Recursion methods
1. Decrease and conquer
2. Divide and conquer
3. Backtracking

We will cover time complexities for decrease and conquer, divide and conquer methods here, along with master theorems for both.
For understanding in detail, along with proofs. please visit Bari’s Algorithm Playlist(18-29) Link:
https://www.youtube.com/playlist?list=PLDN4rrl48XKpZkf03iYFl-O29szjTrs_O

```
//-----------------------------------------------------------------------------
// RECURRENCE RELATIONS for DECREASING FUNCTIONS
//-----------------------------------------------------------------------------

//Example 1:

public void test(int n){
      if(n>0){
            //print value
            System.out.println(n);
            test(n-1);
      }
}

//Recurrence relation: T(n) = T(n-1) + 1;
//Time Complexity --> O(n)

//Example 2:

public void test(int n){
      if(n>0){
            //print value
            for(int i=0; i<n; i++){
                  System.out.println(n);
            }
            test(n-1);
      }
}

//Recurrence relation: T(n) = T(n-1) + n;
```

```
//Time Complexity: O(n^2)

public void test(int n){
	if(n>0){
		//print value
		for(int i=0; i<n; i=i*2){
			System.out.println(n);
		}
		test(n-1);
	}
}

//Recurrence relation: T(n) = T(n-1) + logn;
//Time Complexity: O(n logn)

//Few Recurrence Relations and their time complexities

//Recurrence relation: T(n) = T(n-1) + 1;
//Time Complexity: O(n)

//Recurrence relation: T(n) = T(n-1) + n;
//Time Complexity: O(n^2)

//Recurrence relation: T(n) = T(n-2) + 1;
//Time Complexity: O(n/2) -> O(n)

//Recurrence relation: T(n) = T(n-100) + n;
//Time Complexity: O(n^2)

//Recurrence relation: T(n) = T(n-1) + n^2;
//Time Complexity: O(n^3)

public void test(int n){
	if(n>0){
		//print value
		System.out.println(n);
		test(n-1);
		test(n-1);

	}
}
```

```
//Recurrence relation: T(n) = 2 T(n-1) + 1;
//Time Complexity:  O(2^n)

//Recurrence relation: T(n) = 3 T(n-1) + 1;
//Time Complexity:  O(3^n)

//Recurrence relation: T(n) = 2 T(n-1) + n;
//Time Complexity:  O(n.2^n)



//--------------------------------------------------------------------------
// Masters Theorem for DECREASING FUNCTIONS
//---------------------------------------------------------------------------

// T(n) = a T(n-b) + f(n)

// where a>0, b>0, and f(n) =  O(n^k) , where k >= 0

// Case 1) : if a < 1,  O(n^k)  i.e. O(f(n))
// Case 2) : if a = 1,  O(n^(k+1))  i.e. O(n.f(n))
// Case 3) : if a > 1,  O(n^k.a^(n/b))  i.e. O(f(n).  a^(n/b) )

//---------------------------------------------------------------------------
// RECURRENCE RELATIONS for DIVIDING FUNCTIONS
//---------------------------------------------------------------------------

public void test(int n){
      if(n>1){
            //print value
            System.out.println(n);
            test(n/2);
      }
}

//Recurrence relation: T(n) = T(n/2) + 1;
//Time Complexity: O(logn)

//Recurrence relation: T(n) = T(n/2) + n;
//Time Complexity: O(n)
```

```
public void test(int n){
      if(n>1){
            //print value
            for(int i=0; i<n; i++){
                  System.out.println(n);
            }
            test(n/2);
            test(n/2);
      }
}

//Recurrence relation: T(n) = 2.T(n/2) + n;
//Time Complexity: O(n.logn)

//----------------------------------------------------------------------------
// Masters Theorem for DIVIDING FUNCTIONS
//----------------------------------------------------------------------------
```

//T(n) = a. T(n/b) + f(n)

//where a >= 1, and b>1, and f(n) = $O(n^k . log^p n)$

Case 1: $log^a b$ > k, then its $O(n^{log^a b})$

Case 2: $log^a b$ = k, then 3 sub cases

if p > -1, then $O(n^k . log^{p+1} n)$
if p = -1, then $O(n^k . log\ log\ n)$
if p < -1, then $O(n^k)$

Case 3: $log^a b$ < k, then 2 sub cases

if p >=0, then $O(n^k . log^p n)$
if p < 0, then $O(n^k)$

Case 1 example 1:

//Recurrence relation: T(n) = 2. T(n/2) + 1;

a=2, b=2, and f(n) = 1 = $O(n^0 . log^0 n)$, so k=0, p=0.
and $log^2 2$ = 1, so $log^2 2$ > 0 holds true.

So, time complexity is $O(n^{log^a b}) = O(n^1)$ *i.e.* $O(n)$

Case 1 example 2:

//Recurrence relation: T(n) = 4. T(n/2) + n;

a=4, b=2, and f(n) = n = $O(n^1 . log^0 n)$, so k=1, p=0.
and $log^4 2$ = 2, so $log^4 2$ > 1 holds true.

So, time complexity is $O(n^{log^a b}) = O(n^2)$

Similarly, for T(n) = 8 T(n/2) + n; time complexity $= O(n^3)$

Case 2 example 1 (if p > -1):

//Recurrence relation: T(n) = 2. T(n/2) + n;

a=2, b=2, and f(n) = n = $O(n^1 . log^0 n)$, so k=1, p=0.
i.e. p > -1 condition meets

So, time complexity is $O(n^k . log^{p+1} n) = O(n^1 . log^{0+1} n) = O(n \, log \, n)$

Case 2 example 2 (if p = -1):

//Recurrence relation: T(n) = 2. T(n/2) + n/logn;

a=2, b=2, and f(n) = n/logn = $O(n^1 . log^{-1} n)$, so k=1, p=-1.
i.e. p = -1 condition meets

So, time complexity is $O(n^k . log \, log \, n) = O(n \, . \, log \, log \, n)$

Case 2 example 3 (if p < -1):

//Recurrence relation: T(n) = 2. T(n/2) + n/ $log^2 n$;

a=2, b=2, and f(n) = n/ $log^2 n$ = $O(n^1 . log^{-2} n)$, so k=1, p=-2.
i.e. p < -1 condition meets

So, time complexity is $O(n^k) = O(n)$

Case 3 example 1 ($log^a b$ < k):

//Recurrence relation: T(n) = T(n/2) + n^2 ;

a=1, b=2, i.e. $log^1 2 = 0$ and f(n) = n^2 = $O(n^2 . log^0 n)$, so k=2, p=0.
i.e. p >=0 condition meets

So, time complexity is $O(n^2 . log^0 n)$ = $O(n^2)$

Case 3 example 2 ($log^a b$ < k):
//Recurrence relation: T(n) = 2.T(n/2) + $n^2 / logn$;

a=2, b=2, i.e. $log^2 2 = 1$ and f(n) = n^2 = $O(n^2 . log^{-1} n)$, so k=2, p=-1.
i.e. p < 0 condition meets

So, time complexity is $O(n^k)$ = $O(n^2)$