Satyam Sharma
Section E
37

{ Tutorial – 3 }

① Write linear search pseudo code to search an element in a sorted array with minimum no. of comparison.

→ Sol^n

```
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";
```

Ques ② write pseudo code for iterative and recursive

→ Sol^n ② 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;
  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;

}
```

Ques:- (3) Complexity of all sorting algorithm that has been discussed.

Sol^{n}

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

Ques ④ Divide all sorting algorithms into inplace/Stable/Online

Sol^n

Algorithm	Inplace	Stable	Online
Bubble Sort	✓	✓	✗
Selection Sort	✓	✗	✗
Insert Sort	✓	✓	✓
Count sort	✗	✓	✗
Merge Sort	✗	✓	✗
Quick Sort	✓	✗	✗
Heap Sort	✓	✗	✗

Ques ⑤ Write recursive/Iterative pseudo code for ....

⇒ ⑤ 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;
}
```

(6) Find two indices such that A[i]+A[j] = K in minimum time complexity.

Sol":-

```
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);
}
```

Time Complexity O(nlogn).

Ques (8) Which Sorting is best for practical uses? Explain.

Sol → (8) In practical uses, we mostly prefer merge sort. because of its stability and it can best for very large data. further more, the time complexity of merge sort is same in all cases that is $O(n \log n)$.

Ques (9) What do you mean by the number of inversion

Sol (9) Inversion count for an array indicates — how far the array is from being sorted. If the array is already sorted, inversion count is 0, but if the array is sorted in reverse order the inversion count is maximum.

Pseudo Code for inversion Count :-

```
int getCount (int arr[], int n)
{   int c=0;
    for (i=0; i<n-1; i++)
        for(int j=i+1; j<n; j++)
            if (arr[i] > arr[j]
                c++;
    return c; }
```

Ques-(10) Selection Sort is not stable by a default why..?

Sol":- 10) As the selection sort is not stable because it changes the relative position of some elements after sorting.

Selection Sort can be made stable if instead of swapping the minimum element is placed in its position without swapping that is by placing the number in its position by pushing every element one step forward. In simple words use insertion sort techni which means inserting element in its correct place.

Pseudo Code for stable selection Sort;

```
Void StableSelectionSort (int A[], int n)
{   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-1]
            min--;
          }
       a[i] = key;
}
```

Ques (i) Bubble sort scans whole array even when array is sorted can you modify the bubble sort so that it doesn't scan the whole array once it is sorted.

→ (ii) we can modify bubble sort by placing a flag variable. If array is already sorted we can halt the process by checking the flag variable if its value change or not.

pseudo code :-

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