dizaride bu xtilde tanimlan.
int x=0

```
int fun(int n){
    static int x=0
        if(n>0){
            x++;
            return fun(n-1)+x;
        }
        return 0;
}
int main(){
    int a=5
    printf("%d", fun(a));
return 0;
}
```

x=0 1 2 3 4 (5)

fun(5) → 25

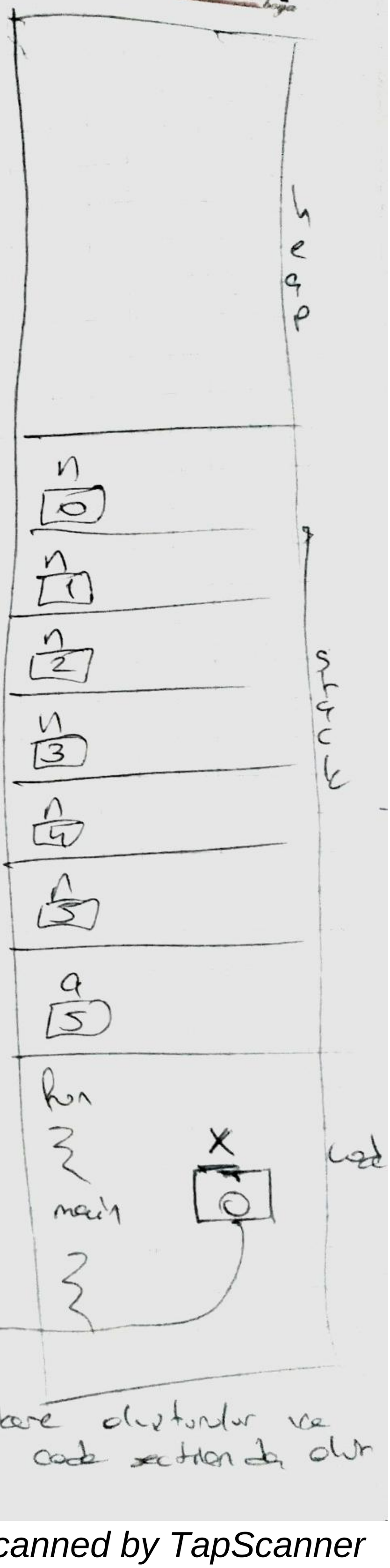
fun(4) + 5 = 25

fun(3) + 5 = 20

fun(2) + 5 = 15

fun(1) + 5 = 10

fun(0) + 5 = 5



PERA

Types of Recursion

1. Tail Recursion
2. Head Recursion
3. Tree Recursion
4. Indirect Recursion
5. Nested Recursion

1- tail Recursion

-> Anladığım kadarıyla Tail Recursionlar return zamanı kadar tekrar çağrıldıklarında) herhangi işlem yapmayan recursion fonlardır.

Tail Recursion

```
void fun(int n){
    if(n>0){
        cout<<n<<;
        fun(n-1);
    }
}

main()
fun(3);
```

Burada çağrı yapıldığında, herhangi bir işlem yapılmadığında tail recursion

```
void fun(int n){
    if(n>0){
        ~~cout<<n<<;~~
    fun(n-1)+n;
    }
}
```

Burada işlem zamanı sonra bekletmek zorunda ta gittiğinden tail fon değil

Kısaca anladığım kadarıyla bir kuyruk nasıl oluyorsa tail fonda aynı işlemde ediyor

Aynı sonucu karşılayan loop veya recursiondan loop daha performanslı,

```
void fun(int n){
    while(n>0)
        cout<<n<<;
    } n--;
}
```

sadece 1 defa oluşturulur

Time O(n) her ikisi içinde

Space O(1) O(n)

```
void fun(int n){
    if(n>0){
        cout<<n<<;
    } fun(n-1);
}
```

çünkü recursion her çağrıldığında, stackte yer alan ...

2- heap


```
void fun(int n){
    if(n>0){
        fun(n-1)
    }
}
```

X heapte çağrıdan sonra başka şey olmamalı,

```
void fun(int n){
    if (n>0){
        fun(n-1);
        printf("%d",n);
    }
}
```

✓

heap recorsion fakat çağrısından sonra her hangi bir işlem yapılmayacak.

