Tutorial - 3

Ques1

```
int linear_search (int A[], int n, int t)
{ if (abs(A[0] - t) > abs(A[n-1] - t))
    for (i=n-1 to 0 ; i--)
      if (A[i] == t)
        { return i ;
        }
      else
       for (i=0 to n-1 ; i++)
        if(A[i] == t)
        return i;
}
```

Ques 2 Iterative Insertion Sort

```
Void insertion (int A[], int n)
{ for (i=1 to n)
  { t = A[i];
    j = i;
  while (j≥0 && t < A[j])
  {
    A[j+1] = A[j];
    j--;
  }
  A[j+1] = t;
  }
}
```

Recursive Insertion Sort

```
Void insertion (int A[], int n)
```

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

Insertion Sort is also called online sorting algorithm because it will if the elements to be sorted are provided are at a time with the understanding that the algorithm must keep the sequence sorted as more elements are added in.

Other sorting algorithms like bubble sort, insertion sort, heap sort etc are considered external sorting technique as they need the data to be sorted in advance.

Ques 3

Sorting	Best case	Worst case
Bubble Sort	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$
Count Sort	$O(n)$	$O(n+k)$
quick sort	$O(n \log n)$	$O(n^2)$
merge sort	$O(n \log n)$	$O(n \log n)$
heap sort	$O(n \log n)$	$O(n \log n)$

Ques4

Sort	Inplace	Stable	Online
Bubble	✓	✓	✗
Selection	✓	✗	✗
Insertion	✓	✓	✓
Count	✗	✓	✗
quick	✓	✗	✗
merge	✗	✓	✗
Heap	✓	✗	✗

Ques6

$T(n)$
↓
$T(n/2)$
↓
$T(n/4)$
↓
⋮
↓
$T(n/2^k)$

recurrance relation $= T(n/2) + 1$ °/1.

Ques7

```
int n, A[n], key;
int i=0, j=n-1;
while (i<j)
{
  if (A[i]+A[j]==key)
    break;
else if ((A[i]+A[j])>key)
   j--;
else
   i++;
}
Count <<i<<" "<<j;
```

O(n log n)

Ques5

```
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;
}
```

T.C of binary search = O(log n)
linear search = O(n)

Ques8

* Quick sort is a fastest general purpose sort.
* In most practical situations, quicksort is the method of choice if stability is important and space is available, merge sort might be best.

Ques9

A pair (a[i], a[j]) is said to be inversion if a[i] > a[j]
In arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5}
Total no of inversion are 31, using merge

Ques10

worst complexity is $O(n^2)$ of ~~Count~~ quick sort, 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.

Ques11 Recurrence relation of

merge sort → $T(n) = 2T(n/2) + n$
quick sort → $T(n) = 2T(n/2) + n$

- merge sort is more efficient and works 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.

Ques12 Stable selection sort

```
Void stable selection (int arr[], int n)
{for (int i=0; i< n-1; i++)
  {
```

```
int mn=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;
}
}
```

Ques 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[i];
              a[j]=a[j+1];
              a[j+1]=t;
              swaps++;
            }
        }
      if (swaps==0)
        breaks;
    }
}
```