Tutorial 3

Data & analysis of algorithm.

Ques:- Write linear search pseudocode to search an element in a sorted array with min. comparisons.

```
void jumpsearch ( int a[], int e, int n )
{  int m = sqrt(n), s
   for( s = 0; a[s] <= e; s += m )
      { if ( a[s] == e )
        { cout << "found" ;
          return;
        }
      }
   s -= m;
   for ( s ; s <= s+m; ++s )
      { if ( a[s] == e )
        { cout << "found" ;
          return;
        }
      }
   cout << " not found" ;
}
```

Ques-2 Write pseudocode for iterative and recursive insertion sort. Insertion sort is called online sorting. why? what about other sorting algo that has been discussed in lectures?

Ans 2:- Iterative :-

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

Recursive :-

```
void insertion( int arr[] , int i, int n)
 { int value = arr[i];
   int j = i ;
   while( j>0 && arr[j-1] > value).
    { arr [j] = arr[j-1],
      j-- ;
    }
    arr [j ] = value;
   if( i+1 <= n)
    { insertion(arr, i+1, n);
    }
}
```

Insertion sort is called an online sorting algo because it considers an input element per iteration and produces a partial solution without considering future elements

Ques-3 Complexity of all sorting algo that has been discussed in lecture.

Ans.

	Best	Avg	Worst
Bubble sort :-	$O(n)$	$O(n^2)$	$O(n^2)$
Selection sort :-	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion sort :-	$O(n)$	$O(n^2)$	$O(n^2)$
Heap sort :-	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick sort :-	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Merge sort :-	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Count sort :-	$O(n + range)$	$O(n + range)$	$O(n + range)$

Ques-4 Divide all the sorting algo into inplace / stable / online sort

Ans.

	Inplace	Stable	Online
Bubble sort	Yes	Yes	No
Selection sort	Yes	No	No
Insertion sort	Yes	Yes	Yes
Quick sort	Yes	No	No
Merge sort	No	Yes	No
Heap sort	Yes	No	No

Ques-5 Write recursive / iterative pseudocode of binary search. What is $T(n)$ and space complexity of linear & binary search.

Iterative -

```
int binary(int a[], int x)
{ int low=0, high= a.length-1;
  while (low<=high)
  { int mid= (low+high)/2;
```

```
if (n == a[mid])
    return mid;
else if (n < a[mid])
    high = mid-1;
else
    low = mid+1;
}
return -1;
}
```

Recursive :-

```
int binary(int a[], int low, int high, int n)
{ if (low > high)
    return -1;
  int mid = (low+high)/2;
  if (n == a[mid])
    return mid;
  else if (n < a[mid])
    return binary(A, low, mid-1, n);
  else
    return binary(A, mid+1, high, n);
}
```

Time complexity :- Iterative :- O(log n)
Recursive :- O(log n)

Space complexity :- Iterative - O(1)
Recursive - O(log n)

Ques:- Write recurrence relation for binary recursive search

Recurrence relation - $T(n) = T(n/2) + 1$

Derivation :-

1st step :- $T(n) = T(n/2) + 1$

2nd step :- $T(n/2) = T(n/4) + 1$ .. $[T(n/4) = T(n/2^2)]$

3rd step :- $T(n/4) = T(n/8) + 1$ .. $[T(n/8) = T(n/2^3)]$

:

n^{th} step :- $T(n/2^{k-1}) = T(n/2^k) + 1$ (1 time)

Adding all eq:

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

$\Rightarrow n/2^k = 1$

$= n = 2^k$

$\Rightarrow \log n = k$

$= k = \log n$

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

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

Q-7 :- Find two indexes such that $A[i] + A[j] = k$ in min time complexity.

```
vector<int> find(arr[], k, n)
{ vector<int> sol;
  for i=0 to n-1
    for j=0 to n
      if arr[i] + arr[j] = k
          sol.push_back(i)
          sol.push_back(j)
          return sol
}
```

Ques-8 Which sorting is best for practical use? Explain.

Ans:- Quicksort is fastest general-purpose sort. In most practical situation quicksort is a method of choice. If stability is imp. & space is available, merge sort might be best. In some performance-critical application, the focus may be on just sorting number, so it is resonable to avoid the costs of using references and sort primitive types instead

Ques-9 What do you mean by no of inversion in an array? Count the no. of inversions in Array [ ]:
{ 7, 21, 31, 8, 10, 1, 20, 6, 4, 5 }

Ans:- Inversion count for an array indicates how far or close the array is from being sorted. If the array is already sorted then the inversion count is 0, but if the array is sorted in reverse order, the inversion count is maximum.

Pair inversion in array arr[]:- (7,1) (7,6) (7,4) (7,5) (21,8) (21,10)
(21,1) (21,20) (21,6) (21,4) (21,5) (31,8)
(31,10) (31,1) (31,20) (31,6) (31,4)
(31,5) (8,1) (8,6) (8,4) (8,5) (10,1)
(10,6) (10,4) (10,5) (20,6) (20,4)
(20,5) (6,4) (6,5)

Inversion Count = 31

Ques-10 Best case :- The best case occurs when the partition process always picks middle element as pivot. Following is the recurrence relation for best case

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

Worst case :- The worst case occurs when the partition process always picks greatest or smallest elements as pivot. If above partition strategy is considered where last element is always picked as pivot, the worst case would occur when the array is already sorted in increasing or decreasing order

Ques-11 Quick sort →

Recurrence relation → Best case - $T(n) = 2T(n/2) + \Theta(n)$

Worst case - $T(n) = T(n-1) + \Theta(n)$

Merge sort →

Recurrence relation → Best case - $2T(n/2) + \Theta(n)$

Worst case - $2T(n/2) + \Theta(n)$

Time complexity	Best case	Worst case
Quick sort	$O(n \log n)$	$O(n^2)$
Merge sort	$O(n \log n)$	$O(n \log n)$

Ques-12

```
void stable(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-13 :-

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

Ques-14

As the size of given array exceeds the size of RAM therefore we will use k-way merge sort ~~the~~ sorting technique as it takes a part of array & sort it, whole array is not loaded into main memory ~~at~~ altogether.

External sorting:- This algo ~~uses~~ loads a part of array and sort it whole array is not loaded into the RAM. especially used to sort array of huge size.
eg:- k-way merge sort

Internal Sorting:- These algo needs whole ~~algo~~ array to reside in RAM during execution.
ex:- bubble sort.