Satyam Rawat
2017006
G

DAA

Tut 3

Q1.> Write Linear Search Pseudocode for Sorted array.

Soln
```
for (i=0 to n)
  { if (arr[i] == item) cout<<"i;
  }
```

Q2.> Write pseudo code for iterative & recursive insertion sort. Insertion Sort is called Online Sorting. Why? What about other sorting algorithms?

Soln Iterative →

```
void insertsort(int a[], int n)
{ for(int i=1; i<n ; i++)
  { int j=i-1 ;
    int temp = a[i];
    while (j≥0 && a[j] > temp)
     { a[j+1] = a[j] ;
       j--          ;
     }
   a[j+1] = temp ;
  }
}
```

Recursive →

```
void insertsort (int a[], int n)
{ if (n<=1) return;
  insertsort(a, n-1);
  int last = a[n-1];
  int j=n-2;
```

```
while (j >= 0 && a[j] > last)
{ a[j+1] = a[j];
    j--;
  }
  a[j+1] = last;
}
```

Insertion sort is called online sort because it does not require to know anything about what values it will sort & the information is requested while the algorithm is running.

Other sorting algorithms examples are not online sorting algorithms.

Q3:> Complexity of all sorting algorithms.

Sorting Algo	Best	Worst	Average
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Quick Sort	$O(n \log n)$	$O(n^2)$	$O(n \log n)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Count Sort	$O(n)$	$O(n)$	$O(n)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Q4:> Divide all sorting algorithms into inplace / online / stable.

Soln.

Inplace	Stable	Online
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 & Space complexity of Linear & Binary Search (Recursive & Iterative).

Soln:>

Linear Search pseudocodes

Iterative →

```
LinearSearch ( array , item)
{ for (i = 0 ; i<n ; i++)
    { if (array[i] == item)
        return i ;
    }
  return 0-1 ;
}
```

Time complexity = O(N)
Space complexity = O(1)

Recursive →

```
LinearSearch ( arr , item , n)
{ if (n==0) return false ;
  if (arr[n] == item) return true ;
  else LinearSearch (arr, item , n-1) ;
}
```

Time Complexity = O(N)
Space complexity = O(N)

Binary Search

Iterative →

```
BinarySearch (arr, item, n)
{  l=0, r= n-1 ;
   while (l<=r)
   { mid= (l+r)/2 ;
      if(arr[mid] == item) return mid ;
      else if (arr[mid] < item) l= mid+1 ;
      else  r= mid-1 ;
   }
   return -1 ;
}
```

Time Complexity = O(logn)
Space Complexity = O(1)

Recursive →

```
Binary Search (arr, item, l, r), n)
{ if (r>=l  && l<= n-1)
  { mid= (l+r)/2 ;
     if (arr[mid] == item)
                  return mid ;
     if (arr[mid] > item)
                  return BinarySearch (arr, item, l, mid-1, n);
     return BinarySearch(arr, item, mid+1, r, n);
  }
  return -1 ;
}
```

Time Complexity = O(logn)
Space Complexity = O(logn)

Q5 Write Recurrence relation for Binary recursive search;

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

$T(n) = T(n/4) + 1+1 = T\left(\frac{n}{2^2}\right) + 2\times1$

$$T(n) = T\left(\frac{n}{2^k}\right) + k$$

Put $\frac{n}{2^k} = 1 \Rightarrow k = \log_2 n$

$$T(n) = T(1) + \log_2 n$$

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

Q7> Find 2 index such that $A[i] + A[j] = k$.

Soln>
```
for (i=0 ; i<n ; i++)
  { for (j=i+1; j< n ;j++)
      if (a[i] + a[j] == k)
        cout << i << j ;
  }
```

Q8> Which Sorting is best for practical uses? Explain

Soln> Quick Sort is best for practical uses because it is faster than other algorithms (O(nlogn)) and it has space complexity O(logn): It also has good locality of reference.

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

Ans Pair (A[i], A[j]) is said to be inverted if

- A[i] > A[j]
- i < j

Total no of inversion in above array are 31 using merge sort.

Q10> In what case Quicksort will give least & worst Time Complexity,

Ans Least O(nlogn) → When the partitions are equally balanced, their sizes are equal or are within 1 of each other

Worst Case →

When pivot element is at extreme. This happens when input array is reverse sorted.

Q11) Write recursive relation of merge / quick sort in best & worst case. What are similarities & difference between complexities of two algorithms & 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 - $T(n) = 2T(n/2) + O(n)$ $O(n \log n)$

Worst Case - $T(n) = T(n-1) + O(n)$ $O(n^2)$

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

In mergesort, the elements are splitted into 2 sub arrays ($n/2$) again and again until only 1 element is remaining

Q12) Selection sort is not stable by default but can you make a Stable Selection Sort?

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

```
while (min > i)
 { a[min] = a[min-j];
    min--;
 }
 a[i] = key;
}
```

Q13:) Bubble Sort scans array even when array is sorted. Can you modify it so that it does not scan if the array is sorted?

Ans: A better version of bubble sort is m-bubble sort, includes a flag that is set of a exchange is made after an entire pass over. If no exchange is made then it should be called that the array is sorted because no 2 elements were exchanged.

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