PERA

Fibonacci Series

fib(n)	0	1	1	2	3	5	8	13
n	0	1	2	3	4	5	6	7

$$fib(n) = \begin{cases} 0 & n=0 \\ 1 & n=1 \\ fib(n-2)+fib(n-1) & n>1 \end{cases}$$

recursive

```
int  fib(int n){
      if(n<=1)
         return n;
   return fib(n-2)+fib(n-1);
}
```

Bun, incelegelim ⟹

loop

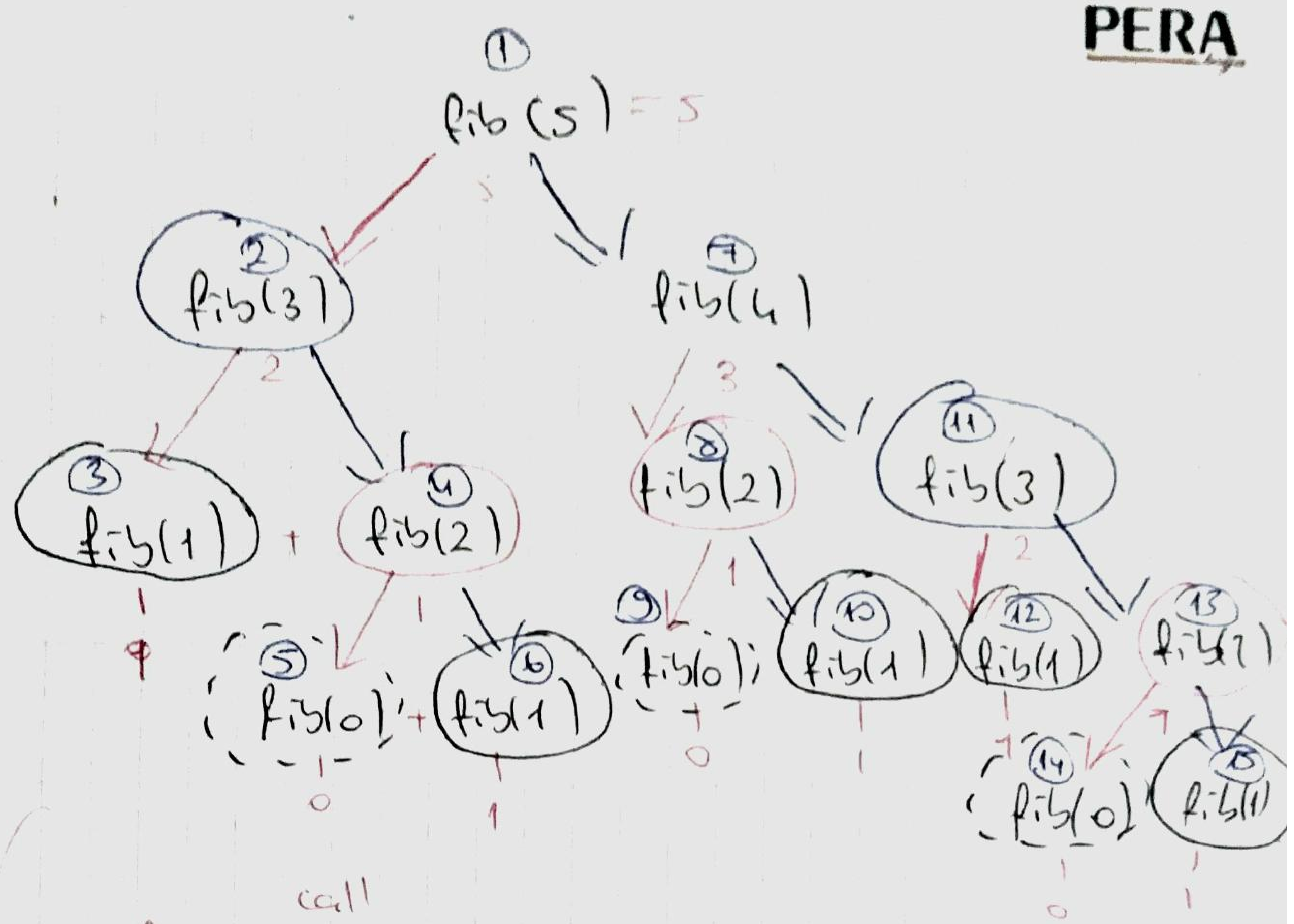
```
~~int  fib(int n){~~
~~     if(n<=1)~~
~~         return n;~~
```

```
       int fib(int n){
1    -  int t0=0 t1=1, s, i;
1    -  if(n<=1) return n;
n+1  -  for(i=2; i<=n; i++){
n-1  -      s=t0+t1
n-1  -      t0=t1;
n-1  -  }   t1=s
1    -  }  return s;
```

4n+1

O(n)

