# Tutorial (3)

Ans. (1)

```
int linearSearch (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;
            }
        }
}
```

Ans (2.)

```
insertionSort (int a[], int n)
{
    int i, j, x;
    for (j=1; i<n ; i++)
    {
        x= a[i];
        j= i-1;
```

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

Recursive:

```
insertionSort (int a[], int n)
{
    if (n <= 1)
        return;
    insertionSort(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 sort because it considers only one input per iteration and produces a partial solution without considering future elements

Ans. (3)

| Sorting | Best | Worst | Average |
|---|---|---|---|
| Bubble | $\Omega(n^2)$ | $\Theta(n^2)$ | $O(n^2)$ |
| Selection | $\Omega(n^2)$ | $\Theta(n^2)$ | $O(n^2)$ |
| Insertion | $\Omega(n^2)$ | $\Theta(n^2)$ | $O(n^2)$ |
| Quick | $\Omega(n\log n)$ | $\Theta(n^2)$ | $O(n\log n)$ |
| Merge | $\Omega(n\log n)$ | $\Theta(n\log n)$ | $O(n\log n)$ |
| Count | $\Omega(n+m)$ | $\Theta(n+m)$ | $O(n+m)$ |
| heap | $\Omega(n\log n)$ | $\Theta(n\log n)$ | $O(n\log n)$ |

Ans. (4)

| Sorting | Inplace | Stable | Online |
|---|---|---|---|
| Bubble | ✓ | ✓ | ✗ |
| Selection | ✓ | ✗ | ✗ |
| Insertion | ✓ | ✓ | ✓ |
| Quick | ✓ | ✗ | ✗ |
| Merge | ✗ | ✓ | ✗ |
| Count | ✗ | ✓ | ✗ |
| Heap | ✓ | ✗ | ✓ |

Recursive:

Ans. (5)

```
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 binary Search(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 Comp. $O(n)$

Space " $O(1)$

Binary Search → Time Comp. $O(\log n)$

Space " $O(1)$

Ans. 6 $T(n) = T(n/2) + 1$

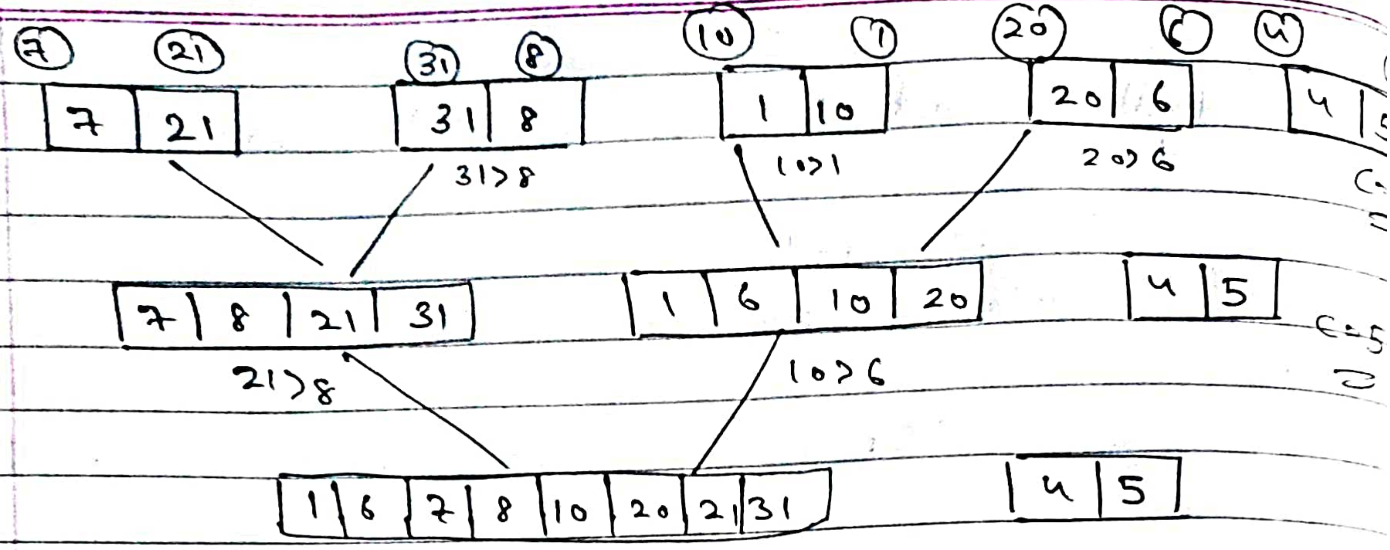
Page No.
Date

Ans. (7)
```
findIndex (int a[], int n, int k)
{
    int i=0, j=1;
    while (i<n && j<n)
    {
        if (i!=j && (a[j]-a[i]==k || a[i]-a[j] == k))
            cout << j << i;
        else if (a[j]-a[i] < k)
            j++;
        else
            i++;
    }
}
```

Ans. (8) Quick Sort is one of the most efficient sorting algorithms 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(nlogn). But in case of a larger array Merge sort is prefffered.

Ans. (9) Inversions in an array define how far or close an array is from being sorted. If array is already sorted inversion ~~count=0~~ count =0, if array is in reverse order, inversion count is maximum.

P.T.O.




7 > 1, 7 > 6, 8 > 1, 8 > 6, 21 > 10, 21 > 20, 31 > 1, 31 > 6,
31 > 10, 31 > 20, 21 > 1, 21 > 6 $\quad$ 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 $\quad$ c = 14

so, 14 + 17 = 31

∴ total inversions = 31

Ans. 10 Best Case:
If pivot / partitioning element is in the middle
Time comp. = O(nlogn)

Worst Case:
If pivot is at extreme position and array is reverse sorted.
Time complexity = O(n²)

Page No.
Date

Ans. 11 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 ~~1~~ time.
∴ T.C. = $O(n \log n)$.

* In quick sort, the array is divided into any ratio depending on the pivot element's position.
∴ T.C. ranges from $O(n^2)$ to $O(n \log n)$.

Ans. 12 In selection sort, normally we swap the min. value with the first value, which makes it unstable to make it stable the code will be:

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

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

Page No.
Date

Ans. (13) void bubbleSort (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;
            }.
    }
```

Ans. (14) In that case, external sorting algo. 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 whereas in internal sorting process takes place entirely within the main memory. e.g. bubble, selection, insertion, et