① Lakshay Sharma
AI and DS
10

Tutorial-3

1) Pseudo code for linear search

```
for (i=0 to n)
{  if (arr[i] == key)
        printf("Element found")
}
```

2) Recursive

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

iterative

```
for (i=1 to n)
    key = A[i]
    j = j-1
    while (j>=0 && A[j] > key)
        { A[j+1] = A[j]
          j = j-1; }
A[j+1] = key;
```

insertion sort is online sorting because it doesn't known the whole input, more input can be inserted while the insertion sorting is running.

3) Complexity of different sorting algorithms

Name	Best case	worst case	Average
Selecting sorting	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble sorting	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion sorting	$O(n)$	$O(n^2)$	$O(n^2)$
Heap	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick	$O(n \log n)$	$O(n^2)$	$O(n \log n)$
merge	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

4) Implace sorting

Bubble
~~Selecting~~ Selection
Insertion
quick
heap

Stable sorting

Merge
~~Bubble~~
Insertion

Online sorting

Insertion

5) Iterative

```
int b_search (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;
}
```

Time complexity → $O(n)$

Recursive:-

```
int b_search(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])
        return b_search(arr, l, mid-1, key);
    else
        return b_search(arr, mid+1, r, key);
} return -1; }
```

Time complexity → $\boxed{O(\log n)}$

6) $T(n) = T(n/2) + 1$ —— (1)

$T(n/2) = T(n/4) + 1$ —— (2)

$T(n/4) = T(n/8) + 1$ —— (3)

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

$= T(n/4) + 2$

$= T(n/8) + 3$

$= T\left(\frac{4}{2^k}\right) + k$

let $g^k = n$

$k = \log n$

$T(n) = T\left(\frac{n}{n}\right) + \log n$

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

Time complexity ⇒ $\boxed{O(\log n)}$

7)
```
for (i=0; i<n; i++)
  { for (int j=0; j<n; j++)
    { if (arr[i] + arr[j] == k)
        printf("%d %d", i, j); }
  }
```

8) Quick sort is fastest general-purpose sort. In most practical situation quicksort is the method of choice is stability is important and space is available, merge sort might be best.

9) Inversions in array :-
- A pair (A[i], A[j]) is said to be inversion of A[i] > A[j]
- i < j
- Total no of inversions is given array are 31 using merge sort.

10) Worst case ($O(n^2)$) :-
- when the pivot element is an extreme (smallest/largest) element. This happens when input array is sorted or reverse sorted and either first or last element is selected as pivot.

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

11) Merge sort :-
- Best case :- $T(n) = 2T(n/2) + O(n)$
- Worst case :- $T(n) = 2T(n/2) + O(n)$

$\} \leftarrow O(n \log n)$

Quick sort :-
- Best case :- $T(n) = 2T(n/2) + O(n) \longrightarrow O(n \log n)$
- Worst case :- $T(n) = T(n-1) + O(n) \longrightarrow O(n^2)$

- In quick sort, array of element are divided into 2 pair repeated until it is not possible to divide further.
- In merge sort, the elements are split into 2 subarray (n/2) again and again until 1 only 1 element is left.

12)
```
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; }
```

13) A better version of bubble sort, known as modified bubble sort, includes a flag that is set of a exchange is made after an entire pass over. If no exchange is made then it should be called the array is already order because no 2 elements need to be switched.

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

}}
```