Tutorial-3

Q. Write linear search pseudocode to search an element in a sorted array with minimum comparisions.

```
Void linearSearch ( int A[], int n, int key)
{
    int flag=0;
    for(int i=0; i<n; i++)
    {   if (A[i] == key)
        {
            flag = 1;
            Break;
        }
    }
    if (flag ==0)
        cout<<"Not Found";
    else
        cout<<"Found";
```

Q.2- Write pseudo code for iterative and recursive insertion sort. Insertion sort is called Online sorting. Why? What about other sorting algorithms that has been discussed in lectures.

sol^n -> Iterative

```
for (i=1 to n-1)
{
    t = A[i], j= i-1
    while ( j>=0 && A[j] > t)
    {   (A[j+1] = A[j])
        j--
    }
    A[j+1] = t ;
}
```

Recursive

```
Void insertionSort (int arr[ ], int n)
{
    if (n<=1)
        return;
```

Qn 1 $T(n) = 3T(n/2) + n^2$

$n^{\log_2 3} = n^{1.5}$

$\therefore n^{1.5} < n^2$

$\therefore$

```
insertionSort(arr, n-1),
Int last = arr[n-1], j = n-2;
While(j >= 0 && arr[j] > last)
  {
     arr[j+1] = arr[j];
     j--;
  }
  arr[j+1] = last;
}
```

Ultse Undayuge as Recursion

Insertion sort is called Online sorting because insertion sort considers one input element per iteration and produces a partial solution without considering future elements. But other sorting algorithm requires access to the entire input, thus considered as offline algorithm.

Qn 3 - Complexity of all sorting algorithm that has been discussed in lectures.

Algorithm	Time Complexity		
	Best	Average	Worst
Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$
Count sort	$O(n+k)$	$O(n+k)$	$O(n+k)$
Quick sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Merge sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Heap sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Q. 4 Divide all sorting algorithms into inplace, stable, online.

Algorithm	Inplace	Stable	Online
Bubble sort	✓	✓	X
Selection sort	✓	X	X
Insertion sort	✓	✓	✓
Count sort	X	✓	X
Merge sort	X	✓	X
Quick sort	✓	X	X
Heap sort	✓	X	X

Q. 5 Write Recursive / iterative pseudocode for binary search. What is the time and space complexity of Linear and Binary Search (Recursive and iterative both).

Sol. Recursive ⇒

```
int binarySearch(int arr[], int l, int r, int key)
{   if (r>=l)
    {
        int mid = l+(r-l)/2;
        if (arr[mid] == key)
            return mid;
        if (arr[mid] > key)
            return binarySearch(arr, l, mid-1, key);
        return binarySearch(arr, mid+1, r, key);
    }
    return -1;
}
```

Iterative ⇒

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

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

Algorithm	Time Complexity		Space Complexity	
	Recursive	Iterative	Recursive	Iterative
Linear Search	$O(n)$	$O(n)$	$O(1)$	$O(1)$
Binary Search	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(1)$

Qu 6 → Write Recurrence Relation for binary recursive Search.

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

Qu 7 → Find two indices such that $A[i] + A[j] = k$ in minimum time complexity.

Soln ⇒
```
void Sum(int A[], int k, int n)
{
    Sort (A, A+n);
    int i=0, j=n-1;
    while (i<j)
    {
        if(A[i] +A[j]==k)
            break;
        else if (A[i]+A[j] > k)
            j--;
        else
            i++;
    }
    print(i,j);
}
```

Here sort function has $O(n(\log n))$ complexity and and for while loop it is $O(n)$

$\therefore$ overall complexity = $O(n \log n))$

Q8 - Which sorting is best for practical uses? Explain

Ans 8 For practical uses, we mostly prefer merge sort, because of its stability and it would be best for very large data. Further, time complexity of merge sort is same in all cases, that is $O(n(\log n))$

- Quick sort - Best for practical use

Ans 10 → In which case Quick sort will give the best and the worst case time complexity.

Ans When the array is already sorted or sorted in reverse order, quick sort gives the worst case time complexity i.e. $O(n^2)$, but when the array is totally unsorted, it will give best case time complexity i.e. $O(n \log n)$.

Ans 11 Write Recurrence Relation of Merge and Quick sort in best and worst case? What are similarities and differences b/w complexities of two algorithms and why?

Ans 11 →

Algorithm	Recurrence Relation Best case	Recurrence Relation Worst case
Quick sort	$T(n) = 2T(n/2) + n$	$T(n) = T(n-1) + n$
Merge sort	$T(n) = 2T(n/2) + n$	$T(n) = 2T(n/2) + n$

Both algorithm are based on the divide and Conquer algorithm. both the algorithm have same time complexity in the best and average case.