Tutorial-2.

Q1.

i	j
1	2
3	3
6	4
10	5
⋮	⋮

$\frac{k(k+1)}{2} = n$

$k^2 = n$

$k = \sqrt{n}$

$\Rightarrow T.C = O(\sqrt{n})$.

Q2. $T(n) = T(n-1) + T(n-2) + 1$.

let $T(n-1) \simeq T(n-2)$.

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

using backward substitution.

$T(n) = 2 \cdot 2(T(n-2)+1)+1$

$= 4(T(n-2)) + 3$.

$T(n-2) = 2T(n-3) + 1$.

$T(n) = 2(2(2(T(n-3)+)+1)+1)$

$= 8T(n-3) + 3$.

$T(n) = 2^k T(n-k) + 2^k - 1$.

$T(0) = 0$

$n - k = 0$

$n = k$.

$T(n) = 2^n (T(n-n)) + 2^n - 1$

$= 2^n + 2^n - 1$

$T.C \Rightarrow O(2^n)$.

Q3. $n(\log n)$.

```
void func(int n).
{ for (i=1; i<=n; i++)
    { for (j=1; j<=n; j=j*2)
        {   // some O(1) task.
        }
    }
}
```

(n^3)

```
void func (int n).
{ for (i=1 to n)
    { for (j=1 to n)
        { for (k=1 to n)
            {
                // some O(1) task
            }
        }
    }
}
```

$(\log(\log(n)))$

```
void func (int n)
{ for(i=n; i>1; i=pow (i,k))
    {
      //some O(1) task
    }
}
```

Q4. $T(n) = T(n/4) + T(n/2) + cn^2$.

Assume $T(n/2) > T(n/4)$.

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

$$c = \log_b^a$$

$$= \log_2^2 = 1.$$

$$\therefore n^c < f(n)$$

$T(n)$ $O(n^2)$.

Q5.

i	j
1	n times
2	n/2 times
3	n/3 times
⋮	⋮
n/n	n/n times

$\therefore T.C = O(n \log n)$.

Q6. $i = 2, 2^k, (2^k)^k, (2k^2)^k = 2^{k^3} \dots 2^{k \log k (\log(n))}$.

$$2^{k \log k (\log(n))} = n.$$

$$2^{\log(1)} = 1.$$

$$\Rightarrow T.C = O(\log(\log(n)))$$

Q7. $T(n) = T(9n/10) + T(n/10) + O(n)$.

taking one branch 99% & other 1%

$T(n) = T(99n/100) + T(n/100) + O(n)$.

1'st level $= n$.

IInd level $= 99n/100 + n/100 = n$.

So III rd level remains same for any kind of position.

$\therefore$ If we take longer branch $= O(n \log_{100/99} n)$.

for shorter branch $= \Omega(n \log_{10} n)$.

either way base complexity of $O(n(\log n))$ remains

Q8 (a.) $100 < \sqrt{n} < \log(\log n) < \log(n) < n < n \log n < \log n! < n^2 < n! < 2^n < 4^n < 2^{2n}$

(b.) $1 < \log(\log n) < \sqrt{\log n} < \log n < \log 2n < n < n \log n = \log(n!) < 2n < 4n = 2(2^n) < n! < n^2$

(c) $96 < \log_2 n < \log n! < n\log_2 n < n\log_6 n < 5n < n! < 8n^2 < 7n^3 < 8^{n^{2n}}$

Q9. Linear Search (Array size, key, flag).

```
Begin
    for (i=0 to n-1)
        if (array[i] == key)
            set flag=1
            break
    if flag = 1
        return flag.
    else
        return -1.
    end.
```

Q10. Iterative

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

Recursive

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

Q11.

	Best	Average	Worst
Selection	$\Omega(n^2)$	$\theta(n^2)$	$O(n^2)$.
Bubble	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$
Insertion	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$
Heap.	$\Omega(n\log n)$	$\theta(n\log n)$	$O(n\log n)$
Quick	$\Omega(n\log n)$	$\theta(n\log n)$	$O(n^2)$
Merge	$\Omega(n\log n)$	$\theta(n\log n)$	$O(n\log n)$.

Q12. Bubble, Insertion & selection sorts are ~~interface~~ inplace sorting algorithms.

Bubble & insertion sort can be applied as stable algo. but selection sort can't.

Merge sort is a stable algo. but not an inplace algo.

Quick sort isn't stable but it is inplace algo.

Heap sort is an inplace algo but not stable.

Q13.
```
int binary (int a[], int x)
{ int l=0, h=A.length-1;
  while (l<=h)
  { int mid = (l+h)/2;
    if (x == A[mid])
        return mid;
    else if (x < A[mid])
        h = mid-1;
    else
        l = mid+1;
  }
  return -1;
}
```

Q14. $T(n) = T(n/2)+1$.