Loop

```
void fun(int n){
    int i=1
    while (i<=n){
        printf("%d",i);
    }
}
```

Recursion

```
void fun(int n){
    if (n>0){
        fun(n-1);
        printf("%d",n);
    }
}
```

head recursionlar direk loopa çevrilemez ancak bir kaç ~~modifi~~ düzenleme yapılarak istenilen sonuç çıktısını alabiliriz

3. Tree Recorsiler

Linear Recursion

```
fun (n) {
  if (n>0) {
     ...
     fun(n);
     ...
  }
}   linear
```

Fonk ortadan olacak ve yalnızca bir defa çağrılacak

```
fun(n) {
  if(n>0) {
     ...
     fun(n)
     ...
     fun(n)
  }
}   trees
```

Tree recursion en az 2 kere çağırılır recursion kullanılır

Tree Recursion

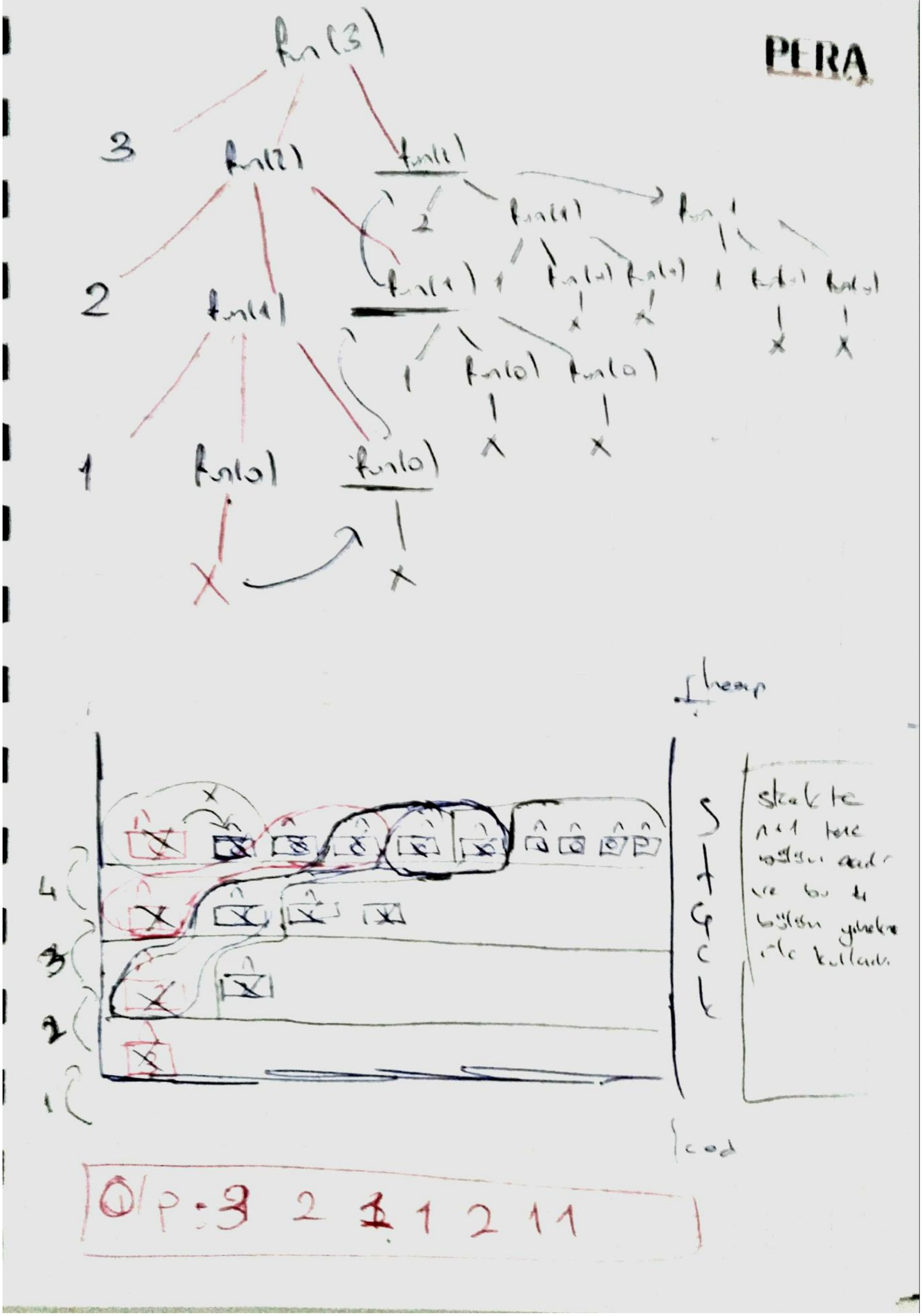
```
fun (int n) {
  if(n>0) {
     ...
     fun(n-1)
     fun(n-1)
  }
}   trees
```

