NAME :- MIHIR ARYAN ANURAJ

CST 38

TUTORIAL - 3

① Pseudo code for linear search

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

②

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

```
for (i=1 to n)              // iterative
    key ← A[i]
    j ← i-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 doesnt know the whole input, more input can be inserted with the insertion sorting is running

③ complexity

Name	Best	Worst	Average
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$
Heap sort	$O(n\log(n))$	$O(n \log n)$	$O(n \log(n))$
Quick sort	$O(n\log(n))$	$O(n^2)$	$O(n\log(n))$
Merge sort	$O(n\log(n))$	$O(n\log n)$	$O(n\log(n))$

④

Inplace Sorting	Stable sorting	Online sorting
Bubble	Merge sort	Insertion
Selection	Bubble	
Insertion	Insertion	
Quick sort	Count	
Heap sort		

⑤
```
int binary (int arr[], int l, int r, int x)
{ if (r>=l)
    { int mid = l + (r-l)/2;
      if (arr[mid] == x)
          return mid;
      else if (arr[mid] > x)
          return binary (arr, l, m-1, x);
      else
          return binary (arr, m+1, r, x);
    }
  return -1;
}
```

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

Time complexity of
Binary Search => $O(\log n)$
Linear Search => $O(n)$

⑥ Recurrence relation for binary recursive search.

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

where $T(n)$ is the time required for binary search in an array of size 'n'.

⑦
```
int find (A[], n, k)
{ sort (A, n)
  for (i= to n-1)
    { x= binary search (A, 0, n-1, k - A[i])
      if (n)
      return 1
    }
  return -1;
}
```

Time Complexity $= O(n \log(n)) + n \cdot O(\log n)$
$= O(n \log(n))$

(8) • Quick Sort is the fastest general purpose sort.
• In most practical situations, quicksort is the method of choice. If stability is important & space is available, merge sort might be best.

(9) A pair $(a[i], a[j])$ is said to be inversion if $a[i] > a[j]$

In arr[] = {7, 2, 31, 8, 10, 1, 20, 6, 4, 5}
total no. of inversions are 31, using merge sort.

(10) The worst case time complexity of quick sort is $O(n^2)$
This case occurs when the picked pivot is always an extreme (smallest or largest) element. This happen when input array is sorted or reverse sorted.
The best case of quick sort is when we will select pivot as a mean element.

(11) Recurrence relation of
Merge Sort → $T(n) = 2T(n/2) + n$
Quick Sort → $T(n) = 2T(n/2) + n$

• Merge Sort is more efficient & work faster than quick sort in case of large array size or data sets
• Worst case complexity for quick sort is $\mathcal{E}(n^2)$ whereas $O(\log(n))$ for merge sort.

(12) Stable selection sort

```
void stable selection (int arr [], int n)
 { for (i = 0; i<n-1; i++)
    { int min = i;
    for (int j = i+1 ; j<n; j+1)
       { if (arr [min] > arr[j])
             min = j:
       }
    int key = arr[min];
    while (min > i)
     {  arr [min] = arr [min -1];
        min --;
     }
    arr[i] = key;
   }
}
```

(13) Modified bubble sorting

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