Name : Nikita Kumari

Roll : 19

Section : CST SPL-1.

Tutorial - 3

Q1.

```
int linear_search(int a[], int n, int key)
{
if( abs(a[0] - key) > abs(a[n-1] - key))
   {
     for( i = n-1 ; i > 0 ; i--)
          if( a[i] == key)
               return i ;
   }
else {
  for( i = 0 ; i < n ; i++)
       if( a[i] == key)
            return i ;
}
```

Q2. Iterative :

```
Insertion sort( int a[], int n)
{
    int i, j, k ;
for( i = 1 ; i < n ; i++)
  {
        k = a[i] ;
        j = i - 1 ;
while( j >= 0 && a[j] > k)
   {
      a[j+1] = a[j] ;
        j-- ;
   }
```

```
        a[j+1] = x;
    }
}
```

Recursive :

```
Insertionsort( int a[], int n)
{
  if ( n<=1)
      return ;
  Insertion sort( a, n-1);
  int x = a[n-1];
  int j = n-2;
  while( j>=0 && a[j]>x) {
      a[j+1] = a[j];
      j--;
    }
  a[j+1] = x;
}
```

⇒ Insertion sort is called online sorting because it considers only one input per iteration and produces a partial solution without considering future elements whereas other sorting algorithm process the whole problem data altogether from the beginning and is required to output an answer which solve the problem at hand.

Q3. Complexity of all the sorting algorithms -

Sorting	Best	Average	worst
Bubble sort	$\Omega(n^2)$	$\theta(n^2)$	$O(n^2)$
Selection sort	$\Omega(n^2)$	$\theta(n^2)$	$O(n^2)$
Insertion sort	$\Omega(n^2)$	$\theta(n^2)$	$O(n^2)$
Quick sort	$\Omega(n \log n)$	$\theta(n^2)$	$O(n \log n)$
Merge sort	$\Omega(n \log n)$	$\theta(n \log n)$	$O(n \log n)$
count sort	$\Omega(n+m)$	$\theta(n+m)$	$O(n+m)$
Heap sort	$\Omega(n \log n)$	$\theta(n \log n)$	$O(n \log n)$

where m = range.

Q4.

Sorting	Inplace	Stable	Online
Bubble sort	✓	✓	✗
Selection sort	✓	✗	✗
Insertion sort	✓	✓	✓
Quick sort	✓	✗	✗
merge sort	✗	✓	✗
count sort	✗	✓	✗
Heap sort	✓	✗	✓

Q5. Recursive

```
int binarysearch (int a[], int l, int r, int x)
{
        int mid;
    while( l <= r) {
      mid = ( l + r) / 2;
    if( x > a[mid])
            return binarysearch (a, mid+1, r, x);
  else if ( a[mid] > x)
    return binarysearch (a, l, mid-1, x);
     else
          return mid;
     } }
```

Iterative

```
  int binarysearch (int a[], int n, int x)
{
  int l=0, r = n-1, mid;
  while ( l <= r)
{
   mid =  (l+r) / 2;
   if ( x <  a[mid])
         r =  mid-1;
else if ( a[mid] < x)
         l = mid +1;
else    return mid; } }
```

Linear search :- Time complexity = O(n)
space complexity = O(1)

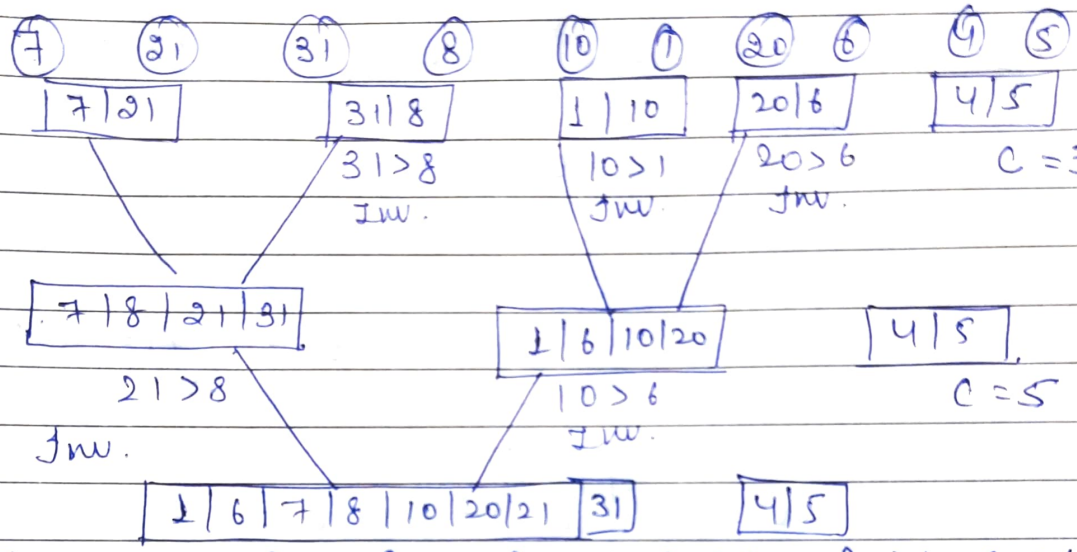
Binary search :- Time complexity = O(log n)
space complexity = O(1).

Q6. Recursive relation for binary search:
$T(n) = T(n/2) + 1$.

Q7.
```
findIndex(int a[], int n, int K)
{
    int j = 0, i = 1;
while( j < n && i < n)
{
    if ( j != i && (a[i] - a[j] == K || a[j]
                - a[i] == K))
            print i, j.
else if (a[i] - a[j] < K)
            i++;
else        j++;
    }
}
```

Q8. Quick sort is one of the most efficient sorting algorithm which makes it one of the most used as well. It is faster as compared to other sorting algorithms. Also, its time complexity is O(n log n). But in case of a larger array, Merge sort is preferred.

Q9. Inversions in an array basically defines how far or close an array is from being sorted. If array is already sorted, inversion count → 0 ; If array is in reverse order, inversion count → maximum.



7>1, 7>6, 8>1, 8>6, 21>10, 21>20, 31>1, 31>6, 31>10, 31>20, 21>1, 21>6. C=17.

1|4|5|6|7|8|10|20|21|31

6>4, 6>5, 7>4, 8>4, 8>5, 10>4, 10>5, 20>4, 20>5, 21>4, 21>5, 31>4, 31>5.

counts = 14

So, 14 + 17 = 31

Total Inversions = 31

Q10. **Best case:**

If pivot/partitioning element is in the middle

Time complexity = O(n log n)

Worst case:

If pivot is at extreme position and array is reverse sorted.
Time complexity = $O(n^2)$.

Q11. Quick sort

Best: $T(n) = 2T(n/2) + n$

worst: $T(n) = T(n-1) + n$.

Merge sort

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

* In merge sort, the array is divided into two equal halves / times.
∴ T.C = $O(n \log n)$.

* In quicksort, the array is divided into any ratio depending on the position of pivot element.
∴ Time comp. ranges from $O(n^2)$ to $O(n \log n)$.

Q12. In selection sort, normally we swap with the first value, which makes is unstable, to make it stable, the code ~~is~~ will be:

the minm valu

```
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 - j];
          min --;
  }
a[i] = key;
}
```

Q13.

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

Q14. In that case, external sorting algorithm such as K-way merge sort is used that can handle large data amount and sort it. which can't fit into main memory.

A part of array resides in RAM during the execution whereas in internal sorting process takes place entirely within the main memory. Ex- Bubble, selection, insertion, etc.