Tutorial - 3

Name :- Daksh Goel

Section :- F

Roll no. :- 51

Univ. Rollno :- 2016713

Q1. Write linear search pseudo code to search an element in a sorted array with minimum comparisons.

⇒
```
for (i = 0 to n)
{
  if (arr [i] = = value)
          // element formed

}
```

Q2. Write pseudo code for iterative and recursive insertion sort. Insertion sort is called online sorting. Why? What about other sorting algorithms that has been discuss in lectures?

⇒ Iterative :-

```
void insertionsort (int A[], int n)
{
   for (int i = 1; i < n; i++)
   {
      j = i-1;
       x = A[i];
   while (j > -1 && A[i] > x)
      {
        A[j+1] = A[j]
            j--;
      }
   A [j+1] = x;
   }
}
```

Daksh

Recursive:-

```
void insertionsort (int arr[], int n)
{
      if (n <=1)
        return;
   insertion sort (arr, n-1);
   int last = arr[n-1];
      int j = n-2;
while (j >=0 && arr[j] > last)
   {
     arr[j+1] = arr[j];
          j--;
   }
  arr [j+1] = last;
  }
```

Insertion sort is called online sort because it does not need to know anything about what values it will sort and the information is requested WHILE the algorithm is running.

Other sorting algorithm:-

- Bubble sort
- Quick sort
- Merge sort
- Selection sort
- Heap sort

Q3. Complexity of all the following algorithm that has been discussed in lectures.

=>

	Best	Worst	Average
Selection sort	O(n^2)	O(n2)	O(n^2)
Bubble sort	O(n)	$O(n^2)$	$O(n^2)$
Insertion sort	O(n)	O(n^2)	$O(n^2)$
Heap sort	O(n logn)	O(n logn)	O(n logn)
Quick sort	O(n logn)	$O(n^2)$	O(n logn)
Merge sort	O(n logn)	O(n logn)	O(n logn)

Daksh

Q4. Divide all the sorting algorithms into implace / stable / online sorting.

⇒ Implace sorting
- Bubble
- Selection
- Insertion
- Quick sort
- Heap sort

stable sorting
- Merge sort
- Bubble
- Insertion
- Count

online sorting
- Insertion

Q5. Write recursive / iterative pseudo code for binary search. What is the Time and space complexity of linear and binary search (Recursive and iterative).

⇒ Iterative :-

```
int binarySearch(int arr[], int l, int r, int key)
{   while(l<=r)
      {
         int m = ((l+r)/2);
         if(arr[m] == key)
               return m;
    else if(key < arr[m])
               r=m-1;
        else
             l=m+1;
         }
         return -1;
      }
```

Recursive :-

```
int binarySearch(int arr[] , int l, int r, int key)
   {
   while (l<=r)
     {
       int m=((l+r)/2);
      if (key == arr[m])
            return m;
   else if (key < arr[m])
```

Daksh

```
return binarysearch (arr, l, mid-1, key);
else
return binary search (arr, mid+1, r, key);
}
return -1;
}
```

Time Complexity:-

- Linear Search - $O(n)$
- Binary Search - $O(n \log n)$

Q6. Write recurrence relation for binary recursive search.

⇒ $T(n) = T(n/2) + 1$ — ①

$T(n/2) = T(n/4) + 1$ — ②

$T(n/4) = T(n/8) + 1$ — ③

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

$= T(n/4) + 1 + 1$ (from eqn - ②)

$= T(n/8) + 1 + 1 + 1$ (from eqn - ③)

$T(n/g^k) + 1$ (k times)

Let $g^k = n$ $\quad T(n) = T(n/n) + \log n$

$k = \log n$ $\quad T(n) = T(1) + \log n$

$T(n) = O(\log n)$

Q7. Find two indexes such that $A[i] + A[j] = k$ in minimum time complexity.

⇒
```
for(int i=0; i<n; i++)
{
    for(int j=0; j<n; j++)
    {
        if(a[i]+a[j]==k)
```

Daksh

```
        printf ("%d %d", i, j);
      }
   }
```

Q8. Which sorting is best for pratical uses? Explain.

⇒ Quick sort is the fastest general-purpose sort. In most pratical situations quicksort is the method of choice. If stability is important and space is available, merge sort might be best.

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

⇒ • A Pair (A[i], A[j]) is said to be inversion is
- A[i] > A[j]
- $i < j$

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

Q10:- In which cases Quick sort will give the best and the worst case time complexity?

⇒ **Worst case ($O(n^2)$)**:- The worst case occurs when the picked pivot is always an extreme (smallest or largest) element. This happens when input array is sorted or reverse sorted and either first or last element is picked as pivot.

Best case ($O(n \log n)$):- The best case occurs when we will select pivot element as a mean element.

Darsh

Q11. Write Recurrence Relation of merge and quick sort in best and worst case? What are the similarties and differences between complexities of two algorithm and why?

=> 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:- $T(n) = 2T(n/2) + O(n) \rightarrow O(n \log n)$

Worst case:- $T(n) = T(n-1) + O(n) \rightarrow O(n^2)$

In quick sort the array of elements is divided into parts repeatedly until it is not possible to divide it further. It is not necessary to divide half.

In merge sort the elements are split into two sub-array (n/2) again and again until only one element is left.

Q12. Selection sort is not stable by default but can you write a version of stable selection.

=>
```
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;
    }
```

Tapsh

Q13. Bubble sort scans array even when array i sorted. Can you modify the bubble sort so that it does not scan the whole array once it is sorted.

=) A better version of bubble sort, known as m bubble sort, includes a flag that is set if exchange is made after an entire pass over the array. If no exchange is made, then it should be the - array is already order because no two element need to be switched. In that case sort is end.

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

Daksh