Adesh Singal
I, 14

Tutorial-3

Q.1) Write Linear Search pseudocode to search an element in a sorted array with minimum comparisons.

```
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.

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

Recursive:- void insertionSort (int arr[], int n)

```
{
    if (n <= 1)
        return;
```

Que: $T(n) = 3T(\frac{n}{2}) + n^2$

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

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

```
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;
}
```

Insertion sort is called Online sorting because Insertion sort considers one Input elements per iteration and produces a partial solution without considering future elements.

But

Qu:- Complexity of all sorting algo. that has been discussed in Algo.

	Time Complexity		
	Best	Average	Worst
Bubblesort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection "	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion "	$O(n)$	$O(n^2)$	$O(n^2)$
Count "	$O(n+k)$	$O(n+k)$	$O(n+k)$
Quick "	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Merge "	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Heap sort O(n log n) O(n log n) O(n log n)

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

Algo.	Inplace	stable	Online
Bubble sort	✓	✓	✗
Selection sort	✓	✗	✗
Insertion sort	✓	✓	✓
Count "	✗	✓	✗
Merge "	✗	✓	✗
Quick "	✓	✗	✗
Heap "	✓	✗	✗

Q.5 write Recursive / iterative pseudocode for binary search. what is the time and space complexity of linear and Binary search (Recursive & iterative both)

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 Comp.		Space complexity	
	Recursive	Iteration	Recursion	Iterative
Linear Search	O(n)	O(n)	O(1)	O(1)
Binary Search	O(logn)	O(logn)	O(logn)	O(1)

Q.6 Write recurrence relation for binary recursive search

$T(n) = T(\frac{n}{2}) + 1$

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

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 for while loop it is $O(n)$.

$\therefore$ Overall Complexity $= O(n \log n)$

Q-8 Which sorting is best for practical uses? Explain.

Ans: for practical uses, we mostly prefer mergesort because of its stability and it would be best for very large data. further, t.c of mergesort is same in all cases, that is $O(n \log(n))$

Q-10 In which case quicksort will give the best & the worst case time complexity.

Ans:- when the array is already sorted or sorted in reverse order, quicksort gives the worst case t.c $O(n^2)$, but when the array is totally unsorted it will give best case t.c $O(n \log n)$

Q-11 write Recurrence Relation of Merge and Quicksort in best and ~~worst~~ worst case? what are similarities and differences b/w complexities of two algorithms and why?

Ans

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

Both algo. are based on the divide and conquer algorithm both the algorithm have same time Complexity in the best & average case.