Sorting Algorithm	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 algo into inplace / stable / Online sorting.

(A)

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		

Q3. Write recursion / iterative Pseudo code for binary search. What is Time & space complexity of linear & Binary search.

(A) Iterative

```
int bin_search (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;
}
```

Recursion

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

```
    }
    return -1;
}
```

Time complexity :-

:) Linear Search - $O(n)$

:) Binary Search - $O(\log n)$

Q6. Write recurrence relation for binary recursive search.

(A) $T(n) = T(n/2) + 1$ — (1)

$T(n/2) = T(n/4) + 1$ — (2)

$T(n/4) = T(n/8) + 1$ — (3)

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

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

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

$\vdots$

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

Let $g^k = n$

$k = \log n$

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

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

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

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

→
```
for(i=0; i<n; i++)
{
    for (int j=0; i<n; j++)
    {
        if (a[i] + a[j] == k)
            printf("%d %d", i, j);
    }
}
```

Q8: 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 as stability is not & space is available, merge sort might be best.

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

(A) A pair (A[i], A[j]) is said to be inversion if
- 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 & worst case time complexity.

(A) 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 & either first or last element is selected as pivot.

Best Case $O(n \log n)$ - It occurs when we will select pivot element as a mean element.

Write Recurrence relation of Merge sort & Quick sort in best & worst case. What are the similarities & differences between complexities of these algo & why?

(A) 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 - $T(n) = T(n-1) + O(n) \rightarrow O(n^2)$

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

In merge sort the elements are split into 2 subarray (n/2) again & again until only one element is left.

Q12: Selection sort is not stable by default but can you write a version of stable selection sort?

(A)

```
for(int i=0; i<n; 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-j];
              min--;
          }
```

```
        a[i] = key;
    }
```

Q17. 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) A better version of bubble sort, known as cocktail sort, includes a flag that is set if 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 swap=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;
    }
}
```

Hart