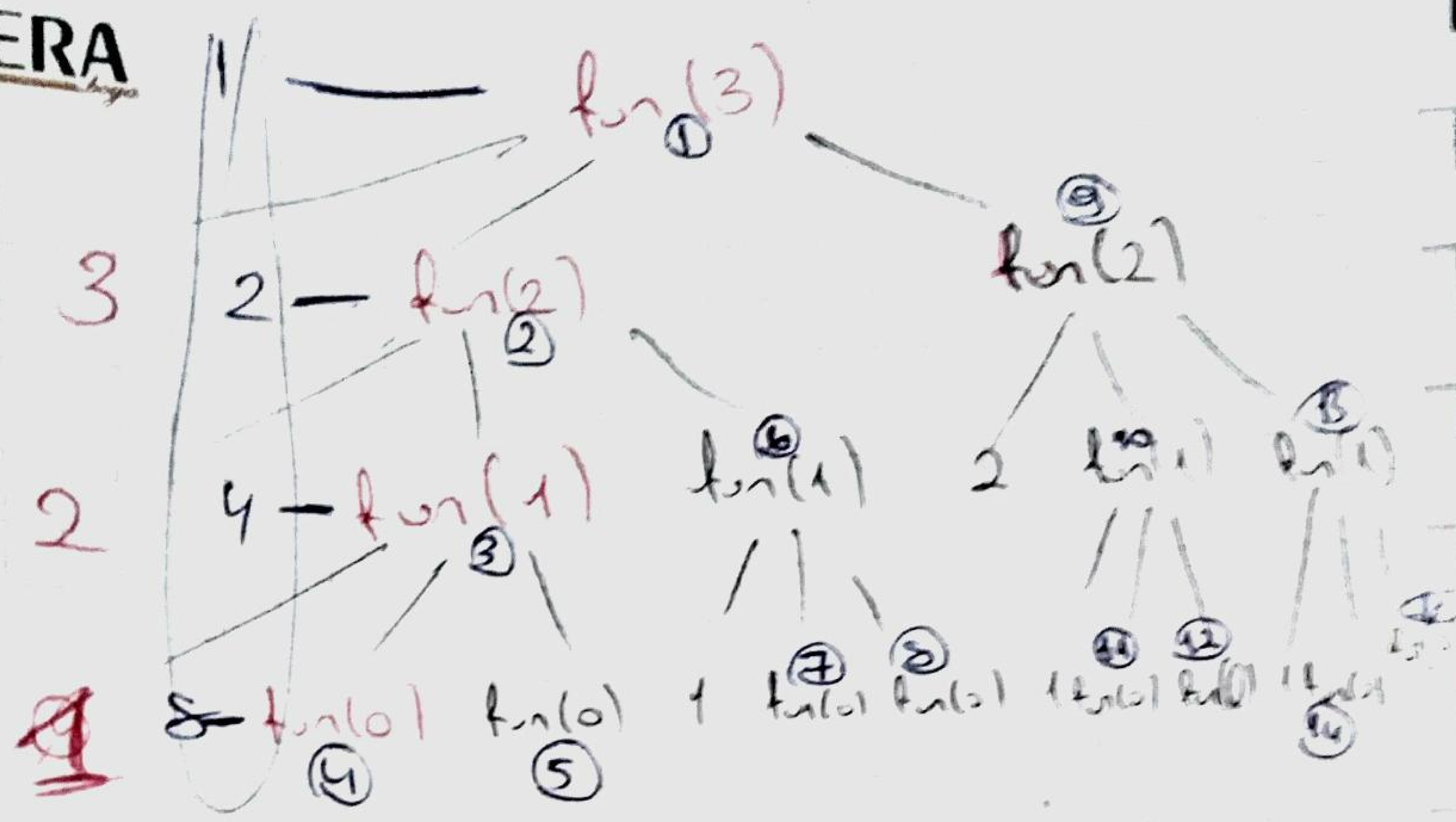
Not: trees için ekrek en az iki kere kendi yerlerde çağırılmalı gerek

```
fun (int n) {
  if (n>0) {
     cout << n << endl   (struck through)
     printf("%d", n);
     fun(n-1);
     fun(n-1);
  }
}
```

Stack kullanımı

fun(3)





$1 + 2 + 4 + 8 = 15$

$2^0 + 2^1 + 2^2 + 2^3 = 2^{3+1} - 1$ $\quad 2^{n+1} - 1$

$O(2^n)$

niye $O(2^{n+1})$ değil dersen buradaki n sonsuzluğu kadar

ve $O(2 \cdot 2^n)$ bunda n'in bir eleman kabul edilir ve

$O(2^n)$'dır

Time – $O(2^n)$

Space – ? → bilemeyiz ancak koda göre değişir