}



call

$f(5) \rightarrow 15$

$f(4) \rightarrow 9$

$f(3) \rightarrow 5$

$2\,fib(n-1)$ → anladığım kadarıyla n-2 yi n-1 yaptı ve diğeri ile topladı

$O(2^n)$

yuvarlak içine aldıklarımızla ilgili

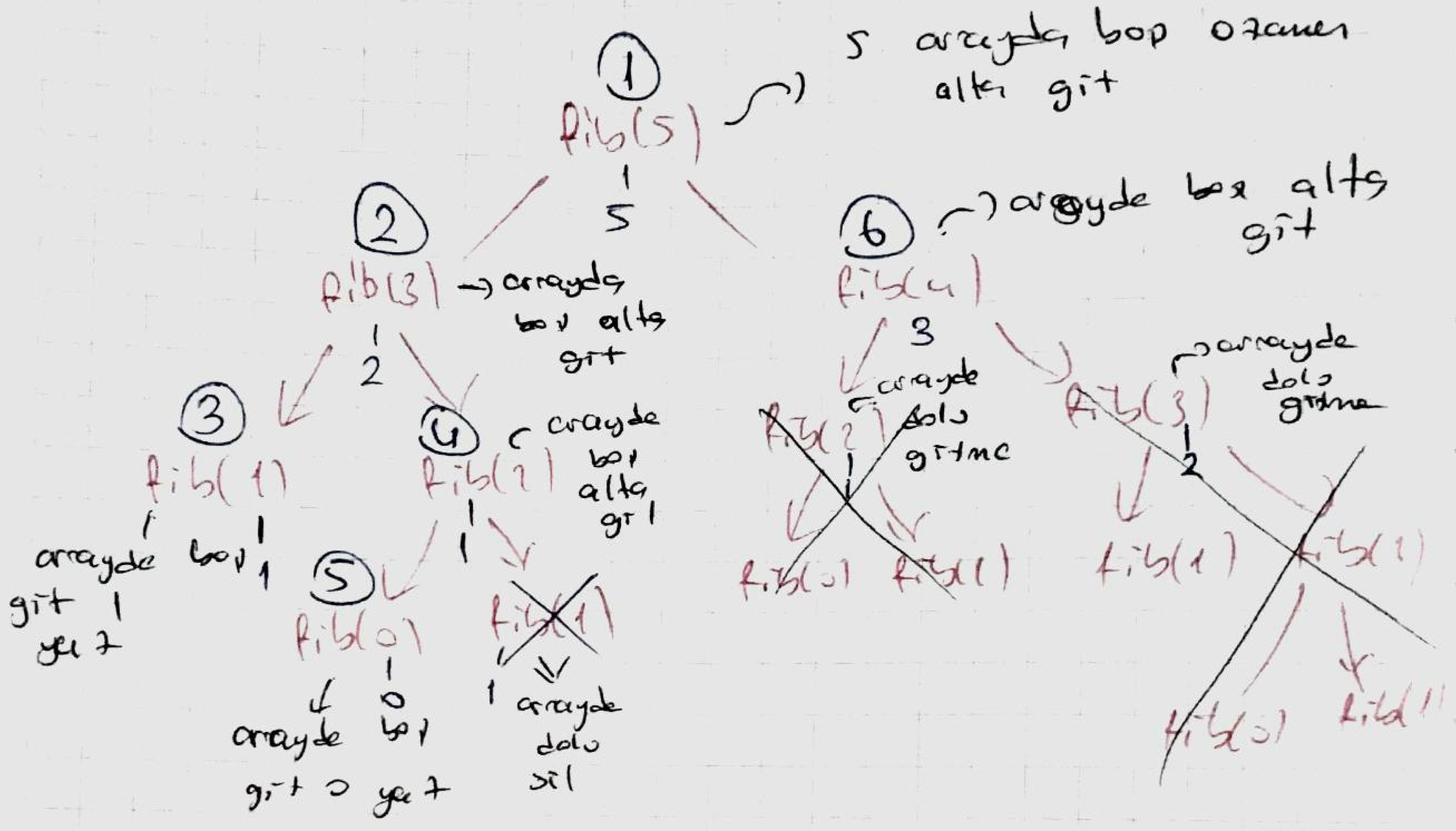
bir fonksiyonu bir kere çağırdığımızda başka bir yerde bir daha çağırmamıza (recursiveliğe) ihtiyacımız yoktur çünkü sonucu bellidir

bu recursivlere

EXCENSIVE REKURSIVE denir

array → global veya static olabilir

F	$-x_0$	$-x_1$	$-x_1$	$-x_2$	$-x_3$	$-x_5$	-1
	0	1	2	3	4	5	6



burada eğer daha önceden çağrılmış bir fon bir daha çağrılmaz direk değeri döndürülür

Toplamda 6 kere çağırma işlemi yapıldı

O(n)

bu işleme

Memoization

denir

PERA

```
int  f[10];
int  fib(int n) {
    if(n<=1) {
        f[n]=n
        return n;
    }
    else {
        if ( f[n-2]==-1 )
            f[n-2]= fib(n-2);
        if (f[n-1]==-1)
            f[n-1]= fib(n-1);
        return f[n-2]+f[n-1]
    }
```