DAA

Q1.) Ans

Pseudocode for linear search is

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

Q2.) Ans.

```
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]>=n)
  {
    arr[j+1] = arr[j];
    j--;
  }
  arr[j+1] = nth;
}
for (i=1 to n)
{ key ← A[i];                    //iterative
  j ← i-1;
```

```
while (j >= 0 && A[j] > key)
{
    A[j+1] <- A[j];
    j <- j - 1;
}
A[j+1] <- key;
}
```

Insertion sort is an online sorting technique because it doesn't have the whole input, more input can be inserted with the ~~inserting~~ insertion sort running.

Q3.)

Ans

Complexity

Name	Best	Worse	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	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Q4.) Ans

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

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

Time complexity of binary search = $O(\log n)$

Time complexity of linear search = $O(n)$

Q6.) Ans Recurrence relation for ...

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

where $T(n)$ is the time required for binary search in array of size n.

Q7.) Ans
```
int find(A[], n, k)
{ sort(A, n);
  for(i=0 to n-1)
  { n = binarysearch(A, 0, n-1, k-A[i]);
    if(n)
      return 1;
  }
  return(-1);
}
```
Time complexity = $O(n \log(n)) - n, O(\log n) = O(n \log n)$

Q8.) Ans → Quick sort is the fastest general purpose ~~test~~ sort.
→ In most practical situations, quicksort is the method of choice. If stability is important and space is available, however, then merge sort might be the best.

Q9.) Ans A pair $(a[i], a[j])$ is said to be inversion of $(a[i] > a[j])$ In arr[] = {7, 21, 31, 8, 10, 1, 3, 9, 15}, total no of inversions are 31 using merge sort.

Ans 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 happens when input array is sorted or reverse sorted.
The best case of quick sort is when we will select pivot as a mean element.

Ans Recurrence solution :-
Merge sort $\Rightarrow T(n) = 2T(n/2) + n$
Quick sort $\Rightarrow T(n) = 2T(n/2) + n$ works

- Merge sort is more efficient and much faster than quick sort in case of larger array size or datasets.
- Worst case complexity for quick sort is $O(n^2)$ whereas $O(n\log n)$ for merge sort.

Ans Stable selection sort :-

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

(Q 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];
        a[j] = a[j+1];
        a[j+1] = t;
        swaps++; }
    }
  }
  if(swaps == 0)
    break;
  }
}
```